LY33-6012-1
File No. S360/S370-29

DOS
PL/I Transient Library:
Program Logic

Program Product

Program Number 5736-LM5
(This product is also distributed as
part of composite package 5736-PL3)

Feature Number 8052

IBM

Order No. LY33-6012-1, Revised October 1976 by TNL LN33-6180

Second Edition (June 1974)

This is a major revision of, and obsoletes, LY33-6012-0 and Technical Newsletters LN33-6060, LN33-6062, and LN33-6072.

This edition applies to Version 1, Release 5, Modification 0 of the DOS PL/I Optimizing Compiler, Program Product 5736-PL1, and to any subsequent version, release, and modification.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using the publication, consult the latest IBM System/370 Bibliography, GC20-0001, and the technical newsletters that amend the bibliography, to learn which edition and technical newsletters are applicable and current.

Requests for copies of IBM publications should be made to the IBM branch office that serves you.

Forms for readers' comments are provided at the back of the publication. If the forms have been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150. All comments and suggestions become the property of IBM.

PREFACE

The purpose of this publication is to summarize the internal logic of the modules contained in the DOS PL/I Transient Library program product. It supplements the program listings by providing descriptive text and flowcharts, but program structure at the machine instruction level is not discussed. The descriptive text is contained in part I of this publication, the flowcharts in part II.

Information on how to use this publication is contained in chapter 1 of part I; although the manual is intended primarily as a source of reference, the user should acquaint himself with the contents of chapter 1 before referring to any other chapter.

PREREQUISITE PUBLICATIONS

To make effective use of this publication, the reader must be familiar with the contents of:

DOS
PL/I Optimizing Compiler: Executicn Logic,
Order No. SC33-0019

ASSOCIATED PROGRAM PRODUCT PUBLICATIONS

Details of the DOS PL/I Resident Library are given in the follcwing publication:

DOS
PL/I Resident Library: Program Logic,
Order No.LY33-6011

The PL/I Optimizing Compiler, its facilities, and its requirements are described briefly in the following publication:

DOS
PL/I Optimizing Compiler:
General Information, Order No. GC33-0004

Details of the langauage implemented by the PL/I Optimizing Compiler are given in:

DOS
PL/I Optimizing Compiler: Language Reference Manual,
Order No. SC33-0005

The relationship between a PL/I program and the Disk Operating System is described in:

DOS
PL/I Optimizing Compiler: Programmer's Guide,
Order No. SC33-0008

Compile-time and execution-time messages for the PL/I Optimizing Compiler are documented in the following program product publication:

DOS
PL/I Optimizing Compiler: Messages, Order No. SC33-0021

RECOMMENDED SYSTEM AND SYSTEM CONTROL PROGRAM PUBLICATIONS

DOS Concepts and Facilities, Order No. GC24-5030

DOS System Control and System Service Programs,
Order No. GC24-5036

DOS Supervisor and Input/Output Macros, Order No. GC24-5037

DOS Data Management Concepts, Order No. GC24-3427

DOS System Generation and Maintenance, Order No. GC24-5033

Introduction to DOS/VS, Order No. GC33-5370

DOS/VS System Management Guide, Order No. GC33-5371

DOS/VS Data Management Guide, Order No. GC33-5372

DOS/VS System Control Statements, Order No. GC33-5376

DOS/VS System Generation, Order No. GC33-5377

DOS/VS Messages, Order No. GC33-5379

DOS/VS Data Management Services Guide, GC26-3783

DOS/VS Data Management Macro Instructions, GC26-3793

Tape and Disk Sort/Merge Program, Order No. GC28-6676

Contents

PART I: MODULE DESCRIPTIONS

FIGURES

CHAPTER 1: INTRODUCTION

The DOS PL/I optimizing compiler is supported by two library program products: the DOS PL/I Resident Library (program number 5736 - LM4). and the DOS PL/I Transient Library (program number 5736 - LM5). The resident library consists of modules that are selectively link-edited with the relocatable object module and hence become part of the executable program phase. The transient library consists of modules that are loaded dynamically at execution time.

This publication describes the DOS PL/I Transient Library. The resident library is described in _DOS PL/I Resident Library: Program Logic_.

THE TRANSIENT LIBRARY

The DOS PL/I Transient Library consists of about fifty program modules which reside on the core image library. Each module performs a single function or a number of closely-related functions and is designed as a single control section.

The transient library exists to reduce the storage requirements of the executable program. Each transient module is loaded only when a need for it arises, and the storage into which the module is loaded is freed when the module is no longer required. In addition to allowing modules that are not required concurrently to be overlaid, this technique enables the loading of modules to be governed by execution-time conditions: the error message modules, for example, are potentially required by any program; whether or not they are in fact required, can be determined only at execution time.

MODULE NAMING CONVENTIONS

CONTROL NAMES

Each module in the transient library is identified for documentation purposes by a unique 6- or 7-letter control name of the following form:

IBMabc[d]

The 4th letter (a) of the control name is either "B", "D", "F", or "H". These have the following meanings:

B -	The module is independent of the operating system and may be included in more than one program product.
D -	The module is written specially for and is dependent upon the Disk Operating System.
F -	This is equivalent to "B" but is used for CICS modules.
H -	The module is written specially for DOS/CICS only.

The 5th letter (b) of the control name specifies the functional area of the library to which the module belongs. The meanings of the letters that may appear in this position are listed in figure 1.1. It will be noted that not all of the functional areas are represented in the transient library; the complete list is given to enable the reader to determine the functional areas of various other modules that are referred to in this publication.

5th Letter of Control Name	Functional Area Identified
A	Computational routines (aggregates)
B	Computational routines (strings)
C	Conversion routines
E	Error-handling routines
I	Interlanguage communication routines
J	Miscellaneous supporting functions
K	Dump routines and miscellaneous supporting functions
M	Mathematical subroutines
O	Open/close routines
P	Program-management routines
R	Record I/O transmission
S	Stream I/O transmission

Figure 1.1. Identification of functional areas

The remaining letters of the control name (c for 6-letter control names, c and d for 7-letter control names) identify the module within its particular functional area. These letters are not necessarily mnemonic, but they have been chosen to give an indication of the module function wherever possible.

ENTRY-POINT NAMES

Entry-point names identify points within a module to which control is passed when the module is called.

They are derived from the module's control name by adding an eighth letter to identify the particular entry point within the module. Entry-point letters are usually allocated consecutively from the beginning of the alphabet.

Modules that have 6-letter control names are given two entry-point letters to make their entry-point names eight characters long.

CONTROL-SECTION NAMES

The control-section name of a library module is formed by adding "1" to its control name if it has seven letters, or "01" if it has six. For example, the control-section name of IBMBEOC is IBMBEOC1.

ARRANGEMENT OF MANUAL

Each of the remaining chapters in this manual deals with the modules in a particular functional area of the library. The chapters are:

Chapter 2: Error-Handling Routines
Chapter 3: Dump Routines

Chapter 4: Program-Management Routines
Chapter 5: Open and Close Routines
Chapter 6: Record I/O Routines
Chapter 7: Stream I/O Routines
Chapter 8: Conversion Routines

Appendixes giving lists of transient library modules and library macro instructions are included at the end of part I of this manual.

Part II contains flowcharts of all the transient modules that contain executable code. The flowcharts are identified by the 4th, 5th, 6th, and 7th letters of the module name; e.g., the flowchart of module IBMDREX appears on chart DREX. The charts are arranged in alphabetic order; part II may thus be used directly for reference.

Each chapter in part I begins with a brief overview of the modules described in the chapter, including details of the common features of the modules and an outline of their relationship with other modules. A full discussion of the part played by the transient library in the execution of a PL/I program is given in DOS *PL/I Optimizing Compiler: Execution Logic*.

Following the overview or introduction, the modules are described in alphabetic order by control names. Where modules have been further divided into functional subgroups, the alphabetic order of presentation is maintained only within the subgroups.

INFORMATION PROVIDED BY MODULE DESCRIPTIONS

The module control names, followed by a short title, are used as the headings for the module descriptions. For each module, information is provided under standard subheadings. The subheadings used and the types of information to be found thereunder are given in the following paragraphs.

Modules: If more than one module is described, the title of each module is given under this heading.

Function: Where this is not obvious from the title of the module, the information provided under this subheading consists of a brief statement of the main function of the module and, where appropriate, of each of the module entry points.

Method: Under this subheading, the method used in implementing the function or functions of the module is described. In general, the program logic and flow information presented in the module flowcharts is summarized and, in some cases, elaborated. Program structure at machine instruction level is not discussed, although reference to the symbolic names of control block fields is made wherever this may be considered helpful.

Linkage: The information provided under this subheading, is confined to a list of those registers containing parameters for the module in question and details of the parameters passed.

Error and Exceptional Conditions: A brief summary of all the error and exceptional conditions that the module is capable of detecting is given under this heading.

Calls: A list of modules that may be invoked by the module described is given under this heading.

Called By: A list of modules capable of invoking the module described is given under this heading.

PROGRAM LISTINGS

The information contained in the following paragraphs may prove helpful to the user of a program listing.

REGISTER NAMING CONVENTIONS

In the program listings, registers are referred to symbolically by prefixing the register number by "R" for general registers, and by "F" for floating point registers, e.g., R0, R5, F4. Exceptions to this general rule are shown in figure 1.2.

General Register	Symbolic Name	Remarks
3	AR	Normally used as addressing (base) register
10	RX	
11	RY	
12	CR	Points to task communications area (TCA)
13	DR	Points to dynamic storage area (DSA)
14	LR	Link register
15	BR	Branch register

Figure 1.2. Symbolic register names

FLOWCHARTING STATEMENTS

Certain statements in the program listings are preceded by */* and followed by */. These statements are used in the production of the module flowcharts given in part II of this publication. A typical flowcharting statement might appear in the listing as:

```
*/* B4 P SAVE REGISTERS */
```

The flowcharting statements serve as general comments on the sections of executable code that follow them in the listings; they are also useful in locating the section of code that corresponds to a particular box on a module flowchart.

PROGRAM FLAGS

In some program listings, the follcwing symbols are used in column 38 to highlight the breaks in sequential flow through the module and to augment the comments:

E	Branch to another module
I	Branch to internal closed subroutine
R	Return to caller
/	Branch to internal error routine
=	Switch set
?	Switch test or other decision

LIBRARY MACRO INSTRUCTIONS

Each macro instruction is identified by a 7-letter or 8-letter name of the following form:

IBMaXbc[d]

The 4th letter (a) of the name is "B", except for those macro instructions that are designed for use only under DOS, when the letter "D" is used.

The 5th letter of the name is always "X"; it identifies the name as that of a library macro instruction.

The final two or three letters are mnemonic of the function. A list of library macro instructions, together with brief descriptions of their functions, is given in Appendix B.

Debugging Macro Instructions

The program listings of many library modules contain debugging macro instructions, which were used during the development of the modules. These macro instructions are not expanded in the listings, nor do they have any corresponding code in the actual library modules. They can, however, be reactivated to provide debugging facilities. Information on how to do this is given in appendix C.

The transient error-handling routines are concerned primarily with the construction and printing of messages. On CICS systems however, those error routines that are usually resident are included in the PL/I Transient Library package.

The messages produced by the error message package are not necessarily error messages; they may relate to PL/I conditions that are not errors, (e.g., ENDFILE) or to conditions that have been deliberately raised by the programmer. In this chapter, however, the term "error message" is used as a generic term for all messages produced by the error message package.

The error messages handled by the error message group of modules can be divided into two basic types: system action messages and SNAP messages.

In general, system action messages relate to execution-time error conditions which have no equivalent PL/I on-condition or for which there is no established on-unit. The messages consist of a message number and a description of the error, followed by the location in compiled code at which the error occurred. The location is expressed as an offset from the start of the current block; the corresponding statement number in the source program is also given if the compiler GOSTMT option was in effect when the program was compiled.

SNAP messages relate only to PL/I on-conditions. They are produced if the source program contains an ON statement specifying SNAP for the condition. These messages identify the condition and the location in compiled code at which the condition was raised. They provide a calling trace which lists all procedures, begin blocks, and on-units through which control has passed, back to the original call in the main procedure. The trace gives the statement number of the call or, if the option NOGOSTMT applies, the offset of the call from the appropriate entry point. Details of the flow of control through the program are also given if the FLOW option has been specified.

If both SNAP and SYSTEM are specified in an ON statement, the resulting message consists of the system action message for the condition, followed by the calling trace part of the corresponding SNAP message.

Error messages are created and put out on SYSPRINT or at the console by a routine consisting of two library modules: IBMDESM and IBMDESN. The routine is divided between the two modules in order to reduce the storage requirements of the error message package, IBMDESM being loaded first and subsequently being overwritten by IBMDESN.

The error message package is always entered from the resident error-handling module IBMDERR. Whenever an error message is required, IBMDERR loads and calls IBMDESM and passes to it the address of the dynamic storage area (DSA) belonging to IBMDERR. This DSA contains four fields that are used by IBMDESM. The first two, HEC1 and HEC2, contain an error code that defines the error that has occurred. The third, HEFL, contains flags specifying the type of message that is required; that is, SNAP or SYSTEM or both. The fourth, HESW, contains information for use when dynamic qualifiers are needed for the message.

IBMDESM is supported by a number of message text modules which provide the texts for system action messages, and by a module which translates the error code obtained from IBMDERR into a PL/I on-code for inclusion in the message. The message text modules all have names of the form

IBMBETx, where x is alphabetic. The on-code module is IBMBEOC.

Messages are transmitted on to the diagnostic file, which is addressed via field ABTS in the diagnostic file block (DFB). The standard file SYSPRINT is used as the diagnostic file whenever possible. If, at any point in the program, SYSPRINT is opened as a PRINT file, the error message entry point address of the SYSPRINT transmitter (IBMDSTV, IBMDSTF, or IBMDSTU) is loaded into field ABTS by the open module IBMDOPA. In these circumstances, all diagnostic information is transmitted to SYSPRINT.

If the diagnostic file has not already been opened when a diagnostic output is required, module IBMDEDO is loaded and invoked. Provided that SYSPRINT has been declared, this module calls the open/close package in an attempt to open SYSPRINT as a PRINT file. If this attempt is unsuccessful, or if SYSPRINT is already open without the PRINT attribute, the console transmitter (IBMDEDW) is loaded and its entry point address is put into ABTS. If messages are routed to the console, only system error messages are transmitted; SNAP messages and traces cannot be obtained.

MODULE DESCRIPTIONS

IBMDESM - Error Message Module (phase 1)

Function

To form the first part of a diagnostic error message.

Method (chart DESM)

The module uses the error information in the DSA of the resident error handler IBMDERR to determine the type of message that is to be printed; that is, SNAP or SYSTEM or SNAP SYSTEM.

For system action messages, the module creates the message in a message area as follows:

1. The module examines the two-byte error code to determine whether or not the condition is an on-type. (For ON conditions the first byte of the error code, HEC1, is less than hexadecimal 50). If it is an on-type, the condition name is placed in the message.

2. Module IBMBEOC is loaded and invoked to calculate the on-code and the returned value placed in the message.

3. The required message text module (IBMBET*) is loaded and the message number and main text of the message are moved to the message area. The required text module and the location of the required message within it are determined by performing arithmetic on the error code.

If a system action message is not required, only step (1) above is executed.

Control is now transferred to the code at the beginning of IBMDESM and a LOAD macro is issued to load phase 2 of the error message module: IBMDESN. IBMDESN is loaded into the storage previously occupied by

IBMDESM; control thus passes in normal sequence to IBMDESN.

Linkage

DR = A(DSA of the resident error-handler IBMDERR)

Calls

IBMBEOC - On-code evaluation.

Called By

IBMDERR - Resident error handler.

IBMDESN - Error Message Module (phase 2)

Function

To complete the error message initialized by module IBMDESM and to print it.

Method (chart DESN)

The main purpose of IBMDESN is to find the location in compiled code at which the error or condition has occurred and, for SNAP messages, to provide a calling trace. On entry to the module, register R7 points to the end of the message constructed by IBMDESM in the output buffer. The way in which IBMDESN constructs the rest of the message depends on whether it is a system-action message or SNAP message.

System Messages:

1. If there is a filename associated with the condition, the words ('ONFILE' = filename) are moved into the output buffer.

2. If the CONDITION condition has been raised, the condition name is moved into the message.

3. The location of the error or condition in compiled code is found by searching back through the DSA chain until a compiled code DSA is found. The link register save slot (OFLR) in this DSA contains the absolute address of the compiled code instruction that would have been executed next had the error or condition not occurred. The absolute location of the error or condition in compiled code can thus be computed.

   The entry point of the procedure, begin block, or on-unit in which the error has occurred is held in the branch register save slot (OFBR) in the previous DSA. IBMDESN picks up this address and uses it to convert the absolute location of the error into an offset within the current block. The words 'AT OFFSET', (or 'NEAR OFFSET' if the interrupt is imprecise), followed by the offset, are then moved into the message.

4. If there is a statement number table associated with the current block, it is used to find the number of the statement that corresponds to the previously-calculated offset. A bit in the second flag byte of the compiled code DSA indicates the

form in which statement numbers are held. Statement numbers will be either four bytes long (TSO number format) or two bytes long (PL/I statement number format). The words 'IN STATEMENT', followed by the statement number, are then moved into the message.

5. The type of compiled code DSA is next determined; if it is a begin block or on-unit, the routine chains back until a procedure DSA is encountered. The words 'IN PROCEDURE WITH ENTRY' followed by the entry point name are then placed in the line. If the SNAP option is in effect for system messages and the compiled code DSA is not a procedure DSA, either 'A BEGIN BLOCK' or 'A xxxx ON-UNIT' is placed in the message and the line is printed. The next compiled code DSA is then found and the trace procedure repeated until the major dummy DSA has been found.

SNAP Messages:

For SNAP messages, the message is constructed as described above from 3 onwards for SNAP system messages.

All printing is performed by a subroutine which calls the transmitter addressed by field ABTS of the diagnostic file block (DFB). If this field is zero, module IBMDEDO is invoked to open the diagnostic file, load a transmitter, and initialize ABTS. If the transmitter addressed by ABTS is the console transmitter, IBMDEDW, this is indicated by a flag in the DFB, and SNAP messages (or the trace part of SNAP SYSTEM messages) are not transmitted to the console. If, on return from the stream I/O transmitter for SYSPRINT, a flag in the DFB has been set on (indicating that a TRANSMIT condition has occurred), IBMBEDC is again invoked to load the console transmitter and update the DFB.

for SNAP messages where FLOW was specified as a compiler option, the FLOW table's contents are printed at the end of the SNAP message in the form given below (where nn and mm are statement numbers).

```
nn   TO   mm   IN (blockname)
nn   TO   mm   IN (blockname)
nn   TO   mm   IN (blockname)
nn   TO   mm   IN ON 'XXX'
```

Linkage

DR = A(save area of IBMDESM)

Calls

IBMDEDO - Open diagnostic file.
IBMDSTV,IBMDSTF,IBMDSTU, or IBMDEDW - Diagnostic file transmitter.

Called By

Control passes in normal sequence from IBMDESM.

IBMBEOC - On-Code Calculator

Function

To translate two bytes of error code into the PL/I on-code.

Method (chart BEOC)

PL/I on-codes vary from 0 to 10,000, this range being split up into a number of scopes. Each type of error has an on-code in a set scope and also a unique value in the first byte of the error code.

The errors can also be classified into on and non-on types. On-types have error codes less than 48 and cn-codes less than 1000. Non-on types have error codes descending by twos from 255 and on-codes ranging from 1000 to 10,000.

The lowest number in each on-code scope is tabulated in IBMBEOC in two tables of halfwords, the order of tabulation being such that the basic on-code value can be accessed by means of a simple arithmetic cperation on the first byte of the error code.

IBMBEOC finds the error code by picking up the chain back field in the current ONCA and obtaining the errcr code from the ONCA thus addressed. If the error is an on-type, the basic on-code value is located in the table by doubling the value in the first byte of the error code; if it is a non-on type, the basic on-code value is located by subtracting the value of the first byte of the errcr code from 255.

The final value of the on-code is obtained by zeroing bits 0, 1, and 2 of the second byte of the error code and adding the result to the basic on-code value just obtained.

Linkage

R1 = A(halfword target for on-code value)

Called By

IBMDESM	-	Error message module.
IBMDKTR	-	Dump trace routine.

IBMBET* - Message Text Modules

Function

To provide the standard text for execution-time error messages. There are eight message text modules:

IBMBETA	-	Miscellaneous non-on messages (1).
IBMBETB	-	Miscellaneous non-on messages (2).
IBMBETC	-	Miscellaneous non-on messages (3) and conputational ncn-on messages.
IBMBETI	-	I/O non-on messages.
IBMBETO	-	ON messages (1).
IBMBETP	-	ON messages (2).

IBMBETQ - ON messages (3).

IBMBETT - EVENT messages.

Method

Each message text module comprises a number of tables of error messages which are used by the error message module IBMDESM. Each table is addressed from a displacement table at the head of the module, and the individual messages in their turn are addressed by an offset from the start of each particular table.

Each message is preceded by one byte, giving the length of the message, and two bytes, giving the message number. The message required is determined by IBMDESM from information in the error code.

The message text modules do not contain any executable code.

IBMDEDO - Open Diagnostic File

Function

To connect the message generating modules to a stream I/O transmitter.

Method (chart DEDO)

The connection between the message generating modules and the stream I/O transmitter is made via the diagnostic file block (DFB), which is addressed from the TCA.

If SYSPRINT has not been either explicitly or implicitly declared, or if SYSPRINT has been declared with file attributes other than PRINT, it cannot be used for diagnostic output. In these circumstances IBMDEDO loads the console transmitter module IBMDEDW into storage and puts its address into field ABTS of the DFB. It also sets flag AWTO in field AFLA of the DFB to indicate that the address in ABTS is that of the console transmitter.

If SYSPRINT has been declared but is not open, the resident open/close bootstrap module IBMDOCL is called to open it. On return a test is made to see whether or not SYSPRINT has been successfully opened as a PRINT file. If it has not, IBMDEDO loads the console transmitter as described above.

Note: If the open module opens SYSPRINT as a PRINT file it sets the address of the message entry point of the SYSPRINT transmitter into ABTS.

Calls

IBMDOCL - Resident open/close bootstrap.

Called By

IBMDESN - Error message module.

IBMDEDW - Console Transmitter

Function

To transmit messages to the console.

Method (chart DEDW)

The console transmitter module loads the message string into a buffer, sets up a transmit CCW for the message, and issues an execute channel program (EXCP) macro. It then issues a WAIT macro, and returns control to the caller when channel end is received.

If the string is the first line of a message, the module accesses the communications region to find the jobname and inserts this name after the message identifier.

Linkage

R1 = A(string locator for message)
R2 = A(flag byte for 1st line)

Called By

IBMDESN - Error message module.

IBMDPIF - Operation Exception Checking (Systems without Floating-point support)

Function

To check whether an operation exception has been caused by an attempt to use a floating-point instruction.

Method (chart DPIF)

IBMDPIF is loaded at program initialization if there is no floating-point hardware support.

If an operation that causes an exception is a floating-point instruction, the error code in the on-condition handler's DSA is modified to indicate that floating-point arithmetic is not supported.

Linkage

R12 = A(TCA)
R13 = A(DSA of on-condition handling routine)

Called By

IBMDPII - Program ISA Initialization.

IBMDESY - Error System Action

Function

To provide a PLIDUMP if the option DUMP applies, when FINISH is raised as system action for the ERROR condition.

Method

The module is called when FINISH is raised as system action for the ERROR condition. If the option DUMP applies to the step, then provided there is enough space the module calls IBMDKMR, with the parameter HB, to produce a PLIDUMP. Finally, IBMDESY returns to the caller.

Linkage

```
  LR =   A(Return to caller)
  BR =   A(Start of IBMDESY)
```

Calls

IBMDKMR - Dump control module.

Called By

IBMDERR - Error-handler (resident).

IBMFEFC - Print COUNT Tables

Function

To print the statement frequency count tables at program termination.

Method

The count output is transmitted to the transient data queue by means of the stream output transmitter. The amount of storage required to hold one page of output is calculated and obtained. This is necessary because three columns of tables are printed on each page.

The page is blanked and page headers and column headers, including the name of the external procedure of the first table, are constructed.

Each table consists of a series of branch-counts, each of which is associated with a statement number. If a statement is the target of a branch during execution of the PL/I program its branch-count is increased by one. If a branch is taken from the statement then the branch-count of the next statement is decreased by one. Thus whatever the number of times the first statement in a procedure was executed, its count-value, is the same as its branch-count. The number of times the next statement was executed is the count-value of the first plus its own branch-count. Thus the count-value of any statement is the count-value of the previous one plus its own branch-count. The latter value may in fact by negative.

By scanning the table for non-zero branch-counts, ranges of statements executed the same number of times are calculated. The statement numbers of the start and end of each range of statements are placed in the page together with the count-value for the range. If the range consists of a single statement, only one number is used. If the count-value is zero the range is not included in the page but is saved in the table so that it may be printed separately.

Checks are made each time a range is constructed in the page for the end of a column and the end of the page. When the page is full each line is printed and the page blanked.

When all tables have been dealt with they are examined again for unexecuted statements. Ranges of unexecuted statements are sorted in the tables during the first processing stage whenever zero count-values are found. The unexecuted statement ranges are then placed in the page.

When all the tables have been finally dealt with the last page of output is transmitted and the core occupied by the tables is freed. Return is then made to the caller.

Called By

IBMFPIRA

External Modules

IBMFSTVA - stream transmitter

IBMFESM - Error Message Routine Phase 1

Function

Module IBMFESM is the first phase of the error message routine called by the error handling module. It constructs messages of the following form in an output buffer:

```
    "System message number ONCODE = NNN (not for SNAP messages)
     condition name CONDITION RAISED (on-conditions only)
     system message text (not for SNAP message)"
```

The module then transfers control to the second phase of the error message routine, module IBMFESN.

Method (chart FESM)

The module examines a two-byte error code in the resident error handler's dynamic storage area (DSA) to determine whether the condition is of the on-type. (The first byte of the error code for on-conditions is less than hexadecimal 1B.) Arithmetic is then performed on the error code to locate the condition name, for on-conditions, and the appropriate system text module. For SNAP messages, control then passes to IBMFESN.

Module IBMBEOC is invoked to calculate the on-code; this module is not, however, called if the message to be printed is one of the warning messages printed following a multiple exception imprecise interrupt, or is a non-system message. The specific message, in the message text module brought into main storage by the DFHPC TYPE=LOAD macro instruction, is located by performing arithmetic on the second byte of the error code. After the required message is moved into the output buffer and the message text module is deleted, control passes to IBMFESN by means of a LOAD and branch.

Called By

IBMFERR(via LOAD and call) - error handling module.

External Modules

1. A subroutine of IBMFPIR to acquire a new library workspace and on-communications area.
2. (TOVF in TCA) - Stack overflow routine.
3. IBMBEOC - On-code calculator.

Modules Loaded

IBMBETx - Message text module.(x=A,B,C,I,O,P,Q,or T)

Linkage (On Exit)

R10(RX) = A(Start of message in output buffer)

R7 = A(Current end of message in output buffer)

Exit

Branch to IBMFESN - Error message routine Phase 2.

IBMFESN - Error Message Routine Phase 2

Function

Module IBMFESN is the second phase of the error message routine. It completes the message constructed by IBMFESN, and calls the message transmitter to put out the message.

Method (chart FESN)

On entry to the module, Register 7 points to the end of the message constructed by IBMFESM in the output buffer. The remainder of the message, to be constructed by IBMFESN, depends on whether it is a system-action message or a SNAP message.

1. System-action messages

   a. If there is a filename associated with the condition, then for most error messages the characters ('ONFILE'=" are moved into the buffer, followed by the filename. For some messages however the characters "FILE=filename" are moved into the buffer instead of those given above.

   b. If the CONDITION condition has been raised, the condition name is moved into the message.

   c. The address of the next instruction to be executed in compiled code is either in the old program status word (PSW), if the interrupt occurred in compiled code, or is the address of the next instruction following the call to the library, if the interrupt occurred in a library routine. In either case, the address is obtained from the Register 14 field of the compiled code save area. If the interrupt occurred in compiled code, the instruction length code in the PSW is tested to see if the interrupt was "imprecise" and, if so, the words "NEAR OFFSET" are placed in the message followed by the offset of the interrupt; otherwise the words "AT OFFSET" are used.

   d. If there is a statement number table associated with the current dynamic storage area (DSA), it is located. The offset from the main entry point of the next instruction to be executed in compiled code is computed, and the statement number table scanned for the number corresponding to this offset. The words "FROM STATEMENT" followed by the statement number are then added to the message.

   e. The type of compiled code DSA is next determined; if it is a begin block or on-unit, the routine chains back until a procedure DSA is encountered. The words "IN PROCEDURE" followed by the entry point name are then placed in the line; the entry point name is obtained from the bytes immediately preceding the entry point. If the SNAP option is in effect for system messages and the compiled code DSA is not a procedure DSA, either "A BEGIN BLOCK" or "A xxxx ON-UNIT" (where XXXX is a four character abbreviation) is placed in the message and the line is printed. The next compiled code DSA is then found and the trace procedure repeated until the major dummy DSA has been found.

2. SNAP messages

   For SNAP messages, the message is constructed as described from 1.c. onwards for SNAP system messages.

All printing is performed by a subroutine which calls the transmitter addressed via the CICS appendage. This field points at a bootstrap which loads the transmitter if it is not in storage.

Called By

IBMFESM (via LOAD and Branch) - Error message routine Phase 1.

Linkage (On Entry)

R10(RX) = A(Start of message in output buffer)

R7 = A(Current end of message in output buffer)

Exit

Return to IBMFERR (the calling routine of IBMFESM)

IBMFERR - Error Handler

Function

To identify execution-time errors or conditions and to take the appropriate action. The module has three entry points:

IBMBERRA: Program check interrupts and abends.

IBMBERRB: Conditions and errors detected by code.

IBMBERRC: Program check interrupts and abends while control is in IBMBERR, modules it calls, or some vulnerable housekeeping functions.

Method (chart FERR)

Program Check Interrupts:

Program check interrupts are handled by entry point IBMBERRA. PL/I interrupt handling is established by means of a DFHPC TYPE-SETXIT macro (issued in module IBMFPIR).

On entry to IBMFERRA, information on the interrupt is contained in the CICS TCA. On entry, module IBMBERR saves the contents of registers 3 through R12 and the second word of the PSW (from PIE) in the current DSA. If the A(interrupt handler) slot in the TIA is non-zero, indicating that non-standard action is required, then IBMFERRA branches to the address.

IBMFERRA then changes the address in the TIA to that of IBMFERRC. This ensures that if an interrupt occurs while control is in IBMBERR, the second interrupt will cause control to be passed to IBMFERRC. A flag is set to indicate that the module was entered at entry point IBMBERRA.

Processing the Interrupt

From the abend code, the module creates a two-byte PL/I internal error code. If the interrupt corresponds to a PL/I ON condition, the floating point registers are saved in case return is to be made to the point of interrupt. If the interrupt is due to floating point underflow, the double word in which the register that underflowed was stored is set to true zero.

If a fixed-point, decimal, or exponent overflow or fixed-point divide has occurred, this may correspond to SIZE in the original program rather than to FIXEDOVERFLOW or ZERODIVIDE. If this is the case then compiled code will have set a bit in the interrupt qualifier byte in the TCA. If this bit is set, IBMBERR creates a code for SIZE and sets the 'ignore' bit in the qualifier byte in the TCA. If this bit is set already and one of the above exceptions has occurred. IBMBERR returns to the point of interrupt. IBMBERRA now branches to the main condition-handling logic described below.

Conditions Detected by Code

When a condition is to be raised by the library or compiled code, IBMFERRB is passed an interrupt control block containing a two- or four-byte code indicating which condition to raise.

For PL/I ON conditions, the code consists of four bytes. The first byte of this code is the same as the code which is placed in the ON cell or the dynamic ONCB when an ON statement for the condition is executed. The second byte gives the particular situation which caused the condition to be raised. The third and fourth bytes contain flags indicating which ON functions and pseudovariables are valid for the condition. If the condition is a qualified condition, the interrupt control block also contains a qualifier.

For conditions for which there is no PL/I ON condition, and for which the action is to comment (i.e., put out a message) and raise the ERROR condition, the code is two bytes long. The first byte gives the class of the condition (e.g., I/O, computational) and the second byte gives the particular error in that class. The values of the first byte of these codes run from X'FF' downwards. If entry has been made via entry point IBMFERRA a code of this type will have been created.

Handling the Condition

IBMFERR determines whether it is dealing with an ON condition or an error condition by testing the value of the first byte of the code passed to it. The code is moved to the ONCA; if the condition is an ON condition the four bytes of the code are moved, otherwise two bytes are moved, the other two bytes in the ONCA being set to zero. The action in the case of a non-ON condition is to first of all print a diagnostic message. The message is produced by the transient module IBMFESM, which is dynamically loaded and invoked.

One return from the transient message module, a code for the ERROR condition is created and the action taken is that for ON conditions.

There are however some error codes which are not really "error codes", as all they require is the message to be printed. In such cases, IBMFERR does not raise any conditions but returns immediately to the caller.

For each ON condition, IBMFERRA contains an action byte containing information on the course of action that is to be taken. The format of the action byte is as follows:

```
Bit 0  =0 Condition may be enabled or disabled.
        1 Condition always enabled.

Bit 1  =0 No comment on standard system action (SSA).
        1 Comment on SSA.
Bit 2  =0 Continue on SSA.
        1 Raise ERROR on SSA.

Bit 3  =0 Return to point of interrupt on return from on-unit.
        1 Special action on return.
```

```
Bit 4  =0 Non-qualified condition.
        1 Qualified condition.
```

Bits 5-7 unused.

When an ON condition is raised, IBMBERR first of all determines the enablement of the condition. Bit 0 of the action byte for the condition determines whether there is an entry in the enable bits of the DSA. If so, IBMBERR tests the relevant bit. If it is 'one', i.e., the condition is disabled, an immediate return is made to the point of interrupt. If there is no entry in the enable bits for the condition it is always enabled.

Further tests must be made in the case of the CHECK condition since a 0 value in the enable bit only means that some CHECK condition is enabled but does not say anything about the particular CHECK which has been raised. The tests are made as follows:

A search is made down the static chain of DSAs. The dynamic ONCBs in each DSA are tested for one containing the code for CHECK and the correct qualifier. If one is found then the enablement is determined from the enablement bit in the ONCB. If no match is found in a DSA, the enable bits for that DSA are tested to determine if there is a CHECK or NOCHECK prefix without a list in the associated block. If not, testing continues with the next DSA in the static chain. If so, a second bit in the current enable bits gives the enablement.

If the condition is found to be enabled, the next step is to determine whether or not it is established. This is done as follows:

If the condition is unqualified, the list of ON cells in each DSA in the dynamic chain is searched by means of a TRT instruction, using the special table in the TCA, for a code matching the first byte of the code passed to IBMFERR. If a matching ON cell has not been located when the dummy DSA is reached, standard system action is taken for the condition, the action being taken from the action byte. If a match is found, the corresponding static ONCB is located.

If the condition is qualified, the chain of dynamic ONCBs in each DSA in the dynamic chain is scanned for one containing the correct code and qualifier. If a match is not found, standard system action is taken except for the CHECK condition. If the matching dynamic ONCB specifies that the condition is not established then the search is continued. If no matching ONCB is found for CHECK, tests must now be made to see if there is an establishment for unqualified CHECK. This is done by searching the ON cells as for an unqualified condition.

When a matching ON cell or a matching ONCB which does not specify unestablished has been found, the ONCB is tested as follows:

1. If SNAP is specified, SNAP messages must be printed. This is done by loading and invoking the transient module IBMFESM.

2. If SYSTEM is specified, standard system action is taken.

3. If there is a GO TO only in the on-unit, then GO TO is performed without entering the on-unit.

4. If there is a null on-unit, the action which is performed on return from an on-unit is taken.

5. If there is an on-unit which is neither null nor consists only of a GOTO statement, then IBMFERR invokes it.

Before branching to the on-unit, IBMFERR must perform some housekeeping duties. The contents of the interrupt qualifier byte in the TCA are saved in IBMFERR's DSA and the qualifier byte is set to zero. IBMFERR then invokes the Get Library Workspace routine to obtain a new LWS. Tests must be made to see if the current ONCA is correctly set up for any ON built-in functions or pseudovariables which may be used in the on-unit. If the condition was signaled, the locators for ONSOURCE, ONKEY, and DATAFIELD are set to give null strings, the string locator for ONCHAR is set to give a blank, and the value of ONCOUNT is set to zero.

Having performed the necessary housekeeping, IBMFERR invokes the on-unit having first loaded register R5 with the address of the DSA in which the matching On cell or dynamic ONCB was found.

Before leaving IBMFERR to enter an on-unit or perform a GO TO, the interrupt linkage to IBMFERRA must be restored. This involves replacing the address of IBMFERRC in the TIA by 0. If there is no GO TO out of the on-unit the address of IBMFERRC is replaced on return to IBMBERR.

On return from an on-unit the interrupt qualifier is restored and the new LWS freed. The action byte for the condition is tested to see if a return is to be made or if special action is to be taken. If the action specified is return, IBMBERR returns to the point of interrupt.

The cases where action other than return must be performed are:

1. ERROR - The FINISH condition is raised.

2. FINISH - If the condition was raised following a STOP statement, or following the raising of ERROR or the normal termination of the program, the task is terminated by setting a return code in the return code slot in the TCA and then entering the GOTO code in the TCA in order to perform a GOTO to the task initialization routine IBMBPIR. If FINISH was raised by the SIGNAL statement, IBMFERR returns to the caller.

3. CONVERSION - Unless the condition was signaled, then a test is made on return from the on-unit to determine whether ONSOURCE or ONCHAR pseudovariables were used in the on-unit. If not, ERROR is raised. If either was used, control is passed to the address contained in the retry slot in the ONCA.

4. ENDPAGE - A return code is set in register BR to indicate that an on-unit was entered.

System Action

System action is performed if no matching ON cell or dynamic ONCB was found in the DSA chain or if the SYSTEM flag was set in the ONCB located as the result of a match.

Performing system action for conditions other than the CHECK condition normally involves printing a system action message. The action byte is tested to see if a message is required. If it is, the transient module IBMBESM is loaded and invoked. On return from the transient message module, the action byte is tested to determine the next action.

If there is no error On-unit, or the error On-unit indicates system action, the fact is noted by IBMBERR. If there is no GOTO from the FINISH On-unit, then the contents of the TORC slot in the TCA, is set negative, thus ensuring that IBMFPIR, the termination module, will test the return code from IBMBERR to see if an ABEND is required.

For ERROR, the FINISH condition is raised. For FINISH, the action is the same as the action taken on return from a FINISH on-unit. For all other conditions either ERROR is raised or return is made to the point of interrupt.

System action for the CHECK condition is performed by a call to module IBMBERC. On return from IBMBERC, return is made to the point of interrupt.

Return to Point of Interrupt

The method of returning from IBMFERR depends on the entry point at which it was entered. If the entry point was IBMFERRB, return is made to the caller in the normal way by a branch on register 14. However, if entry was made via IBMFERRA, a return to the point of the hardware interrupt must be made. This is done in the following way:

1. The module restores the floating point registers and register 15 through 2 and changes the address in the PICA to an address within itself. It then causes an interrupt.

2. The Supervisor gains control after the interrupt and hands control to DFHSRP, in CICS, which in turn hands control to this module.

3. This module then resets the A(interrupt handler) slot in the CICS CSA, sets the second word of the PSW in the interrupt information, and restores registers 3 through 12.

4. Finally control passes to the supervisor which returns to the address of the PSW in PIE, i.e., back to the point of the original interrupt.

Entry Point IBMBERRC

A GETMAIN macro is issued to obtain some storage into which the abend information is moved. An ABEND macro is issued to terminate the task.

Linkage

Entry Point IBMBERRB:

R1 = A(interrupt control block)

Calls

IBMBERC - CHECK system action.
IBMFESM - Error message module (transient).

CHAPTER 3: DUMP ROUTINES

The dump control modules enable the PL/I programmer to obtain snapshot dumps of main storage; calling traces which show the path of program execution through procedure blocks, begin blocks, and on-units; maps of PL/I control blocks; and information on the status of the various files declared in the PL/I program.

The type of information retrieved by the dump modules on any particular entry to the dump package is determined by the argument list attached to the corresponding CALL PLIDUMP statement in the source program. The meanings of the valid dump option characters are as follows:

T - Trace of active blocks required (also CICS option).

NT - No trace of active blocks required.

F - File information required.

NF - No file information required.

H - A hexadecimal dump of main storage is required.

NH - A hexadecimal dump of main storage is not required.

B - Control blocks are to be printed (also CICS option).

K - Print certain relevant CICS control blocks (TWAand TIOAs). (CICS option only)

NB - Control blocks are not to be printed.

S - Execution of the program is to be terminated after the dump (also (CICS option).

C - Execution of the program is to continue after the dump (also CICS option).

R - Report. The lengths and addresses of the main areas of storage in use immediately before the call to PLIDUMP are given.

NR - No report information required.

Q - Quick dump. This gives a DOS system dump with none of the formatting and other information provided by PLIDUMP.

NQ - A DOS system dump is not required.

D - Debug. Additional information about files will be given. This includes the name of the transmitter and the open module, and information on whether ENDFILE or an error has occured on the file.

ND - No debug. The additional file information not required.

60 - The hexadecimal notation will be translated into the 60 character set.

48 - The hexadecimal notation will be translated into the 48 character set.

IBMDKMR controls the retrieval of information by the other dump modules by loading and invoking them in turn. Each time a dump module is to be loaded, the module acquires a VDA of the required length. When control returns from the dump module, the VDA is freed.

If no dump options string is passed to IBMDKMR, the module sets a dump options flag byte to its default values. Should a dump options string be passed, IBMDKMR loads and invokes the dump parameter translation

The default dump options are 'T', 'F', 'C', 'R', 'CHAR48' and 'D'. To turn off these defaults the options 'NT', 'NF', 'S', 'NR', 'CHAR60', and 'ND' respectively must be specified.

The dump modules may be called at any stage in the program. The initial call from compiled code is always to a resident bootstrap module IBMBKDM. This module loads and invokes the main dump control module IBMDKMR, passing to it up to two character strings that it has received from compiled code. These are two optional arguments from the CALL PLIDUMP statements. The first of these is the options list, the second is the user identifier.

On CICS systems the resident bootstrap module loads and calls IBMFKMRA which calls CICS-only dump modules corresponding to the DOS modules. For more details see the description of this routine. For standard DOS systems the following description applies.

The main dump control module is supported by the following modules:

IBMBKPT - The module translates the dump options character string into a flag byte for use by the other dump modules. The defaults are assumed for any unspecified options.

IBMDKTR - The module obtains information for the calling trace if dump option 'T' has been specified.

IBMDKFA - The module obtains any information on PL/I files that is required for the dump.

IBMDKDD - The module prints a complete hexadecimal dump of the main storage used by the program.

IBMDKTC - The module checks DSA chains and loads IBMDKTR.

IBMDKTB - The module prints out hexadecimal maps of TCA, TIA, and DSA chains and prints a flow table, if there is one.

IBMDKDR - The module produces a storage report for the current storage allocation.

IBMDKDT - The module transmits each line of PLIDUMP output to SYSLST file.

When the dump is complete, IBMDKMR returns control to the resident bootstrap module IBMBKDM, passing it a return code to indicate whether or not the program is to continue.

MODULE DESCRIPTIONS

IBMDKMR - Dump Control Module

Function

To control the retrieval of information by the dump modules.

Method (chart DKMR)

module IBMBKPT. This module translates the dump options character string which was passed to IBMDKMR into a byte of flags.

Depending on the specified dump options, IBMDKMR then loads other transient modules to obtain the required information for the dump. The modules are loaded and invoked in the following order.

1. IBMDKTC - Save area chain validity checker.
2. IBMDKFA - File information module.
3. IBMDKDD - Hexadecimal dump module.

Finally, IBMDKMR tidies up the dump environment and returns to IBMBKDM, passing it a return code in register BR to indicate whether the program is to stop or continue.

Printing the Dump:

Dump information is transmitted a line at a time to the logical unit SYSLST by a transmitter routine contained within IBMDKMR. The address of this transmitter is held in IBMDKMRs DSA; it is thus accessible to IBMDKTR, IBMDKFA, IBMDKTB, and IBMDKDD as well as to IBMDKMR itself.

The I/O area specified in the DTF for SYSLST is also in the DSA belonging to IBMDKMR. The address of this DSA is passed in register R1 each time the transmitter is called.

Handling Program Check Interrupts:

The normal method of handling program check interrupts is by entry to a resident error-handling routine (see DOS PL/I Resident Library: Program Logic). To prevent program check interrupts in the dump modules from causing an entry to the resident error handler and possibly terminating the whole program, a special interrupt handling routine is included in module IBMDKMR. The address of this routine is given to the supervisor by a STXIT macro issued when IBMDKMR is first entered.

When a program check interrupt occurs, the interrupt routine in IBMDKMR transfers control to the address held in its DSA. This address is repeatedly changed during the execution of IBMDKMR to ensure that, if an interrupt occurs, execution of the dump is resumed from a suitable point. The effects of this are as follows:

If control is in the dump parameter translation module IBMBKPT, execution of that module is abandoned, the default dump options are assumed, and no user identifier is looked for.

If control is in either IBMDKFA, IBMDKTR, IBMDKTB, IBMDKDR, or IBMDKTC, the "H" option is set on automatically and the program continues from the point in IBMDKMR that is directly after the call to the offending module.

If control is in module IBMDKDD, execution of that module is abandoned and a system DUMP macro is issued.

Otherwise, if control is in IBMDKMR, execution is resumed from the next suitable point.

Before returning control to the resident dump bootstrap module, IBMDKMR issues a STXIT macro to restore the interrupt handling method to normal.

Linkage

Input:

```
R1    = A(PLIST)
PLIST = A(string locator for dump options string)
        A(string locator for dump identifier)
```

Input to SYSLST transmitter:

```
R1 = A(line to be printed)
```

Output to IBMBKDM (Dump Bootstrap):

```
BR = return code (zero for "continue", non-zero for "stop")
```

Calls

IBMBKPT - Dump options translate.
IBMDKTR - Dump Trace.
IBMDKFA - File attributes.
IBMDKDD - Hexadecimal dump.

Called By

IBMBKDM - Resident dump bootstrap.
IBMDESY - Error system action.

IBMDKDD - Hexadecimal Dump

Function

To create a hexadecimal dump of the partition in which the program is resident.

Method (chart DKDD)

The module loads the address of the communications region, which is held in the implementation appendage. From the communication region, it picks up the address of the first byte following the supervisor transient area and the address of the uppermost byte of the problem program area. A hexadecimal dump of the communications region and of the contents of main storage between the two addresses is then given. The dump is created one line at a time in a buffer in IBMDKMRs DSA. Each line consists of the address of the first byte in the line, followed by the contents of 32 bytes of storage in hexadecimal, followed by a character translation of those 32 bytes. The dump is separated at the start of the problem progam area and at the start of LIFO storage (at the TCA). If no other PLIDUMP information is supplied, then the contents of Registers 12 and 13 and the floating point registers will be given at the top of the hex dump.

Linkage

```
DR  = A(Save area of IBMDKMR)
```

Calls

Transmitter routine in IBMDKMR.

Called By

IBMDKMR - Dump control module.

IBMDKDR - Report Option.

Function

To produce a report giving the size of various storage segrents currently allocated.

Method

For each line of output, the start and end addresses of each section of the report are passed to a subroutine. The subroutine produces a line of output giving the length of storage and, at convenient points, the current total figure for that type of storage.

Input:

DR = A(save are of IBMDKMR)

Output to Transmitter Routine:

R1 = A(line for output)

Calls

IBMDKDT - Dump File transmitter.

Called By

IBMDKMR - Dump Control Module.

IBMDKDT - Dump File Transmitter

Function

To transmit lines of output to the PLIDUMP file, and to close the file on program termination.

Method (chart DKDT)

The module is passed the address of the output line in register R1. For Open and Close requests, register R1 points to the request code.

The module first tests the first byte of the record (byte 1) to determine what is required. The values that can appear in this byte are as follows:

X'00'	Open file.
X'01'	Close file.
X'40'	Transmit one line, single space.
X'F0'	Transmit one line, double space.
X'F1'	Transmit one line on new page.

If a new page is required, then the end page exit is taken so that a heading can be produced. The pages are also counted by this module.

Open Requests

The module initializes the dump control block and the DTF. It opens the DTF and passes the address of the dump control block back to its caller.

Transmission Requests

The module moves the line to its internal buffer and sets the appropriate ASA control character for correct spacing. If endpage is encountered the module calls the endpage exit. It then transmits the heading line if necessary and requested data.

Close the dump file:

If a "close" request is received, the file is closed.

Linkage

R1 = A(DUB)

Exit

Normal return to caller via link register.

IBMDKFA - Dump File Attributes

Function

To search the file blocks for open files and to obtain information on the attributes of the files and, if possible, the contents of the buffers. If the dump option "B" has been specified, the module also maps out the relevant I/O control blocks.

Method (chart DKFA)

The module traces through the chain of open files, and takes the following action for each file.

The module first extracts the file name and attributes and prints them out. If required, the module then prints the address of the FCB, followed by a dump of the FCB in hexadecimal together with a character translation.

If the buffer is accessible, its contents are printed out in character form, unless the control block option has been specified, in which case the contents are printed in hexadecimal with a character translation.

Control is returned to IBMDKMR when the last FCB on the open file chain has been analysed.

Linkage

Input:

DR = A(save area of IBMDKMR)

Output to Transmitter Routine:

R1 = A(save area of IBMDKMR)

Calls

Transmitter routine in IBMDKMR.

Called By

IBMDKMR - Dump control module.

IBMDKPT - Dump Parameter Translate

Function

To tranlate a variable length character string of dump options into a series of flags.

Method (chart BKPT)

The translation is performed in accordance with the following table:

Dump Option	Flag Bit	Value	Meaning
T	0	1	Trace
NT		0	No Trace
F	1	1	Files
NF		0	No Files
H	2	1	Hex. Dump
NH		0	No Hex. Dump
S	3	1	Reserved
C		0	
R	4	1	Report
NR		0	No Report
	5	1	Reserved
		0	
B	7	1	Control Blocks
NB		0	No Control Blocks
D	8	1	Debug
ND		0	No Debug
60	9	1	Char60
48		0	Char48
	10	1	Reserved
		0	
	11	1	Reserved
		0	
	12	1	Reserved
		0	
	13	1	Internal Switch
		0	
	14	1	Internal Dump
		0	No Internal Dump
Q	15	1	Quick Dump
NQ		0	No Quick Dump

The module locates the dump options character string by means of a parameter list in the save area of the dump control module IBMDKMR. If the length of the character string is found to be negative or zero, the default options "T", "F", "C", "R", "CHAR48" and "D" are assumed.

If no list of options has been given, the default options are assumed.

Before being analysed, the parameter string is converted into upper case.

Once a valid dump options character string has been found, it is analysed one letter at a time and the appropriate flag is set on or off. The default value is assumed for any unspecified option.

Linkage

DR = A(save area of IBMDKMR)

Called By

IBMDKMR - Dump Control

IBMDKTR - Dump Trace Routine

Function

To trace through the DSAs, finding the calling trace of the calling procedure, and the numbers of the calling statements. To analyse the ONCA and the error handler's save area for useful information and to give a hexadecimal representation of the main control blocks.

The module is loaded by IBMDKTC.

Method (chart DKTR)

Starting at the last DSA, the module chains back through the DSAs, to find those associated with procedures, begin blocks and ON-units. Wherever it can, it prints the name and also the statement number in which the associated block was called. If a hexadecimal dump has been requested, the offsets of the calling statements are also given, together with the entry address and the address of the DSA belonging to the block.

If an ON-unit DSA is found, the appropriate ONCA is examined and the built-in functions ONFILE,ONKEY, ONSOURCE, ONCHAR, ONCOUNT and DATAFIELD are evaluated, if they are valid.

If an ERROR on-unit was entered, the type of condition that caused ERROR to be raised is printed out. The on-code is found by loading and executing IBMBEOC.

The contents of the registers on entry to the resident error handler are printed. The interrupt address is given if the error or finish on-unit was entered as the result of a program check. A chain back through all library save areas to the next compiled code DSA is also given.

Linkage

DR = A(save area of IBMDKMR)

Calls

IBMBEOC - ONCODE module.
IBMDKTB - Save area control block printout.

Called By

IBMDKTC - Save area chain validity checker.

IBMDKTB - Save Area Control Block Printout

Function

To separate and print in hexadecimal the important program management control blocks.

Method

If blocks, trace, and a hexadecimal dump have been requested, the module is called to print individually, in hexadecimal format, the TCA and TIA. The dynamic chain of DSAs is then printed in hexadecimal. The contents of the register save area are printed above each DSA, whose type (e.g., Proc, on-unit etc.,) and if possible, identifier is given as well. Any VDAs that have overflowed the LIFO stack are also printed individually.

|The module also prints the static storage for all active procedures. It |checks every DSA in the chain to see whether it corresponds to a PL/I |external procedure, and if so prints the static storage if it has not |already been printed.

Where trace has been requested, then should there be a flow table, this is printed in the order in which it was dynamically created. Spare block identifiers, if any, are printed first; otherwise, any spare numbers are printed with the words "is unknown" against any pair requiring a block identifier but not having the entry still in the table. Should the number of block identifiers be equal to the number in the table, the module places the correct block identifier names alongside the number pairs.

Finally, the module returns control to IBMDKMR.

Linkage

DR = A(current DSA)

Called By

IBMDKTR - Dump Trace.

IBMDKTC - Save Area Chain Validity Checker

Function

To check the DSA chain for loops and zero chain-back fields.

Method

The module, which is loaded by IBMDKMR, looks at DSA chains, searching for a loop or a zero chain-back field. It notes the positions of a loop, and if one is found, it stores the address of the DSA at the end of loop in a field in the DSA which is common to IBMDKTR, IBMDKTB, and IBMDKTC. Otherwise, it leaves the field zero. It then loads IBMDKTR onto itself.

Linkage

DR = A(save area of IBMDKMR)

Calls

IBMDKTR - Dump Trace.

Called By

IBMDKMR - Dump Control.

IBMFKMR - Dump Control Module (CICS)

Function

To invoke other dump modules and transmit output.

Method (chart FKMR)

This module is invoked by the resident bootstrap IBMDKDM when a PLIDUMP is required in a CICS environment. It invokes IBMFKPT if options are specified, and the other modules if the options require them. The modules operation is similar to IBMDKMR.

IBMFKMR also contains the output transmitter used by all other modules, sending print lines to a transient data queue named CPLD, and an error recovery exit, to continue in the event of failure.

Subordinate modules are deleted when no longer required. An ENQ macro is issued to prevent simultaneous PLIDUMPs becoming intermingled in the output.

Input

```
R1 -> A(string locator for dump options)
      A(string locator for user identifier)-(optional)
or
R1 =  0 if no options specified.
```

Called By

IBMBKDM

Calls

IBMFKPT
IBMFKTC
IBMFKCS
DFHPCP
DFHSCP
DFHKCP
DFHTDP

IBMFKPT - Dump Options Analysis Module (CICS)

Function

To convert the options parameter into flags.

Method (chart FKPT)

If options are specified, this module is invoked by IBMFKMR to convert valid options into flags in IBMBZSAV, a DSA acquired by IBMFKMR and used throughout PLIDUMP as a communications area.

Input

DR -> IBMBZSAV
HAPM (in IBMBZSAV) = R1 on entry to IBMFKMR

Called By

IBMFKMR

Calls

DFHPCP

IBMFKTC - Dump Checking Module (CICS)

Function

Detect and note any errors in DSA chain, and invoke IBMFKTR.

Method (chart FKTC)

IBMFKTC is invoked if the 'T' option applies. The module checks the DSA for loops or zero backchain up to the dummy DSA, and then invokes IBMFKTR. This module also contains an error recovery exit.

Input

DR -> IBMBZSAV

Called By

IBMFKMR

Calls

IBMFKTR
DFHPCP
DFHSCP

IBMFKTR - Dump Trace Module (CICS)

Function

Prints the trace part of a PLIDUMP under CICS.

Method (chart FKTR)

IBMFKTR is invoked if the 'T' option is in effect. The module deletes IBMFKTC and sets HPCN (the 'current module' field in IBMBZSAV) to itself. It traces the dynamic path through the PL/I program, printing trace messages. If an on-unit has been entered, it gives the values of all relevant pseudovariables. If the 'B' option is also specified, registers on entry to error and the number of library modules prior to error, are printed. The 'B' option also causes DSA and entry point addresses to be printed in the trace, and IBMFKTB to be invoked. Statement numbers will be printed if available. All lines are printed via the print transmitter in IBMFKMR, whose address is in IBMBZSAV. This module also contains an error recovery exit.

Input

DR -> ZTRA (DSA allocated by IBMFKTC for IBMFKTR and IBMFKTB)

Called By

IBMFKTC

Calls

IBMFKTB
IBMBEOC
DFHPCP
DFHSCP

IBMFKTB - Dump Blocks Module (CICS)

Function

Prints the PL/I blocks in hexadecimal for PLIDUMP under CICS.

Method (chart FKTB)

IBMFKTB is invoked from IBMFKTR if either the 'B' option is in effect, or there is a flow table. The module deletes IBMFKTR and sets HPCN to itself so that IBMFKMR can delete the correct module when it regains control. If the 'B' option applies, PL/I control blocks (TCA, TIA, DSAs and separate VDAs) will be printed in hexadecimal format, and the flow table will be analyzed and printed if it exists. All lines are printed via IBMFKMR's transmitter whose address is held in IBMBZSAV. This module also contains an error exit to recover from failures.

Input

DR -> ZTRA (DSA allocated by IBMFKTC for IBMFKTR and IBMFKTB)

Called By

IBMFKTR

Calls

DFHPCP
DFHSCP

IBMFKCS - Dump CICS Control Blocks Module

Function

Print certain CICS control blocks in hexadecimal.

Method (chart FKCS)

The CICS transaction work area is printed in hexadecimal, and the chain of TIOAs, if any, is followed, each one being printed in turn. The current one is marked as 'CURRENT'.

Input

DR -> IBMBZSAV

Called By

IBMFKMR

Calls

DFHPCP
DFHSCP

CHAPTER 4: PROGRAM MANAGEMENT ROUTINES

The modules described in this chapter are concerned with program initialization and with putting out messages that relate to errors which prevent correct housekeeping or storage management. The errors are:

1. The executable program phase has no main procedure. This error is detected at program initialization.

2. There is insufficient storage available for the housekeeping control blocks. This error is detected at program initialization.

3. A program check interrupt has occurred while control is in the resident error-handling module (IBMDERR) or in one of the modules called by it.

4. There is insufficient storage available to satisfy a request for non-LIFO storage or for a new segment of LIFO storage.

Messages for errors 1 to 3 above are handled by IBMDPEP, and that listed in 4 is handled by IBMDPES.

CICS

The initialization/termination routine on CICS (HPIR) is part of the CICS nucleus. It is link-edited together with FPCC (routine module), FERR, and FPGR to form the load module DFHSAP. It receives control from the resident intialization bootstrap (DFHPL1I), and completes the initialization without the use of any of the other modules.

MODULE DESCRIPTIONS

IBMDPII - Program ISA Initialization

Function

To initialize the program ISA, prepare to handle operation exceptions, and cause IBMDERR to be entered if a program check occurs.

Method

IBMDPII is loaded and called by IBMDPIR. The space requirements for initialization are calculated. These requirements include space for any FORTRAN buffers and DTF in an ILC environment, space for the program management area, space for the checking code for operation exceptions (for details of CPUs with no floating point hardware - see below), and space for any flow table.

If there is insufficient storage to fulfil these requirements, IBMDPEP is loaded to produce a message to that effect.

The space is initialized in the following way. Firstly, any space required for FORTRAN buffers is reserved immediately following the end of the load module; secondly, the space for the ISA is reserved. Finally, space at the end of the partition is reserved for any FORTRAN

DTF that may be required. A dummy DSA is set up for IBMDPII and the TCA is initialized at the low address end of the ISA. A field TGDS in the TCA is set to address a subroutine provided to get DSA's istead of getting them by inline code. The subroutine is moved into the program management area by IBMDPII.

Besides initializing the TCA, IBMDPII also sets up duplicates of four slots contained therein. The slots are TOVF, TERR, TENV and TGTC. Finally the implementation defined appendage of the TCA is initialized.

A STXIT macro is issued to catch program check interrupts. The STXIT macro, which specifies the addresses of a 72-byte save area and a small section of code in the implementation appendage of the TCA, ensures that program check interrupts are dealt with by the PL/I error-handling package rather than by the system.

To ensure that operation exceptions that occur during execution of the PL/I program are correctly handled, IBMDPII determines, from the communication region, whether or not the floating-point instruction set is supported, and moves the appropriate instructions to a field at the end of the program management area. The address of this field is contained in TAFF, a field in the TIA. If the floating-point instruction set is supported, the instruction that is moved is an immediate return to the caller (i.e., IBMDERR). If the floating-point instruction set is not supported, IBMDPII loads IBMDPIF into LIFO storage and sets its address in TAFF.

After setting up the remaining control blocks necessary for housekeeping, IBMDPII returns to IBMDPIR.

Linkage

```
R4    = Length of ISA
R5    = A(Compiled code PLIST)
R9    = A(Initialization PLIST)
Initialization PLIST = A(Return into IBMDPIR)
                       V(IBMBLOCA)
                       V(IBMBERRA)
                       V(IBMBPGRD)
                       V(IBMBPGRA)
                       V(IBMBPGRB)
                       V(IBMBPGRC)
                       V(IBMBERRB)
                       V(IBMBPGOA)
                       V(IBMBJWTA)
                       V(IBMBTOCA)
                       V(IBMBTOCB)
                       V(IBMBILC1 CSECT)
                       V(SYSPRINT DCLCB)
                       V(PLIFLOW)
                       V(PLITABS)
                       Offset(GET LWS Routine)
                       GOTO out of block routine
RY    -                PLIMAIN
```

Called By

IBMDPIR - Program Initialization (Resident)

IBMDPJI - Program ISA Initialization From Caller

Function

To save the caller's STIXT program check parameters and initialize the dummy DSA; set up the standard part of the TCA and an implementation-defined appendage; create the DFB, dummy and current ONCAs, and the contents of the NAB field; obtain library workspace; prepare to handle operation exceptions, and cause IBMDERR to be entered if a program check occurs.

Method

IBMDPJI is loaded by IBMDPJR and saves the caller's STXIT program check parameters in the program management area. The remaining operations are carried out as described in IBMDPII. After completing all operations, IBMDPJI calls compiled code.

Linkage

```
R4     = Length of ISA
R5     = A(Compiled code PLIST)
R9     = A(Initialization PLIST)
Initialization PLIST = V(IBMBOCLA)
                       V(IBMBERRA)
                       V(IBMBPGRD)
                       V(IBMBPGRA)
                       V(IBMBPGRB)
                       V(IBMBPGRC)
                       V(IBMBERRB)
                       V(IBMBPGOA)
                       V(IBMBJWTA)
                       V(IBMBTOCA)
                       V(IBMBTOCB)
                       A(IBMBILC1 CSECT)
                       A(SYSPRINT DCLCB)
                       A(PLIFLOW)
                       A(PLITABS)
                       Offset OF GET LWS ROUTINE
                       GOTO out of block routine

RX     = A(COMRG)
RY     = A(address of PLIMAIN)
CR     = A(TCA)
```

Called By

IBMDPJR - Program Initialization from Caller

IBMDPEP - Housekeeping Diagnostic Message Module

Function

To put out error messages describing errors that prevent correct housekeeping. The errors handled are indicated under "Method" below.

Method (chart DPEP)

The value in register 1 on entry to the module indicates the message that is required. The error codes are:

R1 = 8   No main procedure

R1 = 12 Insufficient storage available at program initializaticn

R1 = 16 Interrupt in the resident Error Handler or in one of the modules called by it.

If the message for insufficient storage available is required, there may not even be sufficient storage for the module that opens the diagnostic file. In this case, therefore, the message is transmitted to the console by means of an EXCP macro, and a partial dump of the partition is produced. If the PL/I program has been invoked by a caller, return is then made to that caller; otherwise, the program is cancelled.

If the message for an interrupt in the error-handler is required, it is probable that registers and control blocks have been corrupted. In this case, therefore, the message is transmitted to the console by means of an EXCP macro, and a partial dump of the partition is produced. If the PL/I program has been invoked by a caller, return is then made to that caller; otherwise, the program is cancelled.

Linkage

R1 = error code

```
Error code: 8 = no main procedure
           12 = no storage available at
                progam initialization
           16 = interrupt in resident
                error handler.
```

Calls

IBMDEDO - Open diagnostic file.

Called By

IBMDPIR - Program Initialization (Resident)
IBMDERR - Resident error handler.

IBMDPES - Storage Management Diagncstic Message Module

Function

To put out an error message when a request for non-LIFO storage or for a new segment of LIFO storage cannot be satisfied.

Method (chart DPES)

The module is loaded and invoked by the resident program initialization and termination module when a request for non-LIFO storage or for a new segment of LIFO storage cannot be satisfied by the resident storage handling module.

The message produced by the module is:

```
IBM008I jobname NO MAIN STORAGE AVAILABLE, NEEDS xxxxxxxxx MORE BYTES
IN STATEMENT NUMBER xxxxx at OFFSET yyyyy
FROM (either) 'PROCEDURE procedurename WITH
ENTRY POINT entryname' (or) 'ENTRY POINT OF
ontype ON-UNIT'
```

The message is transmitted to SYSPRINT if possible; if SYSPRINT cannot be used, the message is written on the console by means of a EXCP macro.

To prevent transmit errors on SYSPRINT from causing entry to the resident error handler, the module sets field TERR in the TCA to point to a routine in IBMDPES which re-establishes its own DSA and starts its execution again so that the message will appear at the console.

A partial dump of the partition is then produced. If the PL/I program has been invoked by a caller, return is made to that caller; otherwise the program is cancelled.

Linkage

The module finds the current DSA by accessing field TABT in the TCA.

Calls

IBMDST* - Print, *-format transmitter.
IBMDEDW - Console transmitter.
IBMDKMR - Dump control module.
IBMBPGR - Resident storage handling module.

Called By

IBMDPIR - Program Initialization (Resident)

IBMFPCCA - CICS Nucleus Routing Routine

Function

Forms the entry point to DFHSAP and routes control to HPIR when required. Also contains bootstraps to the stream transmitter (FSTV) for REPORT/COUNT/OPEN/CLOSE etc.

IBMFPGD - Storage Management (for REPORT option on CICS)

Function

To allocate and free non-LIFO storage and to obtain and free segments of the LIFO stack when insufficient space is available in the current segment. Furthermore, the module maintains up-to-date information on the use of storage to enable the creation of a report at program termination. The module has four entry points:

IBMBPGDA: Get non-LIFO storage.

IBMBPGDB: Free non-LIFO storage.

IBMBPGDC: LIFO-stack overflow recovery for "get DSA."

IBMBPGDD: LIFO-stack overflow recovery for "get VDA."

Method (chart FPGD)

In addition to the actual storage management duties described below under its four entry points, the module performs the following actions necessary to maintain the information from which the storage report table is constructed.

So that IBMFPGD will always be entered to get LIFO storage, the value of EOS in the TCA is copied into the table by IBMFPIR and the value of EOS in the TCA then set to zero. The value for EOS is then maintained by IBMFPGD in the copy contained in the table.

The module keeps count of the number of times it issues a GETMAIN or FREEMAIN macro and also, the number of times it is entered at entry points IBMBPGDA and IBMBPGDB.

When it is entered for an overflow in the major segment of the LIFO stack or when EOS is changed in the major segment (as a result of a request for non-LIFO storage) the module computes the value of EOS minus NAB and places the result in the storage report table as representing the value of the current unused ISA.

This value is then deducted from a value, contained in the storage report table, which represents the maximum amount of PL/I storage currently in use outside of the ISA.

The result of this computation is then compared with the value in the report table representing the previously calculated maximum value of PL/I storage outside the ISA. i.e., the previous maximum value reduced by amount of unused ISA. If the currently calculated value is greater than the previously calculated value, it is inserted in the report table in place of the latter.

Every time the module issues a DFHSC TYPE=GETMAIN macro it adds the length specified to the length of the PL/I storage currently allocated outside the ISA. If the result is greater than the existing maximum value in the report table, then it is entered in the report table in place of the latter. From this new value entered in the report table, the module subtracts the value representing the current amount of unused ISA (i.e., EOS - NAB). As described in the foregoing paragraph, the result obtained then replaces the existing value in the table, if the result is greater.

Every time the module issues a DFHSC TYPE=FREEMAIN macro, the length is subtracted from the length of PL/I storage currently allocated outside the ISA.

Entry Point IBMBPGDA:

Whenever possible, allocations of non-LIFO storage are made from areas on the free-area chain. IBMBPGD rounds up the amount of storage requested by the caller to a multiple of 8 bytes, and then searches the free-area chain for the smallest area that is large enough to meet the storage requirements. If an area of exactly the required size is found, it is dechained and its address is returned to the caller. If an area is found that meets the above conditions, but which is larger than the area required, the allocation is made from the high address end of the area. In this case the address returned to the caller is the address of the end of the area minus the number of bytes allocated. The length of the remaining free area, which is stored in its first byte, is then reduced by the number of bytes allocated.

If no free-area chain exists, or if all the areas of the chain are too small, an attempt is made to allocate the storage in the area delimited by the next available byte (NAB) pointer and the end of segment (EOS) pointer, if this area is within the ISA. If this area is large enough, EOS is reduced by the number of bytes required and the address of the area is returned to the caller.

If the area betweeen the NAB and EOS pointers is too small or outside the ISA, the allocation cannot be made. Under these circumstances IBMBPGD issues a DFHSC TYPE=GETMAIN macro for the required amount of storage, and adds this storage to HLL storage chain.

Entry Point IBMBPGDB:

The length of storage that is to be freed is first rounded up to a multiple of 8 bytes. If the storage does not belong to the ISA, then it is freed to the supervisor by means of a DFHSC TYPE=FREEMAIN macro, and removed from the HLL chain.

However, if it does belong to the ISA, IBMBPGDB then scans the free area chain to determine whether or not there is a free area contiguous with the high order boundary of the storage that is to be freed. If such an area is found, it is dechained and its length is added to the length that is to be freed.

The free area chain is then scanned again to determine whether or not there is a free area contiguous with the low order boundary of the storage that is to be freed. If there is, its length is increased by the length of the storage that is to be freed, and control is returned to the caller.

If there is no free area contiguous with the low order boundary, a test is made to see whether or not the storage to be freed is at the address pointed to by EOS. If it is, EOS is moved; otherwise the storage to be freed is added to the free area chain. Control is then returned to the caller.

Entry Point IBMBPGDC:

This entry point is called when there is no space in the current segment of the LIFO stack in which to allocate a DSA. Since the caller is in the process of allocating a DSA, IBMBPGDC saves its registers in the special program management save area in the TCA.

When a request is made to IBMBPGDC to get a new LIFO stack segment, empty segments may already exist. If more than one empty segment exists, the current segment is freed and the previous segment is made current. This is done by setting BOS and EOS to the values stored in the two control words at the head of the current segment and then freeing the segment by the method described above for non-LIFO storage. Successive segments are freed in this way until only one empty segment exists.

The allocation of storage can now be made. If there is enough space in the previous non-empty segment for the allocation, the allocation is made in this segment and the current empty segment is freed. Otherwise, the allocation is made in the current empty segment provided that it is large enough. If the current empty segment is too small, it is freed; the action then taken is the same as that for the case when no empty segment exists.

If no empty segment exists an area of non-LIFO storage is obtained by the same method as IBMBPGDA, except that the largest possible area is obtained. BOS and EOS are stored in the area obtained and then updated to address the new segment.

Note that IBMBPGDC is not called if there is space for the allocation in the latest non-empty segment and no empty segments exist.

Entry Point IBMBPGDD:

This entry point is called when there is no space in the current segment of the LIFO stack in which to allocate a VDA. The action taken is the same as that described for entry point IBMBPGDC (above), except that the caller's register 14 is saved in the caller's save area and the floating-point registers are saved and restored.

Error and Exceptional Conditions

If more than 250 segments of storage have been allocated, then the program is terminated with a DFHPC TYPE=ABEND macro.

Called By

Compiled code and any resident or transient library module that requires storage.

Linkage (On Entry)

Entry Point IBMBPGDA:

R0 = length of storage required.

Entry Point IBMBPGDB:

R0 = length of storage to be freed.
R1 = address of storage to be freed.

Entry Points IBMBPGDC and IBMBPGDD:

R0 = length of LIFO storage to be allocated + latest NAB.
R1 = latest NAB.

External Modules

Supervisor - (GETMAIN,FREEMAIN macros)

Linkage (On Exit)

Entry Point IBMBPGDA:

R1 = A(storage obtained)

Entry Points IBMBPGDC and IBMBPGDD:

R0 = new NAB

R1 = address of storage for element

Exit

Normal - Return to caller via link register.
Abnormal - Return to IBMBPIR (resident) to terminate program.

IBMFPGR - Storage Management (for NOREPORT option on CICS)

Function

To allocate and free non-LIFO storage and to obtain and free segments of the LIFO stack when insufficient space is available in the current segment. The module has four entry points:

IBMBPGRA: Get non-LIFO storage.

IBMBPGRB: Free non-LIFO storage.

IBMBPGRC: LIFO-stack overflow recovery for "get DSA."

IBMBPGRD: LIFO-stack overflow recovery for "get VDA."

Method (chart FPGR)

Entry Point IBMBPGRA:

Whenever possible, allocations of non-LIFO storage are made from areas on the free-area chain. IBMFPGR rounds up the amount of storage requested by the caller to a multiple of 8 bytes, and then searches the free-area chain for the smallest area that is large enough to meet the storage requirements. If an area of exactly the required size is found, it is dechained and its address is returned to the caller. If an area is found that meets the above conditions, but which is larger than the area required, the allocation is made from the high address end of the area. In this case the address returned to the caller is the address of the end of the area minus the number of bytes allocated. The length of the remaining free area, which is stored in its first byte, is then reduced by the number of bytes allocated.

If no free-area chain exists, or if all the areas of the chain are too small, an attempt is made to allocate the storage in the area delimited by the next available byte (NAB) pointer and the end of segment (EOS) pointer, if this area is within the ISA. If this area is large enough, EOS is reduced by the number of bytes required and the address of the area is returned to the caller.

If the area betweeen the NAB and EOS pointers is too small or outside the ISA, the allocation cannot be made. Under these circumstances IBMBPGR issues a GETMAIN macro for the required amount of storage, and adds this storage to the HLL storage chain.

Entry Point IBMBPGRB:

The length of storage that is to be freed is first rounded up to a multiple of 8 bytes. If the storage does not belong to the ISA, then it is freed to the supervisor by means of a DFHSC TYPE=FREEMAIN macro, and removed from the HLL chain.

However, if it does belong to the ISA, IBMBPGRB then scans the free area chain to determine whether or not there is a free area contiguous with the high order boundary of the storage that is to be freed. If such an area is found, it is dechained and its length is added to the length that is to be freed.

The free area chain is then scanned again to determine whether or not there is a free area contiguous with the low order boundary of the storage that is to be freed. If there is, its length is increased by the length of the storage that is to be freed, and control is returned to the caller.

If there is no free area contiguous with the low order boundary, a test is made to see whether or not the storage to be freed is at the address pointed to by EOS. If it is, EOS is moved; otherwise the storage to be freed is added to the free area chain. Control is then returned to the caller.

Entry Point IBMBPGRC:

This entry point is called when there is no space in the current segment of the LIFO stack in which to allocate a DSA. Since the caller is in the process of allocating a DSA, IBMBPGRC saves its registers in the special program management save area in the TCA.

When a request is made to IBMBPGRC to get a new LIFO stack segment, empty segments may already exist. If more than one empty segment exists, the current system is freed and the previous segment is made current. This is done by setting BOS and EOS to the values stored in the two control words at the head of the current segment and then freeing the segment by the method described above for non-LIFO storage. Successive segments are freed in this way until only one empty segment exists.

The allocation of storage can now be made. If there is enough space in the previous non-empty segment for the allocation, the allocation is made in this segment and the current empty segment is freed. Otherwise, the allocation is made in the current empty segment provided that it is large enough. If the current empty segment is too small, it is freed; the action then taken is the same as that for the case when no empty segment exists.

If no empty segment exists an area of non-LIFO storage is obtained by the same method as IBMBPGRA, except that the largest possible area is obtained. BOS and EOS are stored in the area obtained and then updated to address the new segment.

Note that IBMBPGRC is not called if there is space for the allocation in the latest non-empty segment and no empty segments exist.

Entry Point IBMBPGRD:

This entry point is called when there is no space in the current segment of the LIFO stack in which to allocate a VDA. The action taken is the same as that described for entry point IBMBPGRC (above), except that the caller's register 14 is saved in the caller's save area.

Error and Exceptional Conditions

If more than 250 segments of storage have been allocated, then the program is terminated with a DFHPC TYPE=ABEND macro.

Called By

Compiled code and any resident or transient library module that requires storage.

Linkage (On Entry)

Entry Point IBMBPGRA:

On entry:

R0 = length of storage required.

Entry Point IBMBPGRB:

R0 = length of storage to be freed.
R1 = address of storage to be freed.

Entry Points IBMBPGRC and D:

R0 = length of LIFO storage to be allocated + latest NAB.
R1 = latest NAB.

External Modules

Supervisor - (GETMAIN,FREEMAIN macros)

Linkage (On Exit)

Entry Point IBMBPGR:

R1 = A(storage obtained)

Entry Points IBMBPGRC and IBMBPGRD:

R0 = new NAB

R1 = address of storage for element

Exit

Normal - Return to caller via link register.
Abnormal - Return to IBMBPIR (resident) to terminate program.

IBMHPIR - CICS Initialization Routine

Function

Initialize the PL/I execution environment for execution under CICS, including the CICS appendage in addition to the standard control blocks.

Method (chart HPIR)

This module is part of the composite load module DFHSAP in the CICS nucleus. It receives control from the resident initialization bootstrap (DFHPL1I) and obtains storage for, and initializes the various control blocks, the ISA etc, according to the options specified.

Called By

DFHPL1I

Linkage

```
    R1  = Parameter list to be passed to PL/I.
    R0  = A(callers (=CICS)DSA) with callers registers saved in
          this DSA.
    R12 = A(CICS TCA)
    R13 = A(CICS CSA)
    R10 = A(initialization parameter list)
```

Linkage (On Exit)

Routine to CICS

IBMFPMR - CICS Storage Report

Function

Produce a storage report

Method (chart FPMR)

This module prints the storage report table via the stream transmitter (FSTV) accessed via the CICS appendage.

The following discussion does not apply to CICS systems where file opening/closing is handled by FSTV. This is discussed in Chapter 7.

To open or close a file, a call is made to the resident library module IBMDOCL, known as the open/close bootstrap. When closing a file, IBMDOCL loads the module IBMDOCA to carry out the final I/O operations on the file and to close it (this module is described in detail later in this chapter).

When opening a file, IBMDOCL is passed the address of the FCB, the address of the OCB (if one is required), and the address of the string locator for the TITLE option if this option is being used. IBMDOCL then calls one of five transient library modules, depending on the type of file to be opened. The five transient library modules are the initial modules of five groups of modules, totalling eleven, that carry out open operations on varying types of files. The eleven modules, plus IBMDOCL, are shown in figure 5.1 with their associated file types and functions.

The module called by IBMDOCL for non-VSAM files issues the OPEN macro instruction, loads the transmitter if necessary, sets up the ERROPT and EOFADDR fields of the FCB, and handles, where appropriate, TITLE, PAGESIZE, and any repositioning options such as REWIND. If buffer space has been allocated during compilation, no further action will be necessary. However, if buffers are required and the DYNBUF option, variable environment options, or an invalid declaration has prevented buffer space being acquired during compilation, buffer space must be calculated and obtained before the OPEN macro instruction is issued. This is carried out by the lower level modules shown in figure 5.1.

In order to minimize space requirements, any second level modules called are overlayed on the high address end of the first level modules, and third level modules overlayed in the same way on second level modules. After the second, and possibly the third or fourth, module has been executed, a return is made to the first transient module to load the transmitter and set up the ERROPT and EOFADDR exits. The first level module returns by way of IBMDOCL to compiled code.

For VSAM files, module IBMDOPV is called for each file to be opened. Module IBMDOPV obtains space for the ACB (Access Method Control Block), the IOCB, and RPL (Request Parameter List). The module opens the ACB, determines the type of VSAM processing (Keyed or Entry Sequence) and loads the appropriate transmitter. IBMDOPV then returns to IBMDOCL.

For a multiple open, the compiler calls the library once for each file to be opened. Space for the open modules and for buffers is acquired in non-LIFO storage. IBMDOCL obtains 2K bytes for the transient open routines and, if buffer space is required and the length of the buffers was known at compile time, also obtains sufficient storage for the buffers. The transient routines are loaded into the low address end of the storage, and buffers at the high address end.

If the length of the buffers is unknown during compilation, the second level of modules (IBMDOPQ, IBMDOPT, IBMDOPY or IBMDOPV) obtains the necessary space. When this occurs, an unused free area will normally be left between the transmitter and the buffer space. The address of this space is placed on the free area chain.

A transmitter is loaded if it is not already present in main storage. (A test is made on the chain of loaded modules to see if it is.) If a transmitter is loaded, it is overlayed on the high address end of the first transient module.

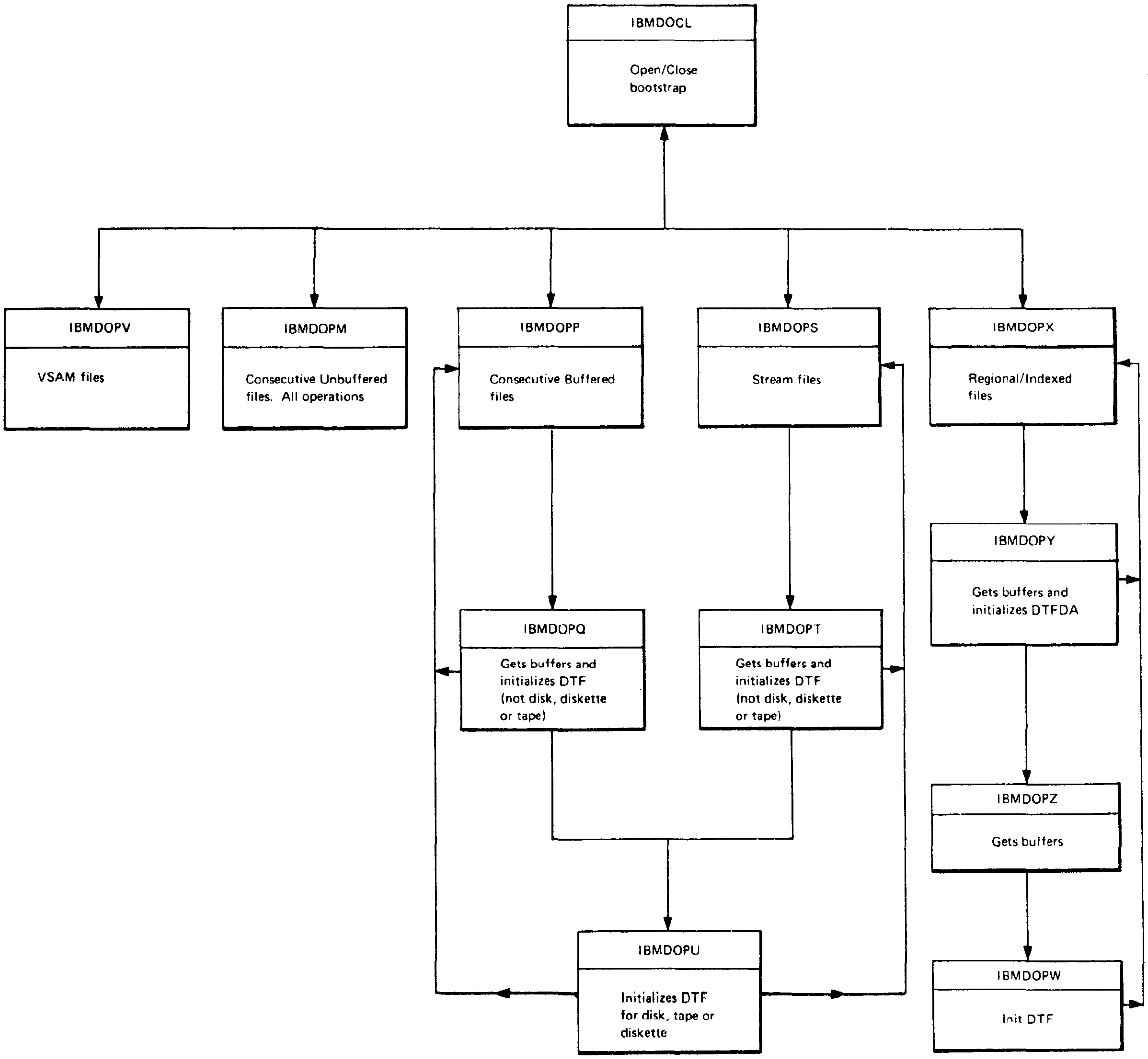

IBMDOCL
Open/Close bootstrap
IBMDOPV
VSAM files
IBMDOPM
Consecutive Unbuffered files. All operations
IBMDOPP
Consecutive Buffered files
IBMDOPS
Stream files
IBMDOPX
Regional/Indexed files
IBMDOPY
Gets buffers and initializes DTFDA
IBMDOPQ
Gets buffers and initializes DTF (not disk, diskette or tape)
IBMDOPT
Gets buffers and initializes DTF (not disk, diskette or tape)
IBMDOPZ
Gets buffers
IBMDOPU
Initializes DTF for disk, tape or diskette
IBMDOPW
Init DTF

Figure 5.1. The OPEN modules and their relationship

MODULE DESCRIPTIONS

IBMDOPM - Open (CONSECUTIVE UNBUFFERED Files)

Function

To process the options on the OPEN statement and issue an OPEN macro instruction for a CONSECUTIVE UNBUFFERED file.

Method (chart DOPM)

The module handles the entire opening process necessary for the file. If the file is being opened implicitly, an OCB is created for the file, specifying INPUT or OUTPUT as applicable.

If the compiler completes the DTF, IBMDOPM has only to process the TITLE option, the INPUT and OUTPUT attributes if present, and the disposition options for tape files. If the compiler could not complete the DTF, IBMDOPM must also check the ENV options and insert BLKSIZE in the DTF. When these processes have been completed, the module will then issue the OPEN macro instruction. For input files on tape, IBMDOPM simulates opening of the file by explicitly repositioning the tape.

After issuing the OPEN macro instruction, the module loads a transmitter for the file and inserts the addresses of the ERRCPT and EOFADDR routines into the DTF.

The UNDEFINEDFILE condition is raised if the file failed to open because an error was found.

Finally, if the file was successfully opened, the FCB is added to the open file chain.

Linkage

Explicit Open:

```
R1    = (PLIST)
PLIST = A(number of files) - This number is always 1
        A(file control block)
        A(open control block)
        A(TITLE string locator) or zero
        A(PAGESIZE) or zero
        A(LINESIZE) or zero
```

Implicit Open:

```
R1    = A(PLIST)
PLIST = A(file control block)
        A(request control block)
```

Calls

IBMDERR - Error handler (resident)

Called By

IBMDOCL - Open/Close bootstrap

IBMDOPP - Open (CONSECUTIVE BUFFERED Files)

Function

To process the options on the OPEN statement and issue an OPEN macro instruction for a CONSECUTIVE BUFFERED file.

IBMDOPP loads the necessary library transmitter, initializes the data set, and, if required, raises the UNDEFINEDFILE condition.

Method (chart DOPP)

The module handles the entire opening process for the file being opened unless the file has variable ENV options or DYNBUF was specified. The presence of such options prevents the compiler from calculating buffer and workspace sizes, and from initializing the DTF. In such cases, IBMDOPQ is loaded by IBMDOPP to continue the opening process.

If the compiler completes the DTF and gets buffers, IBMDOPP has only to process the TITLE option and the disposition options for tape files. When these processes have been completed, the module will then issue the OPEN macro instruction. After issuing the OPEN macro instruction or after returning from IBMDOPQ, the module loads a transmitter for the file and inserts the addresses of the ERROPT and EOFADDR routines into the DTF.

The UNDEFINEDFILE condition is raised if the file failed to open because an error was found. Finally, if the file was successfully opened, the FCB is added to the open file chain.

Linkage

Explicit Open:

```
R1    = A(PLIST)
PLIST = A(number of files) - This number is always 1
        A(file control block)
        A(open control block)
        A(TITLE string locator) or zero
        A(PAGESIZE) or zero
        A(LINESIZE) or zero
```

Implicit Open:

```
R1    = A(PLIST)
PLIST = A(file control block)
        A(request control block)
```

Calls

IBMDERR - Error Handler (resident) IBMDOPQ - Open module

Called By

IBMDOCL - Open/Close bootstrap

IBMDOPS - Open (STREAM Files)

Function

To process the options on the OPEN statement and issue an OPEN macro instruction for a STREAM file. IBMDOPS loads the necessary library transmitter, initializes the data set and buffer pointers, and, if required, raises the UNDEFINEDFILE condition.

Method (chart DOPS)

The module handles the entire opening process for the file being opened unless the file has variable ENV options or the LINESIZE option appears on the OPEN statement or DYNBUF was specified. The presence of such options prevents the compiler from calculating buffer and workspace sizes, and from initializing the DTF. In such cases, IBMDOPT is loaded by IBMDOPS to continue the opening process.

If the compiler completes the DTF and gets buffers, IBMDOPS has only to process the TITLE option, the PAGESIZE option, and the disposition options for tape files. When these processes have been completed, the module will then issue the OPEN macro instruction. After issuing the OPEN macro instruction or after returning from IBMDOPT, the module loads a transmitter for the file and inserts the addresses of the ERROPT and EOFADDR routines into the DTF.

Note: The stream input and F-format stream print transmitters IBMDSTI and IBMDSTF need not be loaded as they are always link-edited.

The UNDEFINEDFILE condition is raised if the file failed to open because an error was found. Finally, if the file was successfully opened, the FCB is added to the open file chain.

Linkage

Explicit Open:

```
R1    = A(PLIST)
PLIST = A(number of files) - This number is always 1
         A(file control block)
         A(open control block)
         A(TITLE string locator) or zero
         A(PAGESIZE) or zero
         A(LINESIZE) or zero
```

Implicit Open:

```
R1    = A(PLIST)
PLIST = A(file control block)
        A(request control block)
```

Calls

IBMDERR - Error Handler (resident) IBMDOPT - Open module

Called By

IBMDOCL - Open/Close bootstrap

IBMDOPX - Open (REGIONAL and INDEXED Files)

Function

To process the options on the OPEN statement and issue an OPEN macro instruction for a REGIONAL or INDEXED file.

IBMDOPX loads the necessary library transmitter, initializes the data set, and, if required, raises the UNDEFINEDFILE condition.

Method (chart DOPX)

The module handles the entire opening process for the file being opened unless the file has variable ENV options or DYNBUF was specified. The presence of such options prevents the compiler from calculating buffer and workspace sizes, and from initializing the DTF. In such cases, IBMDOPY is loaded by IBMDOPX to continue the opening process.

If the compiler completes the DTF and gets buffers, IBMDOPX has only to process the TITLE option. When these processes have been completed, the module will then issue the OPEN macro instruction. After issuing the OPEN macro instruction or after returning from IBMDOPY, the module loads a transmitter for the file and inserts the addresses of the ERROPT and EOFADDR routines into the DTF.

Furthermore, for a regional output file, all tracks are initialized with WRITE RZERO and dummy records written for Regional(1) direct output. In the case of an Indexed file, the module issues either a SETFL macro instruction if an output file, or a SETL macro instruction if the file is a sequential input file.

The UNDEFINEDFILE condition is raised if the file failed to open because an error was found. Finally, if the file was successfully opened, the FCB is added to the open file chain.

Linkage

Explicit Open:

```
R1    = A(PLIST)
PLIST = A(number of files) - This number is always 1
        A(file control block)
        A(open control block)
        A(TITLE string locator) or zero
        A(PAGESIZE) or zero
        A(LINESIZE) or zero
```

Implicit Open:

```
R1    = A(PLIST)
PLIST = A(file control block)
        A(request control block)
```

Calls

IBMDERR - Error Handler (resident) IBMDOPY - Open module

Called By

IBMDOCL - Open/Close bootstrap

IBMDOPQ - Open (CONSECUTIVE BUFFERED Files) - Level 2

Function

This module is loaded only if a CONSECUTIVE BUFFERED file could not be opened entirely by IBMDOPP, the first level transient open module. IBMDOPQ obtains the buffer space for all consecutive files and initializes the DTF for files other than those on disk, tape, or diskette.

Method (Chart DOPQ)

The module handles the remainder of the open processes not carried out by the associated first level module. Specifically, IBMDOPQ carries out the checking of variable ENV options, calculates the buffer size, and gets space for the buffers. It also calculates the size of IOAREA (the product of record length and command chaining factor) for diskette files.

IBMDOPQ also initializes the DTF and issues the OPEN macro instruction unless the file dealt with is on disk or tape, in which case, the module calls IBMDOPU. IBMDOPU, if called, initializes the DTF and issues the OPEN macro instruction, then returns to IBMDOPP, the first level open module.

Linkage

```
R2    = A(FCB)
R6    = A(DTF)




R7    = A(PLIST)
PLIST = A(number of files) - This number is always 1
        A(file control block)
        A(open control block) or zero
        A(TITLE string locator) or zero
        A(PAGESIZE) or zero
        A(LINESIZE) or zero
```

Calls

IBMDOPU - Open(level 3)

Called by

IBMDOPP - Open(level 1)

IBMDOPT - Open (STREAM Files) - Level 2

Function

This module is loaded only if a STREAM file could not be opened entirely by IBMDOPS, the first level transient open module. IBMDOPT obtains the buffer space for all stream files and initializes the DTF for files other than those on disk, tape, or diskette.

Method (Chart DOPT)

The module handles the remainder of the open process not carried out by the associated first level module. Specifically, IBMDOPT carries out the checking of variable ENV options and the LINESIZE option, calculates buffer requirements, and gets space for the buffers. It also calculates the size of IOAREA (the product of record length and command chaining factor) for diskette files.

IBMDOPT also initializes the DTF and issues the OPEN macro instruction unless the file dealt with is on disk or tape, in which case, the module calls IBMDOPU. IBMDOPU, if called, initializes the DTF and issues the OPEN macro instruction, and then returns to IBMDOPS, the first level open module.

Linkage

```
R2    = A(FCB)
R6    = A(DTF)

R7    = A(PLIST)
PLIST = A(number of files) - This number is always 1
        A(file control block)
        A(open control block) or zero
        A(TITLE string locator) or zero
        A(PAGESIZE) or zero
        A(LINESIZE) or zero
```

Calls

IBMDOPU - Open(level 3)

Called by

IBMDOPS - Open(level 1)

IBMDOPY - Open (REGIONAL/INDEXED Files) - Level 2

Function

This module is loaded only if a REGIONAL or INDEXED file could not be opened entirely by IBMDOPX, the first level transient open module. IBMDOPY obtains the buffer space and initializes the DTF for all regional files.

Method (Chart DOPY)

The module handles the remainder of the open process not carried out by the associated first level module. Specifically, IBMDOPY carries out the checking of variable ENV options and processes the TITLE option for Regional and Indexed files.

For REGIONAL files, IBMDOPY also initializes the DTF and issues the OPEN macro instruction. For indexed files, the module calls IBMDOPZ which initializes the DTF and issues the OPEN macro instruction, then returns to IBMDOPX, the first level open module.

Linkage

```
R2    = A(FCB)
R6    = A(DTF)

R7    = A(PLIST)
PLIST = A(number of files) - This number is always 1
        A(file control block)
        A(open control block) or zero
        A(TITLE string locator) or zero
        A(PAGESIZE) or zero
        A(LINESIZE) or zero
```

Calls

IBMDOPZ - Open(level 3)

Called by

IBMDOPX - Open(level 1)

IBMDOPV and IBMDOPE - Open (VSAM)

Function

IBMDOPV is loaded whenever a VSAM file is to be processed. It handles the setting up of the various control blocks necessary for VSAM (ACB, IOCB, RPL) and opens the file. IBMDOPE then obtains buffer and key storage space if required, and chains the file control block into the open file chain. IBMDOPV then loads the appropriate transmitter. If any errors are detected, they are raised by calling the error handler.

Method

IBMDOPV and IBMDOPE handle the entire opening process for the file being opened. IBMDOPV determines the length of storage required for the ACB and RPL by issuing a SHOWCB macro. It then obtains space in non-lifo storage for the ACB and generates an ACB using the GENCB macro. The filename is obtained from the TITLE option if specified and the PASSWORD from the PASSWORD ENV option. The values of BUFND, BUFNI, and BUFSP are obtained from the corresponding ENV options if specified. IBMDOPV then issues an OPEN macro and tests the return codes set up in the ACB. If the return code is correct space is obtained for IBMDOPE which is then loaded. The actual values of RECSIZE, KEYLENGTH and KEYLOC are checked against any ENV values specified. A TESTCB macro is then issued to determine the organisation of the data set (ESDS, KSDS or PATH, or RRDS). Provided the file attributes are valid for the data set organisation, the module obtains space from non-lifo storage for the IOCB and RPL, key space, and a dummy buffer if the file has the BUFFERED attribute.

The address of the RPL is placed in the IOCB and a GENCB macro issued to create the RPL. IBMDOPE then returns to IBMDOPV which frees the space obtained for IBMDOPE. The appropriate transmitter is then loaded from the following list:

IBMDRVZ	(ESDS)
IBMDRVT	(KSDS SEQ OUTPUT)
IBMDRVS	(KSDS and PATH Input/Update/Direct transmitter)
IBMDRVR	(RRDS transmitter)

Finally the FCB is added to the open-file chain. If any errors are detected then the file is closed and the storage for the ACB freed before the condition is raised.

Linkage

Explicit Open

R2	=	A(FCB)
R6	=	A(Module table)
R7	=	A(Plist as passed to IBMDOCL)

PLIST	=	A(number of files) - Always 1
		A(File Control Block)
		A(Open Control Block)
		A(Title string locator) or zero

```
A(Pagesize)             or zero
A(Linesize)             or zero
```

Implicit Open

```
R7    =  A(Plist) as passed to IBMDOCL
R6    =  A(Module table)
R2    =  A(FCB)
PLIST =  A(File Control Block)
         A(Request Control Block)
```

Calls

IBMDERR - Error Handler (resident)

Called by

IBMDOCL - Open/Close bootstrap

IBMDOPU - Open (CONSECUTIVE BUFFERED/STREAM Files) - Level 3

Function

To complete the open process on CONSECUTIVE BUFFERED and STREAM files on tape or disk by initializing the DTF and issuing the OPEN macro instruction.

Method (chart DOPU)

IBMDOPU is called from the associated level open modules for ccnsecutive buffered and stream files if the file is on tape, disk or diskette. This module will initialize the DTF and issue the OPEN macro instruction, and return to the first level open module.

Linkage

```
R2    =  A(FCB)
R6    =  (ADTF)
```

Called by

IBMDOPQ/IBMDOPT - Open(level 2)

IBMDOPZ - Open (REGIONAL/INDEXED Files) - Level 3

Function

To calculate buffer and workspace requirements for INDEXED files, and obtain the required space.

Method (chart DOPZ)

IBMDOPZ is called from IBMDOPY for INDEXED files if the open process cannot be completed by the first level of open modules. Having calculated buffer and workspace requirements, and obtained space, IBMDOPZ calls IBMDCPW to initialize the DTF.

Linkage

R2 = A(FCB)
R6 = A(DTF)

Calls

IBMDOPW - Open (level 4)

Called by

IBMDOPY - Open(level 2)

IBMDOPW - Open (INDEXED Files) - Level 4

Function

To initialize the DTF with buffer addresses and issue an OPEN macro.

Method

When the module is entered, the buffer space will have already been obtained and the offsets of the buffer's various components saved temporarily in the FCB. These offsets are used to initialize the DTF with buffer and workspace addresses. For output and direct update files, the DTF is also initialized with device constants (such as track size). After the OPEN macro has been issued, control is returned to the first-level module, IBMDOPX.

Linkage

R2 = A(FCB)
R6 = A(DTF)

Called By

IBMDOPZ - Open (level 3).

IBMDOCA - Close

Function

To carry out final I/O operations on PL/I files and then to close them. The module has two entry points.

IBMDOCAA: Explicit close.

IBMDOCAB: Implicit close.

Method (chart DOCA)

For explicit close, the compiled code parameter list is scanned to find the files to be closed. In the case of implicit close, the open file chain is scanned instead.

Because only one CLOSE macro is issued for all files to be closed, the module obtains space for a parameter list, for use by the CLOSE macro.

For some files, I/O operations are necessary before that file can be closed. A related LOCATE statement must be completed and for regional (1) sequential output files, the remainder of the data set is formatted with dummy records. For indexed output files, an ENDFL macro is issued.

As each file is processed as described above, the address of its DTF is added to the parameter list, intended for the CLOSE macro. The module sets flags in the DTF for any tape files that may require repositioning so that the CLOSE macro can deal with such repositioning.

After issuing the CLOSE macro, all files closed are removed from the file chain and any relevant transmitters no longer required by other files in the now amended file chain are removed from the transmitter chain. Furthermore, if previously loaded, the error and endfile modules are deleted. For transmitters still required by open files, the responsibility count for each of those transmitters is reduced by the number of relevant files now closed. The STREAM INPUT and F-format STREAM PRINT transmitters are always link-edited and thus are not deleted.

Any event variables associated with operations on the closed files which have not been waited on, are completed. The FCB for each file closed is reset to its pre-OPEN state. The DTF for any sequential disk file that has been closed is also reset. The setting of the DTF for any other type of file closed is handled by the operating system and not by IBMDOCA.

Finally, control is returned to IBMDOCL.

Linkage

Entry point IBMDOCAA:

```
 R1   = A(PLIST)
 PLIST= A(number of files)
        A(file control block)
        A(disposition options)
```

The last two parameters are repeated for each file that is to be closed.

Entry point IBMDOCAB:

No parameters are passed.

Calls

IBMDRAW - Regional(3) sequential unbuffered output F- and U-format transmitter.
IBMDRAX - Regional(3) sequential buffered output F- and U-format transmitter.
IBMDRAY - Regional(1) unbuffered output F-format transmitter.
IBMBRAZ - Regional(1) buffered output F-format transmitter.
IBMDRLZ - Indexed sequential output F-format transmitter.
IBMDRRX - Consecutive sequential buffered U-format transmitter.
IBMDRRY - Consecutive sequential buffered V-format transmitter.
IBMDRRZ - Consecutive sequential buffered F-format transmitter.

Called By

IBMDOCL - Resident open/close bootstrap.

IBMDOCV - Close (VSAM)

Function

To carry out final I/O operations on PL/I VSAM files and then to CLOSE them, and free any associated storage.

Method

If the last operation on the VSAM file was a LOCATE statement, then the transmitter is called to output the last record. The file is then closed. Any active EVENT variable associated with the file is set complete and STATUS set to one. The IOCB, RPL and buffers are then freed, after which the ACB is also freed. The appropriate transmitter and record I/O error module (if it has been loaded) are then dechained from the transmitter chain and if their use count is zero, the storage is freed. The FCB is then taken out of the open file chain, and set to its pre open state. Finally, the module returns to IBMDOCA.

Linkage

Entry point IBMDOCVA

```
 R2   =  A(FCB) to close.
 R6   =  A(ACB)
```

Calls

IBMDRVZ - ESDS transmitter.
IBMDRVT - KSDS SEQ OUTPUT transmitter.
IBMDRVR - RRDS transmitter.
IBMDRVS - KSDS and PATH Input/Update transmitter.

Called By

IBMDOCA - Close module.

Record I/O under the DOS PL/I Optimizing Compiler is implemented using the logical input/output control system (LIOCS) routines of DOS data management. With the exception of a small resident interface module (IBMDRIO), all the modules concerned with record I/O are in the transient library.

RECORD I/O TRANSMITTER MODULES

The transmission of records is effected by a number of transmitter modules which issue the appropriate data management macro instructions. To minimize the size of the transmitter modules, each transmitter is designed to service only one type of PL/I file. A list of file types with their corresponding transmitter modules is given in figure 6.1.

Each transmitter module has a 16-byte prefix containing the following information:

Bytes 0-3	Length of module
Bytes 4-7	Chain back
Bytes 8-10	5th, 6th, and 7th letters of module name
Byte 11	Responsibility count
Bytes 12-15	Not used

The responsibility count in byte 11 indicates how many files are using the transmitter. When the first file requiring a particular transmitter is opened, the transmitter module is loaded into main storage and added to the chain of loaded transmitters. Its responsibility count is set to "1". After this, the responsibility count is increased by one each time a file requiring the transmitter is opened, and decreased by one each time a file that has been using the transmitter is closed. If at any time the responsibility count is reduced to zero, the transmitter is removed from the chain and the storage containing it is freed.

The transmitter chain contains all record I/O transmitters, stream I/O transmitters, and record I/O error modules that are currently in store.

A small exit module, IBMDRRR, exists and is used to handle exits from LIOCS that are due to ENDFILE or TRANSMIT conditions on consecutive buffered files.

VSAM TRANSMITTERS

Four transmitters are provided to handle PL/I files used with VSAM data sets. Both BUFFERED and UNBUFFERED PL/I files are handled in the same transmitter. The four transmitters are as follows:

IBMDRVZ -	ESDS Transmitter
IBMDRVT -	KSDS Sequential Output Transmitter
IBMDRVS -	KSDS or PATH INPUT/UPDATE/DIRECT Transmitter
IBMDRVR -	RRDS Transmitter

An ESDS is an Entry-Sequenced Data Set, a KSDS is a Key-Sequenced Data Set, and an RRDS is a Relative Record Data Set. A path is the combination of an ESDS or a KSDS and an Alternate Index, and is similar in structure to a KSDS.

All VSAM operations for PL/I support of ESDS, as a replacement for CONSECUTIVE organisation, are *addressed sequential* with OPTCD=(ADR,SEQ) in the Request Parameter List, henceforth abbreviated to RPL. Those for support of KSDS as an INDEXED replacement and for RRDS as a replacement for REGIONAL(1) are *keyed* and either *sequential* or *direct*, depending on the file type, with OPTCD=(KEY,SEQ) or (KEY,DIR) respectively in the RPL.

As already stated, both BUFFERED and UNBUFFERED PL/I file types will be handled by the same transmitter. In VSAM terms there is no difference, since the system always uses its own buffers. In PL/I, however, the BUFFERED attribute permits the use of locate mode I/O statements, whilst UNBUFFERED permits EVENT I/O.

The Use of IOCBs

Since both BUFFERED and UNBUFFERED file types are handled in the same transmitter, the IOCB provided by OPEN will always be used, to record information, including error codes, on a statement basis.

For VSAM a special extended form of IOCB is used. The additional fields include the following:

IDUB	A(dummy buffer)	used for LOCATE I/O on BUFFERED files.
IKSV,IKST	A(key save areas)	
IEVC	Data Management ECB	
IPTR	A(VSAM buffer) used with GET LOC.	
	space for MODCB plist to modify RPL parameters.	
	space for SHOWCB plist to display RPL parameters.	

There are also various element control entries for the RPL parameters requiring to be modified or displayed by the VSAM transmitters. The IOCB field names listed below are used as setting or receiving fields for the corresponding RPL parameters.

IOPT	OPTCD
IARA	AREA
IARL	AREALEN
IRCL	RECLEN
IKYL	KEYLEN
ISUA	FDBK/RBA/FTNCD

Locate Mode I/O and the Use of Dummy Buffers

OPEN provides a dummy buffer for use with all locate mode statements on buffered files. PL/I uses the VSAM LOCATE mode only in situations where the contents of the input record are irrelevant, as for READ IGNORE and for implied READs. However, for data sets which have spanned records LOCATE mode operations are not allowed, so the dummy buffer or a VDA is used.

Record Checking

For ESDS and KSDS all records are of variable length up to a maximum LRECL supplied to the DEFINE utility program. In these cases the RECORD condition checking rules for V and U formats will be applied. For LOCATE and WRITE statements, comparison will take place against the maximum value of LRECL; for READ statements comparison will be against the current length of the record read; for REWRITE statements on KSDS it will be against the maximum value of LRECL, and on ESDS, against the current length of the record to be updated. For LOCATE, WRITE and REWRITE statements on KSDS an additional RECORD condition will be detected, namely record variable shorter than keylength + relative key position. As in the case of a zero-length record variable on output, no data will be transmitted.

With the exception of the READ statement, all record checking will be handled entirely by the transmitter. A READ INTO will be implemented by a VSAM GET from the system buffer to the record variable. If the record variable is too short, VSAM will give a logical error return code and no transmission will take place. The transmitter will then reissue the request, provide an intermediate dummy buffer, and finally move the truncated record to the record variable.

Event I/O

All DOS VSAM operations are synchronous, and the return code will be tested by the transmitter immediately after issuing an action macro. Event I/O is simulated - that is if an error is detected for a statement with the EVENT option, it is held over until the corresponding WAIT statement.

DATA SET ORGANIZATION	FILE ATTRIBUTES		TRANSMITTER MODULE
REGIONAL(1)	SEQUENTIAL	UNBUFFERED OUTPUT F-format	IBMDRAY
		BUFFERED OUTPUT F-format	IBMDRAZ
		UNBUFFERED INPUT/UPDATE F-format	IBMDRBZ
		BUFFERED INPUT/UPDATE F-format	IBMDRBW
	DIRECT	F-format	IBMDRDZ
REGIONAL(3)	SEQUENTIAL	UNBUFFERED OUTPUT F/U-format	IBMDRAW
		BUFFERED OUTPUT F/U-format	IBMDRAX
		UNBUFFERED INPUT/UPDATE F/U-format	IBMDRBY
		BUFFERED INPUT/UPDATE F/U-format	IBMDRBX
	DIRECT	F/U-format	IBMDRDY
CONSECUTIVE	SEQUENTIAL	UNBUFFERED U-format	IBMDRCY
		UNBUFFERED F-format	IBMDRCZ
		BUFFERED F-format OMR	IBMDRRT
		BUFFERED U-format, Associate	IBMDRRU
		BUFFERED V-format, Associate	IBMDRRV
		BUFFERED F-format, Associate	IBMDRRW
		BUFFERED U-format	IBMDRRX
		BUFFERED V-format	IBMDRRY
		BUFFERED F-format	IBMDRRZ
INDEXED	SEQUENTIAL	INPUT/UPDATE F-format	IBMDRJZ
		OUTPUT F-format	IBMDRLZ
	DIRECT	INPUT/UPDATE F-format	IBMDRKZ
VSAM ESDS	SEQUENTIAL	ALL	IBMDRVZ
VSAM KSDS or PATH	SEQUENTIAL	OUTPUT	IBMDRVT
		INPUT/UPDATE	IBMDRVS
	DIRECT	ALL	IBMDRVS
VSAM RRDS	SEQL/DIRECT	ALL	IBMDRVR

Figure 6.1. Record I/O transmitter modules

Linkage for RECORD I/O Transmitters

The following parameters are passed in registers to the RECORD I/O transmitters by the resident interface module IBMDRIO:

```
R1 = A(compiled code parameter list)
R2 = A(file control block (FCB))
R4 = A(define the file (DTF) block)
R5 = A(request control block)
R6 = A(record descriptor) or A(ignore factor)
      or A(target for buffer address for READ SET statement)
      or A(pointer to target for buffer address after LOCATE)
R7 = A(key descriptor) or zero
R8 = A(event variable) or zero
```

The compiled code parameter list addressed by register R1 is the parameter list that the resident interface module receives from compiled code. It has the following format:

```
A(file control block)
A(request control block)
A(record descriptor) or A(ignore factor)
       or A(target for buffer address for READ SET statements)
       or A(pointer to target for buffer address after LOCATE)
A(key descriptor) or zero
A(event variable) or zero
A(abnormal locate return label) or zero
```

ERROR HANDLING

To avoid including large sections of similar code, which may not be required, in all the transmitter modules, the error handling code for RECORD I/O is contained in four error handling modules. These modules are:

IBMDREV	-	Error Module for all VSAM files.
IBMDREX	-	Error-handling module for INDEXED files.
IBMDREY	-	Error-handling module for REGIONAL files and for UNBUFFERED CONSECUTIVE files.
IBMDREZ	-	Error-handling module for BUFFERED CONSECUTIVE files.

A further module, IBMDREF, is provided to handle the ENDFILE condition. The error-handling modules, however, are also capable of handling the ENDFILE condition; the conditions under which the end of file module is used are described below.

Communication between the transmitters and the error and end of file modules is provided by fields in the FCB. During the opening of a file (chapter 4), fields FEMT and FEFT in the FCB are initialized to indicate the error module (i.e. IBMDREX, REY, or REZ) that corresponds to the file type. Also, field FERM is set to the address of a small resident bootstrap routine(part of IBMDRIO) that is used for loading the error modules.

When a transmitter module detects an error or a condition during an I/O operation, it first sets an error code into field FERR in the FCB. This field comprises two 1-byte fields FER1 and FER2. The meanings of the error codes in these two fields are shown in figure 6.2. Two fields are necessary because the TRANSMIT condition can be raised at the same time as the RECORD condition.

FER1

X'02'	=	input transmit
X'03'	=	output transmit
X'1A'	=	OMR read error
X'1C'	=	input transmit (index Set)
X'1D'	=	output transmit (index set)
X'1E'	=	input transmit (sequence set)
X'1F'	=	output transmit (sequence set)

FER2

X'01'	=	end of file
X'02'	=	(not used)
X'03'	=	(not used)
X'04'	=	zero length record variable
X'05'	=	short record variable
X'06'	=	long record variable
X'07'	=	key conversion
X'08'	=	key duplication
X'09'	=	key sequence
X'0A'	=	key specification
X'0B'	=	key not found
X'0C'	=	no space for keyed record
X'OD'	=	too many I/O events outstanding
X'0E'	=	active event
X'0F'	=	no prior read before rewrite
X'11'	=	permanent output error
X'12'	=	zero length record read
X'13'	=	record ref. out of data set limits
X'14'	=	unidentifiable I/O error
X'16'	=	no space for record in sequential output data set
X'18'	=	key conversion (negative binary number)
X'1B'	=	I/O sequence error (associated files)
X'21'	=	record length less than keylen + RKP
X'22'	=	record already held
X'23'	=	record on non-mounted volume
X'24'	=	data set cannot be extended
X'25'	=	no virtual storage for VSAM
X'26'	=	no keyrange for insertion
X'27'	=	no positioning for sequential read
x'28'	=	attempt to reposition failed
X'29'	=	STRNO for data set exceeded
X'2A'	=	index upgrade error
X'2B'	=	maximum number of index pointers exceeded
X'2C'	=	invalid alternate index pointer
X'2D'	=	invalid sequential write

Figure 6.2. RECORD I/O Error Codes

If the condition is ENDFILE, the transmitter module next changes field FEFT to indicate the end of file module IBMDREF. Finally, the transmitter branches to the address held in field FERM.

If the error or condition is the first that has been detected cn the file, FERM holds the address of the resident bootstrap routine. This routine loads the module specified in FEFT and puts its entry point address in FERM. It then branches to the the module that it has loaded. Thus if ENDFILE is the first condition that is raised on a file, the end of file module IBMDREF is loaded, rather than the error module. The end of file module is smaller and faster than the error modules; it thus enables the ENDFILE condition to be handled more efficiently.

Subsequent errors or conditions on the file cause the end of file module or the error module (whichever has been loaded) tc be entered directly. If IBMDREF is entered and the condition is not ENDFILE, this mcdule loads the error module, puts its address into FERM, and branches to the module that it has loaded. All subsequent conditions, including ENDFILE, are thus handled by the error module.

Note: For VSAM files all error conditions including ENDFILE are handled directly by IBMDREV.

MODULE DESCRIPTICNS

IBMDRAY - REGIONAL(1) SEQUENTIAL UNBUFFERED OUTPUT F-Format Transmitter

Function

To implement PL/I record statements for REGIONAL (1) SEQUENTIAL UNBUFFERED OUTPUT files by moving data from the PL/I variable to the data set. The module also handles final output operations when the file is closed.

Method (chart DRAY)

The module interfaces with data management direct access method (DAM). It is called once for each execution of the PL/I WRITE staterent; it is also called to handle a WAIT statement following an EVENT option in the WRITE statement. The main steps are as follows:

1. If module has been called to handle a wait statement, go to step 9.

2. Check for any outstanding events.

3. Check and activate event variatle, if the EVENT option is specified.

4. Check validity of key and raise KEY condition if invalid.

5. Check record length and raise RECORD condition if invalid.

6. If record is not to be placed in region adjoining that of previous record, write dummy records on intervening regions using the WRITE AFTER and WAITF macros.

7. Move the record from the PL/I variable into the tuffer and write the record in the required region ty means of a WRITE AFTER macro. This is followed by a WAITF macro if no EVENT macrc has been specified.

8. Return to compiled code if the EVENT option has been specified.

9. Issue WAITF macro.

10. If no errors have occurred, return to compiled code or the WAIT module; otherwise, call the error module.

11. When called by the Close module, the transmitter is passed the maximum region number, causing it to write dummy records in the remaining regions of the data set.

Linkage

R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD)
R7 = A(KD)
R8 = A(EV) or zero

```
PLIST = A(FCB)
        A(RCB)
        A(RD)
        A(KD)
        A(EV) or zero
        Zero
```

Calls

IBMDREY - Error module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDOCA - Close module.
IBMDRIO - Resident interface module.

IBMDRAZ - REGIONAL(1) SEQUENTIAL BUFFERED OUTPUT F-format Transmitter

Function

To implement PL/I RECORD output statements for REGIONAL (1) SEQUENTIAL BUFFERED files. The module moves data from the PL/I variable to the output buffer (if WRITE FROM is specified), and then from the buffer to the data set. The module also handles final output operations when the file is being closed.

Method (chart DRAZ)

The module interfaces with the DOS data management direct access method (DAM). Its processing is as follows:

1. If the previous PL/I output statement for this file was a LOCATE, issue a WRITE AFTER macro followed by a WAITF macro, to output record in that statement.

2. Check validity of key and raise KEY condition, if invalid.

3. Check the record length and raise RECORD condition, if necessary.

4. Issue a WAITF macro for previous PL/I output statement, if this was a WRITE statement.

5. If record is not to be placed in the region adjoining that cf the previous record, write dummy records cn intervening regions, using the WRITE AFTER and WAITF macros.

6. If the PL/I statement is a LOCATE and if no errors have been detected, set the buffer pointer to the current record and return control to the compiled code. If any errors have been detected, call the error mcdule.

7. If the PL/I statement is a WRITE, move the reccrd from the PL/I variable into the buffer and issue a WRITE AFTER macro tc mcve it from the buffer to the data set. If no errors have been detected, return control to compiled code; if there are any errors, call the error module.

8. If the module has been called by the Close module, write dummy records in all the remaining regions of the data set. (The close module, when it calls the transmitter, passes the maximum value for a region number. It is by trying to implement this call that the transmitter carries out these final steps).

Linkage

```
R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD) or A(address of slot for
       buffer pointer for LOCATE)
R7 = A(KD)
R8 = Zero

PLIST = A(FCB)
        A(RCB)
        A(RD) or A(address of buffer pointer for LOCATE)
        A(KD)
        Zero
        A(abnormal locate return label) or zero
```

Calls

IBMDREY - Error module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDOCA - Close module.
IBMDRIO - Resident interface module.

IBMDRBZ - REGIONAL(1) SEQUENTIAL UNBUFFERED INPUT/UPDATE F-Format Transmitter

Function

To implement PL/I RECORD input statements for REGIONAL (1), SEQUENTIAL, UNBUFFERED INPUT/UPDATE files by moving data from the data set to the PL/I variable. The module also implements record update statements for these files by moving data from a PL/I variable to the data set.

Method (chart DRBZ)

The module interfaces with data management direct access method (DAM). The records are transmitted by means of the READ ID and WRITE ID macros. Code is provided to handle all possible options in the PL/I statement and to handle a WAIT statement following an EVENT in the READ or REWRITE statement. The method is as follows:

1. If the call has been made to handle a WAIT statement, branch to step 4.1(3) or 4.3(4).

2. Check for any outstanding event.

3. Check and activate event variable, if EVENT option specified.

4.1 For READ statement:

(1) Issue READ ID macro.

(2) Set prior read flag.

(3) Issue WAITF macro (unless EVENT option is specified).

(4) Check record length and raise RECORD condition if necessary.

(5) Move record to record variable.

(6) If the KEYTO option is specified, convert the region number to CHARACTER and move it to the specified variable.

4.2 For READ with IGNORE(n) option:

(1) Clear prior read flag.

(2) Issue "n" READ ID macros and (n-1) corresponding WAITF macros. If an EVENT option has not been specified, issue the final WAITF macro.

4.3 For REWRITE statement:

(1) Check for prior READ statement.

(2) Check the variable specified in the FROM option and raise RECORD condition if necessary.

(3) Move record to buffer.

(4) Issue WRITE ID macro.

(5) If an EVENT option has not been specified, issue a WAITF macro.

5.1 Call error module if any error has been found, otherwise return to compiled code or the WAIT module.

In all cases the current region number is recorded in the FCB. The KEYTO option is implemented by converting the region number to CHARACTER and moving it to the key variable.

Linkage

R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD) or A(ignore factor)
R7 = A(KD) or zero
R8 = A(EV) or zero

```
PLIST = A(FCB)
        A(RCB)
        A(RD) or A(ignore factor)
        A(KD) or zero
        A(EV) or zero
        Zero
```

Calls

IBMDREY - Error module.
IBMDREF - End-of-file module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDRIO - Resident interface module.

IBMBRBW - REGIONAL(1) SEQUENTIAL BUFFERED INPUT/UPDATE F-Format Transmitter

Function

To implement PL/I RECORD input statements for REGIONAL(1) SEQUENTIAL, BUFFERED files by moving data from the data set into a buffer or a PL/I variable. The module also implements record update statements for these files by moving data from a PL/I variable or a buffer to the data set.

Method (chart DRBW)

The module interfaces with data management direct access method (DAM). Code is provided to handle all possible options in the PL/I statement as follows:

A. READ with INTO option:

1. Issue READ ID and WAITF macros to transmit data from the data set into a buffer.

2. Check record variable and raise RECORD condition, if necessary.

3. Move data from the PL/I variable to the buffer.

B. READ with SET option:

1. Issue READ ID and WAITF macros to transmit data from the data set into a buffer.

2. Set the pointer to the address of this buffer.

C. READ with IGNORE option:

1. Issue READ ID and WAITF macros n times.

D. REWRITE and REWRITE with FROM option:

1. Check that a prior READ has been issued.

2. If FROM is not specified, branch to step 5.

3. Check record variable and raise RECORD condition, if necessary.

4. Move data from the PL/I variable to the buffer.

5. Issue WRITE ID and WAITF macros to transmit the data from the buffer to the data set.

In all cases the current region number is recorded in the FCB. The KEYTO option is implemented by converting the region number to CHARACTER and moving it to the key variable.

Linkage

```
R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD) or A(ignore factor) or A(buffer
          address for READ SET statement)
R7 = A(KD) or zero
R8 = Zero

PLIST = A(FCB)
        A(RCB)
        A(RD or ignore factor or buffer address for READ SET
          statements)
        A(KD) or zero
        Zero
        Zero
```

Calls

IBMDREY - Error module.
IBMDREF - End-of-file module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDRIO - Resident interface module.

IBMDRDZ - REGIONAL(1) DIRECT F-Format Transmitter

Function

To implement PL/I RECORD I/O statements for REGIONAL(1), DIRECT files by transmitting data between the data set and the PL/I variable.

Method (chart DRDZ)

The module interfaces with data management direct access method (DAM). It is called once for each execution of the Pl/I READ, WRITE or REWRITE statement; it is also called to handle a WAIT statement following an EVENT option on these statements. Its main steps are:

1. If the module has been called to handle a WAIT statement, gc to step 6.
2. Check and activate event variable, if the EVENT option is specified.
3. Check validity of key and raise KEY ccndition if invalid.
4. For PL/I WRITE or REWRITE statements, check record variable, raise RECORD condition, if necessary, and move the record variable to the buffer.
5. (1) For PL/I READ issue READ ID macro.

   (2) For PL/I WRITE issue WRITE ID macro.

   (3) For PL/I REWRITE issue WRITE ID macro.
6. Issue WAITF macro (unless the EVENT option has been specified).
7. For a PL/I READ statement, check record variable, raise RECORD condition if necessary, and move data from buffer to record variable.
8. Return control to compiled code or the WAIT module; otherwise, call the error module.

Linkage

R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD)
R7 = A(KD)
R8 = A(EV) or zero

```
PLIST = A(FCB)
        A(RCB)
        A(RD)
        A(KD) or zero
        A(EV) or zero
        Zero
```

Calls

IBMDREY - Error module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDRIO - Resident interface module.

IBMDRAW - REGIONAL(3) SEQUENTIAL UNBUFFERED OUTPUT F- and U-Format Transmitter

Function

To implement PL/I RECORD output statements for REGIONAL(3), SEQUENTIAL, UNBUFFERED files by moving data from the PI/I variable to the data set. The module also handles final output operations when the file is closed.

Method (chart DRAW)

The module interfaces with the DOS data management direct access method

DAM. It is called once for each execution of the P1/I WRITE statement; it is also called to handle a WAIT statement following an EVENT option in the WRITE statement. The main steps are as fcllows:

1. If module has been called to handle a WAIT statement, proceed as detailed in step 8.

2. Check for any outstanding events.

3. Check and activate event variable, if EVENT option is specified.

4. Check validity of key, and raise KEY codition if invalid.

5. Check record variable and raise RECORD condition if necessary.

6. Move record to buffer.

7. Add the record to the required region by issuing the WRITE AFTER macro.

8. The corresponding WAITF macro is also issued if there is no EVENT option.

9. If called by the Close module, complete the previous output cperation by issuing a WAITF macro.

Linkage

R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD)
R7 = A(KD)
R8 = A(EV) or zero

```
PLIST = A(FBC)
        A(RCB)
        A(RD)
        A(KD)
        A(EV) or zero
        Zero
```

Calls

IBMDREY - Error module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDOCA - Close module.
IBMDRIO - Resident interface module.

IBMDRAX - REGIONAL(3) SEQUENTIAL BUFFERED OUTPUT F- and U-Format Transmitter

Function

To implement PL/I RECORD output statements for REGIONAI(3), SEQUENTIAL, BUFFERED files by moving data from the Pl/I variable to the output buffer (if WRITE FROM is specified) and from the buffer to the data set. The module also handles final output operations when the file is closed.

Method (chart DRAX)

The module interfaces with data mangement direct access method (DAM). Its processing is as follows:

1. If the previous PL/I output statement for this file was a LOCATE, issue WRITE AFTER and WAITF macros to output record in that statement.

2. Check validity of key and raise KEY condition, if invalid.

3. Issue WAITF macro for previous PL/I output statement (if this was a WRITE).

4. Move source key having the length specified in the ENVIRONMENT option into the buffer.

5. If the PL/I statement is a LOCATE and if no errors have been detected, set the buffer pointer to the current record and return control to the compiled code. If any errors have been detected, call the error module.

6. If the Pl/I statement is a WRITE, move the record from the PL/I variable into the buffer and issue a WRITE AFTER macro to move it from the buffer to the data set. If no errors have been detected, return control to compiled code; if there are any errors, call the error handler.

7. If called by the Close module, complete the previous output operation by issuing a WAITF macro.

Linkage

```
R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD) or A(address of buffer pointer for LOCATE)
R7 = A(KD)
R8 = Zero

PLIST = A(FCB)
        A(RCB)
        A(RD)
        A(KD) or zero
        Zero
        A(abnormal locate return label) or zero
```

Calls

IBMDREY - Error module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDOCA - Close module.
IBMDRIO - Resident interface module.

IBMDRBY - REGIONAL(3) SEQUENTIAL UNBUFFERED INPUT/UPDATE F- and U-Format Transmitter

Function

To implement PL/I RECORD input statements for REGIONAL(3), SEQUENTIAL, UNBUFFERED input files by moving data from the data set to the Pl/I variable. The module also implements record update statements for these files by moving data from a PL/I variable to the data set.

Method (chart DRBY)

The module interfaces with data management direct access method (DAM). The records are transmitted by means of the READ ID and WRITE ID macros.

Code is provided to handle all possible options in the PI/I statement and to handle a WAIT statement fcllowing an EVENT option in the READ or REWRITE statement. The method is as follows:

1.1 If the call has been made to handle a WAIT statement, branch to step 4.1(2) or 4.3(4).

2.1 Check for any outstanding events.

3.1 Check and activate event variable, if EVENT option is specified.

4.1 For READ statement:

(1) Issue READ ID macro.

(2) Issue WAITF macro (unless EVENT option is specified).

(3) Check record variable and raise RECORD condition if necessary.

(4) Move record from buffer to record variable.

(5) If the KEYTO option is specified, convert the region number to CHARACTER and move it to the specified variable.

4.2 For READ with IGNORE (n) option:

(1) Issue "n" READ ID macros and (n-1) corresponding WAITF macros.

(2) If an EVENT option has not been specified, issue the final WAITF macro.

4.3 For REWRITE statement:

(1) Check for prior READ statement.

(2) Check the variable specified in the FROM option and raise the RECORD condition if necessary.

(3) Move record to buffer.

(4) Issue WRITE ID macro.

(5) If an EVENT option has not been specified, issue a WAITF macro.

5.1 Call error module if any error has been found, otherwise, return to compiled code or the WAIT module.

Linkage

R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD) or A(ignore factor) or zero
R7 = A(KD) or zero
R8 = A(EV) or zero

```
PLIST = A(FCB)
        A(RCB)
        A(RD) or A(ignore factor) cr zero
        A(KD) or zero
        A(EV) or zero
        Zero
```

Calls

IBMDREY - Error module.
IBMDREF - End-of-file module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDRIO - Resident interface module.

IBMBRBX - REGIONAL(3) SEQUENTIAL BUFFERED INPUT/UPDATE F- and U-Format Transmitter

Function

To implement PL/I RECORD input statements for REGIONAL(3), SEQUENTIAL, BUFFERED files by moving data from the data set into a buffer cr PL/I variable. The module also implements RECORD UPDATE statements fcr these files by moving data from a PL/I variable or buffer to the data set.

Method (chart DRBX)

The module interfaces with data mangement direct access method (DAM). The records are transmitted by means of the READ ID and WRITE ID macros. Code is provided to handle all possible PL/I statements, as follows:

A. READ statement:

1. Issue READ ID macro.

2. Issue WAITF macro.

3. If the KEYTO option is specified, move the key from the buffer to the specified variable.

4. If the SET option is specified, set pointer to address of buffer.

5. If INTO was specified, check the record, raise the RECORD condition if necessary, and move the record to the PL/I variable.

6. Call error module to raise any errors; otherwise, return to compiled code.

B. READ with IGNORE(n) option:

1. Issue "n" READ ID and WAITF macros.

C. REWRITE statement:

1. Check for prior READ statement.

2. If FROM is specified, check the record variable, raise the RECORD condition if necessary, and move the data from the PL/I variable to the buffer.

3. Issue WRITE ID macro.

4. Issue WAITF macro.

5. Call the error module to raise errors if required, or return to compiled code.

Linkage

```
R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD or ignore factor or buffer address
       for READ SET)
R7 = A(KD) or zero
R8 = Zero

PLIST = A(FCB)
        A(RCB)
        A(RD or ignore factor or buffer address for
          READ SET statements)
        A(KD) or zero
        Zero
        Zero
```

Calls

IBMDREY - Error module.
IBMDREF - End-of-file module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDRIO - Resident interface module.

IBMDRDY - REGIONAL(3) DIRECT F- and U-Format Transmitter

Function

To implement PL/I RECORD I/O statements for REGIONAL(3), DIRECT files by transmitting data between the data set and the PL/I variable.

Method (chart DRDY)

The module interfaces with data management direct access method (DAM). It is called once for each execution of the PL/I READ, WRITE, or REWRITE statement; it is also called to handle a WAIT statement following an EVENT option in the READ, WRITE, or REWRITE statement. Its main steps are:

1. If handling a WAIT statement, go to step 7.

2. Check for outstanding events.

3. Check and activate event variable, if EVENT option specified.

4. Check validity of key and raise KEY condition if invalid.

5. For PL/I WRITE and REWRITE, check record variable, raise RECORD condition if necessary, and move record to buffer.

6. (1) For PL/I READ issue READ KEY macro.

   (2) For PL/I WRITE issue WRITE AFTER macro.

   (3) For PL/I REWRITE issue WRITE KEY macro.

7. If an EVENT option has not been specified, issue a WAITF macro.

8. Check record variable, raise RECORD condition if necessary, and move data from buffer to PL/I variable (READ statement).

9. Call error module to raise any errors. Otherwise return to compiled code or the WAIT module.

Linkage

```
R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD)
R7 = A(KD)
R8 = A(EV) or zero

PLIST = A(FCB)
        A(RCB)
        A(RD)
        A(KD)
        A(EV) or zero
        Zero
```

Calls

IBMDREY - Error module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDRIO - Resident interface module.

IBMDRCY - CONSECUTIVE SEQUENTIAL UNBUFFERED U-Format Transmitter

Function

To implement PL/I record I/O statements for CONSECUTIVE UNBUFFERED files.

Method (chart DRCY)

The module interfaces with the data management sequential access method (SAM) using the workfile macros. The module is called once for each execution of the PL/I READ or WRITE statement; it is also called to handle a WAIT statement following an EVENT option on the READ or WRITE statement. The method is as follows:

1. Raise ERROR if there are any outstanding events.
2. Raise ERROR if the EVENT option specifies an already active event.
3. Activate any event specified and set fields in the FCB for the corresponding WAIT statement.

The module then proceeds as follows:

READ INTO:

1. Commence record checking, saving any values for the WAIT statement.

2. If the record variable is shorter than the maximum length cf the record, get buffer space (BACKWARDS option cnly).

3. If the file has the BACKWARDS option, point at the end of either the buffer or the record variable.

4. Issue READ macro.

* 5. Issue CHECK macro.

* 6. Complete the record checking.

* 7. If a buffer was obtained, move the record to the record variatle.

* 8. Raise any conditions.

* 9. Return to compiled code or to the WAIT module.

READ IGNORE:

1. Issue N-1 READ and CHECK macros, for one byte of each record.

2. Issue READ macro.

* 3. Issue CHECK macro.

* 4. Return to compiled code or the WAIT module.

WRITE and REWRITE:

1. Raise ERROR if no prior READ (REWRITE only).

2. Check record variable.

3. Issue WRITE macro.

* 4. Issue CHECK macro.

* 5. Raise any conditions.

* 6. Return to compiled code or the WAIT module.

Note: If the EVENT option is specified, then those actions marked above with "*" are carried out consequent to the corresponding WAIT statement.

EOFADDR

When ENDFILE is first raised, the EOFADDR routine replaces the transmitter address in the FCB with the address cf a routine in the transmitter. The routine thus addressed will raise ENDFIIE on any subsequent READ, or will return tc compiled code if EVENT is specified.

ERROPT

When a TRANSMIT condition is first raised, and a permanent cutput error condition is found, the ERROPT routine replaces the address of the transmitter in the FCB, so that subsequent WRITE statements will cause a branch to be taken to a routine in the transmitter. This routine raises the permanent output error condition.

Linkage

```
R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD) or A(ignore factor)
R7 = zero
R8 = A(EV) or zero

PIIST = A(FCB)
        A(RCB)
        A(RD) or A(ignore factor)
        zero
        A(EV) or zero
        zero
```

Calls

IBMDREY - Error module.
IBMDREF - End-of-file module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDRIO - Resident interface module.

IBMDRCZ - CONSECUTIVE SEQUENTIAL UNBUFFERED F-format Transmitter

Function

To implement PL/I record I/O statements for SEQUENTIAL UNBUFFERED files with F-format records, by interfacing with the sequential access method and using the WORKFILE macros.

Method (chart DRCZ)

1. Raise ERROR if there are any outstanding events.

2. Raise ERROR if the EVENT opticn specifies an already active event.

3. Activate any event specified and set fields in the FCB fcr the corresponding WAIT statement.

The module then proceeds as follows:

READ INTO:

1. Commence record checking, saving values for the WAIT statement.

2. If the record variatle is too small, get buffer space.

3. If the file has the BACKWARDS option, point at the end of either the buffer or the record variatle.

4. Issue READ macro.

* 5. Issue CHECK macro.

* 6. Complete the record checking.

* 7. Move record to record variable, if the READ was into a buffer.

* 8. Free any hidden buffer.

* 9. Raise any conditions.

*10. Return to compiled code or to the WAIT module.

READ IGNORE:

1. Get space for buffer.

2. Issue N-1 READ and CHECK macros.

3. Issue READ macro.

* 4. Issue CHECK macro.

* 5. Free hidden buffer.

* 6. Return to compiled code or the WAIT module.

WRITE and REWRITE:

1. Raise ERROR if no prior READ (REWRITE only).

2. Check for the RECCRD conditicn.

3. If the record variatle is too small, get space for buffer and move in the record.

4. Issue WRITE macro.

* 5. Issue CHECK macro.

* 6. Complete the record checking.

* 7. Free any hidden buffer.

* 8. Raise any conditions.

* 9. Return to compiled code or the WAIT module.

Note: If the EVENT option is specified, those actions marked above with "*" are carried out consequent to the corresponding WAIT statement.

EOFADDR

When ENDFILE is first raised, the EOFADDR routine replaces the transmitter address in the FCE with the address of a routine in the transmitter. The routine thus addressed will raise ENDFILE on any subsequent READ or will return to compiled code if EVENT is specified.

ERROPT

When a TRANSMIT condition is first raised, and a permanent output error condition is found, the ERROPT routine replaces the address of the transmitter in the FCB, so that subsequent WRITE statements will cause a branch to be taken to a routine in the transmitter. The latter routine raises the permanent output error condition.

Linkage

```
R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD) or A(ignore factor)
R7 = Zero
R8 = A(EV) or zero

PLIST = A(FCB)
        A(RCB)
        A(RD) or A(ignore factor)
        zero
        A(EV) or zero
        zero
```

Calls

IBMDREY - Error module.

IBMDREF - End-of-file module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDRIO - Resident interface module.

IBMDRRT - CONSECUTIVE SEQUENTIAL BUFFERED F-format OMR Transmitter

Function

To read cards containing optical mark read (OMR) data for record I/O statements on CONSECUTIVE BUFFERED F-format files. It does so by interfacing with the sequential access method.

Method (chart DRRT)

READ INTO:

1. Issue GET macro to read a record.
2. Check the length of the record.
3. Move the record to the variable.
4. Test for OMR READ errors.
5. Issue a CNTRL macro to select a stacker for the card just read.
6. Raise any errors.
7. Return to compiled code.

READ SET:

1. Issue GET macro to read a record.
2. Set pointer to record in buffer.
3. Test for OMR READ errors.
4. Issue a CNTRL macro to select a stacker for the card just read.
5. Raise any errors.
6. Return to compiled code.

Linkage

```
R1 = A(PLIST)
R2 = A(FCB)
R3 = A(DTF)
R5 = A(RCB)
R6 = A(RD) or A(ignore factor)
R7 = Zero

PLIST = A(FCB)
        A(RCB)
        A(RD) or A(ignore factor)
        Zero
        Zero
        A(abnormal LOCATE return label)
```

Calls

IBMDREZ - Error module.
IBMDREF - End-of-file module.

Called By

IBMDOCA - Close module.
IBMDRIO - Resident interface module.

IBMDRRU - CONSECUTIVE SEQUENTIAL BUFFERED ASSOCIATE U-format Transmitter

Function

To implement PL/I record I/O statements for CONSECUTIVE BUFFERED ASSOCIATE U-format files, by interfacing with the sequential access method.

Method (chart DRRU)

WRITE statement:

1. Check record condition.
2. Move record to buffer.
3. Check I/O sequence.
4. Issue PUT macro.
5. Raise any conditions.
6. Return to compiled code.

LOCATE statement:

1. Check I/O sequence.
2. Issue PUT macro for previous record.
3. Check record variable.
4. Set pointer.
5. Return to compiled code.

Linkage

R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD)
R0 = Zero

```
PLIST = A(FCB)
        A(RCB)
        A(RD)
        Zero
        Zero
        A(abnormal LOCATE return label)
```

Calls

IBMDREZ - Error module.
IBMDREF - End-of-file module.

Called By

IBMDOCA - Close module.
IBMDRIO - Resident interface module.

IBMDRRV - CONSECUTIVE SEQUENTIAL BUFFERED ASSOCIATE V-format Transmitter

Function

To implement PL/I record I/O statements for CONSECUTIVE BUFFERED ASSOCIATE V-format files, by interfacing with the sequential access method.

Method (chart DRRV)

WRITE statement:

1. Check for the RECORD condition.
2. Move the record to the buffer.
3. Check I/O sequence.
4. Issue PUT macro.
5. Raise any conditions.
6. Return to compiled code.

LOCATE statement:

1. Check I/O sequence.
2. Issue PUT macro for previcus record.
3. Check the record variable.
4. Set pointer.
5. Return to compiled code.

Linkage

```
R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD)
R0 = Zero

PLIST = A(FCB)
        A(RCB)
        A(RD)
        Zero
        Zero
        A(abnormal LOCATE return label)
```

Calls

IBMDREZ - Error module.
IBMDREF - End-of-file module.

Called By

IBMDOCA - Close module.
IBMDRIO - Resident interface module.

IBMDRRW - CONSECUTIVE SEQUENTIAL BUFFERED ASSOCIATE F-format Transmitter

Function

To implement PL/I record I/O statements for CONSECUTIVE BUFFERED ASSOCIATE F-format files, by interfacing with the sequential access method.

Method (chart DRRW)

WRITE statement:

1. Check for the RECORD condition.
2. Move the record to the buffer.
3. Check I/O sequence.
4. Issue PUT macro.
5. Raise any conditions.
6. Return to compiled code.

LOCATE statement:

1. Check I/O sequence.

2. Issue PUT macro for previous record.

3. Check the record variable.

4. Set pointer.

5. Return to compiled code.

READ INTO:

1. Check I/O sequence.

2. Issue GET macro.

3. Check the record length.

4. Move record to PL/I variable.

5. Raise any conditions.

6. Return to compiled code.

READ SET:

1. Check I/O sequence.

2. Issue GET macro.

3. Raise any conditions.

4. Return to compiled code.

Linkage

```
R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD) or A(ignore factor)
R0 = Zero

PLIST = A(FCB)
        A(RCB)
        A(RD) or A(ignore factor)
        Zero
        Zero
        A(abnormal LOCATE return label)
```

Calls

IBMDREZ - Error module.
IBMDREF - End-of-file module.

Called By

IBMDOCA - Close module.
IBMDRIO - Resident interface module.

IBMDRRX - CONSECUTIVE SEQUENTIAL BU'FEREC U-format Transmitter

Function

To implement PL/I record I/O statements for CONSECUTIVE BUFFERED files.

Method (chart CRRX)

The module interfaces with the data management sequential access method (SAM). It is called once for each execution of the FL/I READ cr WRITE statement. Its main steps are:

WRITE statement:

1. Check for the RECORD condition.
2. Issue PUT macro (for the previous record).
3. Raise any conditions.
4. Return to compiled code.

LOCATE statement:

1. Check record variable and if in error, raise RECORD condition and return to compiled code.
2. Issue PUT macro (for the previous record).
3. Set pointer.
4. Return to compiled code.

Note: The first time the transmitter is called for a particular file, the PUT macro is not issued, since the address in the CTF will cause a tranch to an initial PUT routine contained in module IBMDRRR. This routine will set the address of LIOCS in the DTF, for subsequent staterents. The pointer for the first record is obtained from the FCB, having been placed there by the OPEN macro.

READ INTO:

1. Issue GET macro.
2. Check the record.
3. Move record to the variable from the buffer.
4. Raise any conditions.
5. Return to compiled code.

READ SET:

1. Issue GET macro.

2. Set the pointer. (If the EACKWARDS option is specified, a move may be required to ensure alignment on a doubleword boundary).

3. Raise any conditions.

4. Return to compiled code.

REWRITE:

1. Check for prior READ.

2. Check for the RECORD condition (if FROM option is specified).

3. Move record to buffer if FROM option specified.

4. Issue PUT macro.

5. Raise any conditions.

6. Return to compiled code.

ERROPT and EOFADDR

The ERROPT and EOFADDR routines are contained in a separate module IBMDRRR. LIOCS will branch directly to the appropriate rcutine in IBMDRRR if the TRANSMIT condition or the ENDFILE condition is detected.

Linkage

```
R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD) or A(ignore factor) or zero
R7 = zero

PLIST = A(FCB)
        A(RCB)
        A(RD) or A(ignore factor) or zero
        zero
        zero
        A(abnormal LOCATE return label)
```

Calls

IBMDREZ - Error module.
IBMDREF - End-of-file module.

Called By

IBMDOCA - Close module.
IBMDRIO - Resident interface module.

IBMDRRY - CONSECUTIVE SEQUENTIAL BUFFERED V-Format Transmitter

Function

To implement PL/I record I/O statements for CONSECUTIVE BUFFERED files.

Method (chart DRRY)

The module interfaces with the data management sequential access method (SAM). It is called once for each execution of the PL/I READ or WRITE statements. Its main steps are detailed below:

WRITE statement:

1. Check for the RECORD condition.

2. Issue PUT macro (for the previous record).

3. Issue TRUNC macro if there is not enough room in the buffer for the new record.

4. Raise any conditions.

5. Return to compiled code.

LOCATE statement:

1. Check record variable and if in error, raise the RECORD condition and return to compiled code.

2. Issue PUT macro (for the previous record).

3. Issue TRUNC macro if there is not enough room in the buffer for the new record.

4. Set pointer.

5. Return to compiled code.

Note: The first time the transmitter is called for a particular file, the PUT macro is not issued, since the address in the DTF will cause a branch to an initial PUT routine contained in module IBMDRRR. The initial PUT routine will set the address of LIOCS in the DTF, for subsequent statements. The pointer for the first record is obtained from the FCB, having been set there by the OPEN macro.

READ INTO:

1. Issue GET macro.

2. Check the record.

3. Move record to the variable from the buffer.

4. Raise any conditions.

5. Return to compiled code.

READ SET:

1. Issue GET macro.

2. Set pointer.

3. Raise any conditions.

4. Return to compiled code.

REWRITE:

1. Check for prior READ.

2. Check record condition (FROM specified).

3. Move record to buffer if FROM option specified.

4. Issue PUT macro.

5. Raise any conditions.

6. Return to compiled code.

ERROPT and EOFADDR

The ERROPT and EOFADDR routines are contained in a separate module IBMDRRR. LIOCS will branch directly to the appropriate routine in IBMDRRR if the TRANSMIT condition or the ENDFILE condition is detected.

Linkage

```
R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD) or A(ignore factor) or zero
R7 = zero

PLIST = A(FCB)
        A(RCB)
        A(RD) or A(ignore factor) cr zerc
        zero
        zero
        A(abnormal LOCATE return label)
```

Calls

IBMDREZ - Error module.
IBMDREF - End-of-file module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDOCA - Close module.
IBMDRIO - Resident interface module.

IBMDRRZ - CONSECUTIVE SEQUENTIAL BUFFERED F-format Transmitter

Function

To implement PL/I record I/O statements for CONSECUTIVE BUFFEREL files.

Method (chart DRRZ)

WRITE statement:

1. Check for the RECORD conditicn.

2. Issue PUT macro (for the previous record).

3. Raise any conditions.

4. Return to compiled code.

LOCATE statement:

1. Check record variable and if in error, raise RECORD condition and return to compiled code.

2. Issue PUT macro (for the previous record).

3. Set pointer.

4. Return to compiled code.

Note: The first time the transmitter is called for a particular file, the PUT macro is not issued, since the address in the DTF will cause a branch to an initial PUT routine contained in module IBMDRRR. The initial PUT routine will set the address of LIOCS in the DTF, for subsequent statements. The pointer fcr the first record is obtained from the FCB, having been set there by the OPEN macro.

READ INTO:

1. Get a record. (For blocked records, using disk or magnetic tape, deblocking is done by the transmitter. A GET macro is issued only at the end of a block.)

2. Check the record.

3. Move record to the variable from the buffer.

4. Raise any conditions.

5. Return to compiled code.

READ SET

1. Get a record. (For blocked records, using disk or magnetic tape, deblocking is done by the transmitter. A GET macro is issued only at the end of a block.)

2. Set pointer (a move may be required to ensure doubleword alignment, if the BACKWARDS option is specified).

3. Raise any conditions.

4. Return to compiled code.

REWRITE:

1. Check for prior READ.

2. Check record condition (FROM specified).

3. Move record to buffer if the FROM option is specified.

4. Issue PUT macro.

5. Raise any conditions.

6. Return to compiled code.

ERROPT and EOFADDR

The ERROPT and EOFADDR routines are contained in a separate module IBMDRRR. LIOCS will branch directly to the appropriate rcutine in IBMDRRR if the TRANSMIT conditicn or the ENDFILE condition is detected.

Linkage

R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD) or A(ignore factor) or zero
R7 = zero

```
PLIST = A(FCB)
        A(RCB)
        A(RD) or A(ignore factor) cr zerc
        zero
        zero
        A(abnormal LOCATE return label) or zero
```

Calls

IBMDREZ - Error module.
IBMDREF - End-of-file module.
Bootstrap routine in IBMDRIO (resident interface module).

Called By

IBMDOCA - Close module.
IBMDRIO - Resident interface module.

IBMDRRR - CONSECUTIVE BUFFERED Exit Module

Function

To provide TRANSMIT and ENDFILE exits for all CONSECUTIVE BUFFERED RECORC files. The module also contains a routine used when the initial PUT statement is issued. The module has three entry points:

IBMDRRRX: ERROPT routine.

IBMDRRRZ: EOFADDR routine.

IBMDRRRI: Initial PUT routine.

Method (chart DRRR)

The module interfaces with data management sequential access method (SAM).

ERROPT routine:

The transmit error flags are set, except in the case of a READ IGNORE statement. A return is then made to LIOCS (where possible using an ERET macro), except in the case of magnetic tape cutput, which would ABENC. For magnetic tape output, the transmitter address in the FCB and the address of LIOCS in the DTF are replaced by the address of a routine in the exit module, so that permanent output error is raised on any subsequent statement, and one of two courses is taken:

For library-call I/O, the error module is called.

For inline I/O, a return is made via the label constant.

EOFADDR routine:

When ENDFILE is first raised, the transmitter address in the FCB and the address of LIOCS in the DTF are replaced by the address of a routine in the exit module. The routine thus addressed will cause ENDFILE or NO PRIOR READ ERROR to be raised on subsequent statements.

Initial PUT routine:

When an OUTPUT file is opened, the address of LICCS in the DTF points to the initial PUT routine. The first PUT issued causes a branch to the routine. When branched to, the routine sets the address of LIOCS in the DTF, fcr subsequent statements. The pointer for the first record is obtained from the FCB, having been placed there by the OPEN macro.

Error and Exceptional Conditions

For ERROPT in the case of a permanent output error:

a) Inline I/O: Return via label constant.

b) Library call I/O: Call error module.

For EOFADDR:

a) Inline I/O: Return via label constant.

b) Not inline I/O: Call the error module.

Linkage

R1 = A(DTF)
R2 = A(FCB)
R5 = A(RCB) (only if transmitter issued PUT)
R8 = A(Label constant) for return to In-line code

Calls

IBMDREZ - Error module.

Called By

Entry points IBMDRRRX and IBMDRRRZ:

LIOCS - During the execution of a GET or PUT statement.

Entry point IBMDRRRI :

Compiled code or the appropriate transmitter.

IBMDRJZ - INDEXED SEQUENTIAL INPUT/UPDATE F-format Transmitter

Function

To implement PL/I record statements for SEQUENTIAL INDEXED INPUT and UPDATE files by transmitting data from a data set to the PL/I variable (READ statement) or vice versa (REWRITE statement).

Method (chart DRJZ)

The module interfaces with data management indexed sequential access method (ISAM). The access method does not support locate mode processing, but this is simulated by the module. IOREG is specified in the DTF so that Register 11 points to the beginning of the record in the I/O area; the register is then used to set the PL/I pointer when a READ SET statement is executed. The I/O area is acquired by the open module and is aligned on a double word boundary. A dummy buffer is required to align overflow records because of the ten byte link field that precedes the record itself. The module's main processing steps are as follows:

READ statement:

1. If READ KEY statement, the module issues ESETL and SETL macros to position file at key specified, after ensuring that the key specified is valid.

2. If record is not found, respositioning to the next higher key is done in the error module.

3. Issue GET macro.

4. If PL/I INTO option specified, check record variable, raise RECORD condition if necessary, and move record from I/O area into the variable.

5. If PL/I SET is option specified, set pointer to I/O area (or dummy buffer, if overflow record. In such a case, the record is first moved from the I/O area to the dummy buffer).

6. If the KEYTO option is specified, the key is moved to KEYTO after moving the record to the specified variable. In this way KEYTO is valid even if it is overlapped by the record variable.

7. Call error module if any errors have been raised. Otherwise, return to compiled code.

REWRITE statement:

1. Check that the previous PL/I I/O statement for this file was a READ.

2. Ensure that new embedded key is valid, by overwriting it with READ KEY.

3. Check size of record variable.

4. If FROM option not specified in REWRITE statement, and if READ statement utilized dummy buffer, move record from buffer to I/O area.

5. If FROM option specified, move record from variable to I/C area.

6. Issue PUT macro.

7. Call error module if any errors have been raised.

Linkage

```
R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD) or A(ignore factor)
R7 = A(KD) or zero
R8 = zero

PLIST = A(FCB)
        A(RCB)
        A(RD) or A(ignore factor) cr
            A(buffer address for READ SET statements)
        A(KD) or zero
        zero
        A(abnormal locate return) cr zero
```

Calls

IBMDREX - Error module.
IBMDREF - End-of-file module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDRIO - Resident interface module.

IBMDRLZ - INDEXED SEQUENTIAL OUTPUT F-format Transmitter

Function

To implement PL/I record statements for INDEXED SEQUENTIAL CUTPUT files by transmitting data from a PL/I variable to the data set.

Methcd (chart DRLZ)

The module interfaces with data management indexed sequential access method (ISAM). The access method does not support locate mode processing, but this is simulated by the module. When a PI/I LCCATE statement is executed, the module sets a register to point to the record position in the WORKL work area; this area is acquired by the open module and aligned so that the based variable is cn a double word boundary. The module also handles any outstanding output operations when the file is closed. The module's main processing steps are as follows.

1. If the previous PL/I output statement for this file was a LOCATE, overwrite the embedded key in WORKL with the stored KEYFROM string, and issue WRITE NEWKEY macro.

2. Check current KEYFROM string; raise KEY condition if invalid.

3. If current PL/I statement is LOCATE, store KEYFROM string.

4. Check record length and raise RECORD condition if invalid.

5. If the current PL/I statement is a WRITE, construct current record in WORKL and issue the WRITE NEWKEY macro. An embedded key is first overwritten by the KEYFROM string.

6. If the current PL/I statement is a LOCATE, check for valid key sequence, move the record key to WORKL if the format is unblocked, and set the pointer to the record area of WORKL.

7. Call the error module if any errors are indicated; otherwise, return to compiled code.

Linkage

R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD)
R7 = A(KD)
R8 = zero

```
PLIST = A(FCB)
        A(RCB)
        A(RD) or A(buffer address for LOCATE statements)
        A(KD) or zero
        zero
        A(abnormal locate return) or zero.
```

Calls

IBMDREX - Error module.
Bootstrap routine in IBMDRIO (Resident interface module).

Called By

IBMDRIO - Resident interface module.

IBMDRKZ - INDEXED DIRECT INPUT/UPDATE F-format Transmitter

Function

To implement PL/I record statements for DIRECT INDEXED INPUT and UPDATE files by transmitting data from a data set to the PL/I variable (READ statement) and vice versa (WRITE and REWRITE statements). The module is also called to handle a WAIT statement following an EVENT option in the I/O statement.

Method (chart DRKZ)

The module interfaces with data management indexed sequential access method (ISAM). IOREG is specified in the DTF so that processing may be carried out in the I/O area during the execution of REWRITE statements. WORKL is specified in the DTF and records to be added to the data set are constructed in this work area. The main processing steps are as follows.

1. If call was made to handle a WAIT statement, branch to check phase.

I/O Phase:

2. Activate any event variable.

3. Check validity of record and key.

4. If PL/I statement is a READ, issue READ macro to transmit record to I/O area.

5. If the PL/I statement is a REWRITE, and if the previous I/O statement for this file was not a READ, execute an implicit READ for the REWRITE key.

6. If the PL/I statement is a REWRITE, move record from the variable to I/O area, and issue WRITE KEY macro to transmit it to the data set.

7. If PL/I statement is a WRITE, move record from the variable to work area and issue WRITE NEWKEY macro to transmit it to the data set.

8. If EVENT option specified, return to calling routine.

Check phase:

9. Issue WAITF macro.

10. In PL/I statement is a READ, and if input is completed successfully, move record from I/O area to variable.

11. If no errors have been raised, free the event variable and return to compiled code. Otherwise, call the error module.

Linkage

```
R1 = A(PLIST)
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R6 = A(RD) or A(ignore factor) or zero
R7 = A(KD) or zero
R8 = A(EV) or zero

PLIST = A(FCB)
        A(RCB)
        A(RD) or A(ignore factor) or
         A(buffer address for READ SET statements)
        A(KD) or zero
        A(EV) or zero
```

Calls

IBMDREX - Error module.
Bootstrap routine in IBMDRIO (resident interface module).

Called By

IBMDRIO - Resident interface module.

PL/I FILES AND VSAM TRANSMITTERS

Figure 6.3 gives details relevant to the handling of PL/I files by VSAM transmitters. The explanatory notes following the figure should be read in conjunction with the transmitter descriptions given in the ensuing pages.

IBMDRVZ - ESDS Transmitter

Method (see chart DRVZ)

Depending on the statement code in the request control block, the transmitter performs any necessary record checking and then sets up and issues the appropriate macro request (GET or PUT).

If the EVENT option is in effect, the event variable is activated for the duration of the operation (until the WAIT statement has been executed.)

External Modules

1. (TOVV in TCA) - Overflow routine for GET VDA.

Exit

1. Normal: To compiled code or the wait module, on return to IBMDOCV, via link register

2. Error: To IBMDRIOB or IBMDREVA

Error and Exceptional Conditions

1. ENDFILE - end of file encountered on any READ
2. RECORD - short record variable on READ INTO
3. RECORD - zero-length record variable on WRITE/REWRITE FROM or LOCATE
4. RECORD - long record variable on WRITE/REWRITE FROM or LOCATE
5. TRANSMIT - read error in data set
6. TRANSMIT - write error in data set
7. ERROR - outstanding operation on file
8. ERROR - event variable already active (statement with EVENT)
9. ERROR - no prior READ for REWRITE [FROM]
10. ERROR - data already held in exclusive control
11. ERROR - record on non-mounted volume

12. ERROR - data set cannot be extended
13. ERROR - insufficient virtual storage to finish request
14. ERROR - no positioning for sequential READ
15. ERROR - too many concurrent operations on data set
16. ERROR - error in index upgrade
17. KEY - key invalid

IBMDRVT - KSDS Sequential Output Transmitter

Method (see chart DRVT)

Depending on the statement code in the request control block, the transmitter performs any necessary key checking and record checking and then sets up the appropriate macro request(s). The PUT macro is issued for a prior LOCATE.

If the EVENT option is in effect, the event variable is activated for the duration of the operation.

For sequential output (the only provision in PL/I for loading a KSDS), the records must be presented in ascending key sequence. The transmitter will check for this on both 'initial' load and any subsequent 'resume' load, and raise the 'key sequence' or 'duplicate key' error for any violation of the condition.

Exit

1. Normal: To compiled code or the edit module, on return to IBMDOCV, via link register.

2. Error: To IBMDRIOB or IBMDREVA

Error and Exceptional Conditions

1. KEY - duplicate key
2. KEY - null key on any statement
3. KEY - key sequence error (may be duplicate key)
4. KEY - key range not specified for insertion
5. RECORD - zero-length record variable
6. RECORD - record variable shorter than keylength + RKP
7. RECORD - long record variable
8. TRANSMIT - read error in data set
9. TRANSMIT - read error in index set
10. TRANSMIT - read error in sequence set
11. TRANSMIT - write error in data set
12. TRANSMIT - write error in index set
13. TRANSMIT - write error in sequence set
14. ERROR - outstanding operation on file
15. ERROR - event variable already active (statement with EVENT)
16. ERROR - record on non-mounted volume
17. ERROR - data set cannot be extended
18. ERROR - insufficient virtual storage to finish request
19. ERROR - too many concurrent operations on data set
20. ERROR - error in index upgrade

IBMDRVS - KSDS or PATH Input/Update/Direct Transmitter

Method (see chart DRVS)

Depending on the statement code in the request control block, the transmitter performs any necessary key checking and record checking and then sets up the appropriate macro request(s). For READ statements the GET macro is used; for WRITE and REWRITE statements the PUT macro is used; and for DELETE statements the ERASE macro is used. REWRITE and DELETE statements need not be preceded by a READ statement. In the absence of a prior READ for the same key, the transmitter will execute an implied READ KEY statement. An UNLOCK statement for an exclusive update file is treated as a NO-OP, except that the normal check for a null key is performed.

Records can be added to a non-empty data set by the WRITE statement. For the 'initial load' operation, the file must be opened for sequential output, and at least one record written before closing.

If the EVENT option is in effect, the event variable is activated for the duration of the operation.

External Modules

1. (TOVV in TCA) - Overflow routine for GET VDA

Exit

1. Normal: To compiled code or the wait module, via link register.
2. Error: To IBMDRIOB or IBMDREVA

Error and Exceptional Conditions

1. ENDFILE - end of file encountered on any READ without KEY
2. KEY - key not found on READ/DELETE KEY
3. KEY - change of embedded key on REWRITE [FROM]
4. KEY - null key on statement with KEY
5. RECORD - short record variable on READ INTO
6. RECORD - zero-length record variable on REWRITE FROM
7. RECORD - record variable shorter than keylength + RKP on REWRITE FROM
8. RECORD - long record variable on REWRITE FROM
9. TRANSMIT - read error in data set
10. TRANSMIT - read error in index set
11. TRANSMIT - read error in sequence set
12. TRANSMIT - write error in data set
13. TRANSMIT - write error in index set
14. TRANSMIT - write error in sequence set
15. ERROR - outstanding operation on file
16. ERROR - event variable already active (statement with EVENT)
17. ERROR - no prior READ for REWRITE [FROM] or DELETE without KEY
18. ERROR - record on non-mounted volume
19. ERROR - data set cannot be extended
20. ERROR - insufficient virtual storage to finish request
21. ERROR - no positioning for sequential READ
22. ERROR - too many concurrent operations on data set
23. ERROR - error in index upgrade
24. ERROR - invalid alternate index pointer
25. ERROR - maximum number of alternate index pointers exceeded

IBMDRVR - RRDS Transmitter

Method (see chart DRVR)

Depending on the statement code in the request control block, the transmitter performs any necessary key checking and record checking and then sets up the appropriate macro request(s). For READ statements the GET macro is used; for WRITE and REWRITE statements the PUT macro is used; and for DELETE statements the ERASE macro is used. REWRITE and DELETE statements need not be preceded by a READ statement. In the absence of a prior READ for the same key, the transmitter will execute an implied READ KEY statement.

If the EVENT option is in effect, the event variable is activated for the duration of the operation.

Records can be added to a non-empty data set by the WRITE statement. For the 'initial load' operation, the file must be opened for output, and at least one record written before closing.

External Modules

1. (TOVV in TCA) - Overflow routine for GET VDA
2. (TGCL in TCA) - get control routine
3. (TRCL in TCA) - release control module

Exit

1. Normal: To compiled code or the wait module via link register.

2. Error: To IBMDRIOB or IBMDREVA

Error and Exceptional Conditions

1.	KEY -	duplicate key on WRITE KEYFROM
2.	KEY -	key not found on READ/REWRITE/DELETE KEY
3.	KEY -	null key on any statement
4.	KEY -	key range not specified for insertion
5.	KEY -	key conversion
6.	RECORD -	short record variable on READ INTO
7.	RECORD -	zero-length record variable on WRITE/REWRITE FROM
8.	RECORD -	record variable shorter than keylength + RKP on WRITE/REWRITE FROM
9.	RECORD -	long record variable on WRITE/REWRITE FROM
10.	TRANSMIT -	read error in data set
11.	TRANSMIT -	read error in index set
12.	TRANSMIT -	read error in sequence set
13.	TRANSMIT -	write error in data set
14.	TRANSMIT -	write error in index set
15.	TRANSMIT -	write error in sequence set
16.	ERROR -	outstanding operation on file
17.	ERROR -	event variable already active (statement with EVENT)
18.	ERROR -	record on non-mounted volume
19.	ERROR -	data set cannot be extended
20.	ERROR -	insufficient virtual storage to finish request
21.	ERROR -	too many concurrent operations on data set
22.	ERROR -	invalid sequential WRITE

Figure 6.3 gives details of files handled by VSAM transmitters; the

key to the various abbreviations used in the figure is as follows:

MVE	- MOVE mode processing
LOC	- LOCATE mode processing
NUP	- non-update mode
UPD	- update mode
FKS	- full key search
GEN	- generic key search
KEQ	- key equal search
d.b.	- dummy buffer
r.v.	- record variable

Statements	File Type	Macro	OPTCD options (see Note 1)	Other RPL parameters
LOCATE	OUT BUF	PUT	MVE,NUP	AREA/RECLEN - d.b.
WRITE FROM	OUT BUF/UNB	PUT	MVE,NUP	AREA/RECLEN - r.v.
WRITE FROM EVENT	OUT UNB			
LOCATE KEYFROM	OUT BUF	PUT	MVE,NUP	AREA/RECLEN - d.b.
WRITE FROM KEYFROM	OUT BUF/UNB	PUT	MVE,NUP	AREA/RECLEN - r.v.
WRITE FROM KEYFROM EVENT	OUT UNB			
READ INTO [KEYTO]	IN/UPD BUF/UNB	GET	MVE,NUP/UPD	AREA/AREALEN - r.v.
READ INTO [KEYTO] EVENT	IN/UPD UNB			
READ SET [KEYTO]	IN/UPD BUF	GET	MVE,NUP/UPD	AREA/AREALEN - d.b.
READ IGNORE	IN/UPD BUF/UNB	GET	LOC,NUP	AREA/AREALEN- LOC ptr
READ IGNORE EVENT	IN/UPD UNB			
READ INTO KEY	IN/UPD BUF/UNB	[POINT	FKS/GEN,KEQ]	(1) (KEYLEN if GEN)
		GET	MVE,NUP/UPD	AREA/AREALEN - r.v.
READ INTO KEY EVENT	IN/UPD UNB	[POINT	FKS/GEN,KEQ]	(1) (KEYLEN if GEN)
		GET	MVE,NUP/UPD	AREA/AREALEN - r.v.
READ SET KEY	IN/UPD BUF	[POINT	FKS/GEN,KEQ]	(1) (KEYLEN if GEN)
		GET	MVE,NUP/UPD	AREA/AREALEN - d.b.

Figure 6.3. (part 1 of 2) Relationships between PL/I Files and VSAM Transmitters

Statements	File Type	Macro	OPTCD options (see Note 1)	Other RPL parameters
REWRITE	UPD BUF	PUT	MVE,UPD	(AREA/RECLEN - d.b. set)
REWRITE FROM REWRITE FROM EVENT	UPD BUF/UNB UPD UNB	PUT	MVE,UPD	AREA/RECLEN - r.v.
DELETE DELETE EVENT	UPD BUF/UNB UPD UNB	ERASE	MVE,UPD	
REWRITE FROM KEY (2) REWRITE FROM KEY EVENT (2)	UPD BUF/UNB UPD UNB	PUT	MVE,UPD	AREA/RECLEN - r.v.
DELETE KEY (2) DELETE KEY EVENT (2)	UPD BUF/UNB UPD UNB	ERASE	MVE,UPD	

Note 1: POINT and GET SEQ are used if the file is SEQUENTIAL. GET DIR is used if the file is DIRECT.

Note 2: If there was no prior READ for the same key then an implied READ KEY is done.

Figure 6.3. (Part 2 of 2) Relationships between PL/I Files and VSAM Transmitters

The following explanatory notes should be read in conjunction both with Figure 6.3. and the relevant transmitter descriptions given earlier in this chapter.

IBMDRVZ, IBMDRVT, IBMDRVS, and IBMDRVR

1. All macro requests are synchronous.

IBMDRVZ, IBMDRVS, and IBMDRVR

1. For READ INTO and READ SET, the OPTCD option will be NUP or UPD depending on whether the file is input or update.

IBMDRVT, IBMDRVR

1. The KEYFROM option and the KEY option overwrite the embedded key in the record variable.

IBMDRVZ

1. On update the length of a record must not change (a VSAM restriction for ESDS). Thus a REWRITE must use the RPL RECLEN value set by the GET request for the prior READ. The normal record checking rules for a REWRITE will apply.

IBMDRVS

1. The POINT macro will use a full key search (OPTCD = FKS), unless GENKEY was specified in the ENVIRONMENT option.

2. The KEYTO option extracts the embedded key in the record variable.

IBMDREF - Endfile Module

Function

To act as an interface between all input transmitters and the resident error handler, when raising ENDFILE.

Method (chart DREF)

This module is called by the transmitter whenever endfile is being raised without any other error having occurred previously.

1. If condition being raised is not ENDFILE, branch to the error-module loading routine in IBMDRIO.

2. If the EVENT option has been specified:

   (a) Indicate that this event variable raised an error.

   (b) Set abnormal status.

3. Set up endfile parameter list.

4. Reset flags in FCB.

5. Set up a null onkey.

6. Set oncount = 1.

7. Branch to error handler.

8. Return to compiled code or to the wait module.

Linkage

```
R2  =  A(FCB)
R4  =  A(DTF)
R5  =  A(RCB)
R7  =  A(KD) or A(onkey) or zero
R8  =  A(EV) or zero
DR  =  A(DSA of IBMDRIO)
```

Note: The registers are saved on entry to IBMDRIO.

```
PLIST = A(FCB)
        A(RCB)
        A(RD)
        A(KD) or zero
        A(EV) or zero
        Zero
```

Calls

Error-module loading routine in IBMDRIO.
IBMDERR - Resident error-handler.

Called By

All input transmitters.

IBMDREV - Error Module for VSAM Files

Function

To act as an interface between transmitters and the resident error handler, for VSAM files.

Method (chart DREV)

The module is called by the transmitter whenever an error or exceptional condition is raised. It is also called for ENDFILE if another condition has been raised previously.

1. Get workspace.

2. Determine type of error, using the error code in the IOCB.

3. Insert the oncode into the parameter list.

4. Set the onkey field (if file has KEYED attribute). If the "key not found" error occurred during sequential processing, reposition file to next higher key. If no more records exist, then reposition to end of file.

5. Reset flags in IOCB.

6. Set ONCOUNT.

7. Branch to error handler.

On return from error handler:

8. If interrupt was a multiple one and not all errors have been dealt with, repeat steps 6 to 7 for next error.

9. If PL/I statement causing the error was a LOCATE, return to compiled code at address held in abnormal locate return entry in plist. Otherwise, execute normal return.

Linkage

R2 = A(IOCB)
R4 = A(DTF)
R5 = A(RCB)
R7 = A(KD) or zero
R8 = A(EV) or zero
DR = A(DSA of IBMDRIO)

OFR1 in DSA of IBMDRIO points to compiled code PLIST

```
PLIST = A(IOCB)
        A(RCB)
        A(RD) or A(ignore factor) or
           A(buffer address for READ SET statements)
        A(KD) or zero
        A(abnormal locate return) or zero
```

Calls

IBMDERR - Resident error-handling module.

Called By

IBMDRVZ - ESDS transmitter.
IBMDRVT - KSDS Sequential output transmitter.
|IBMDRVS - KSDS and PATH input/update/direct transmitter.
|IBMDRVR - RRDS transmitter.

IBMDREX - Error Module for INDEXED SEQUENTIAL Files

Function

To act as an interface between transmitters and the resident error handler, for INDEXED files.

Method (chart DREX)

The module is called by the transmitter whenever an error or exceptional condition is raised. It is also called for ENDFILE if another condition has been raised previously.

1. Get workspace.

2. Determine type of error, using the error code in the FCB.

3. Insert the oncode into the parameter list.

4. Set the onkey field (if file has KEYED attribute). If the "key not found" error occurred during sequential processing, reposition file to next higher key. If no more records exist, then repcsiticn to end of file.

5. Reset flags in FCB.

6. Set ONCOUNT

7. Branch to error handler.

On return from error handler:

8. If interrupt was a multiple one and not all errors have been dealt with, repeat steps 6 to 7 for next error.

9. If PL/I statement causing the error was a LOCATE, return to compiled code at address held in abnormal locate return entry in plist. Otherwise, execute normal return.

Linkage

```
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R7 = A(KD) or zero
R8 = A(EV) or zero
DR = A(DSA of IBMDRIO)
```

OFR1 in DSA of IBMDRI points to compiled code PLIST

```
PLIST = A(FCB)
        A(RCB)
        A(RD) or A(ignore factor) cr
           A(buffer address for READ SET statements)
        A(KD) or zero
        A(abnormal locate return) or zero
```

Calls

IBMDERR - Resident error-handling module.

Called By

IBMDRLZ - Sequential indexed transmitter.
IBMDRJZ - Sequential indexed transmitter.
IBMDRKZ - Direct indexed transmitter.

IBMDREY - Error Module for REGIONAL and UNBUFFERED CONSECUTIVE Files

Function

To act as an interface between transmitters for REGIONAL and UNBUFFERED CONSECUTIVE files and the resident error handler.

Method (chart DREY)

The module is called by the transmitter whenever an error or exceptional condition is raised. It is also called for ENDFILE if another condition has been raised previously.

1. Obtain workspace.

2. Determine type of error, using the error code in the FCB.

3. Insert the error code into the parameter list.

4. Reset flags in FCB.

5. If the EVENT option has been specified:

   a. Indicate that this event variable has raised an error.

   b. Set the abnormal status.

6. Set the ONKEY field (if file has KEYED attribute).

7. Set the oncount field.

8. Branch to error handler.

9. If more than one error repeat from step 7 above.

10. If PL/I statement causing the error was a LOCATE, return to code at the address given as the abnormal locate return in the parameter list. Otherwise, execute normal return to compiled code or return to wait module.

Linkage

```
R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
R7 = A(KD) or zero or A(onkey)
R8 = A(EV) or zero
DR = A(DSA of IBMDRIO)
```

OFR1 in DSA of IBMDRIO points to compiled code PLIST

```
PLIST = A(FCB)
        A(RCB)
        A(RD) or A(ignore factor) or
          A(buffer address for READ SET statements) or
          A(address of buffer pointer for LOCATE)
        A(KD) or zero
        A(EV) or zero
        A(abnormal locate return) or zero
```

Calls

IBMDERR - Resident error-handling module.

Called By

All REGIONAL and CONSECUTIVE UNBUFFERED transmitters.

IBMDREZ - Error Module for BUFFERED, CONSECUTIVE Files

Function

To act as an interface between transmitters for BUFFERED CONSECUTIVE files and the resident error handler.

Method (chart DREZ)

The module is called by the transmitter whenever an error or exceptional condition is raised, except when ENDFILE is raised without any other error having been raised previously. (Such a condition is handled by the end-of-file module).

1. Get Workspace

2. Determine type of error, using the FCB.

3. Insert the oncode into the parameter list.

4. Reset flags in FCB and TCA.

5. Set ONCOUNT.

6. Branch to error handler.

On return from error handler:

7. If interrupt was a multiple one and not all errors have been dealt with, repeat steps 5 and 6 for next error.

8. If PL/I statement causing the error was a LOCATE, return to compiled code at the address given as the abnormal locate return in the parameter list. Otherwise, execute normal return to compiled code or return to wait module.

Linkage

R2 = A(FCB)
R4 = A(DTF)
R5 = A(RCB)
DR = A(DSA of IBMDRIO)

OFR1 in DSA of IBMDRIO points to compiled code PLIST

```
PLIST = A(FBC)
        A(RCB)
        A(RD) or A(ignore factor) or
          A(buffer address for READ SET statements)
        zero
        zero
        A(abnormal locate return) or zero
```

Calls

IBMDERR - Resident error-handling module.

Called By

Consecutive buffered transmitters.
IBMDRQX - U format transmitter.
IBMDRQY - V format transmitter.
IBMDRQZ - F format transmitter.

|The transient library contains five STREAM I/O transmitters and one CICS |transmitter. The function of the transmitters is to move records between the buffers and data sets associated with PL/I files. A list of PL/I file types with their associated stream I/O transmitters is given in figure 7.1. The transmitters for F-format PRINT files and for STREAM INPUT files are in the DOS PL/I resident library.

FILE ATTRIBUTES	TRANSMITTER
OUTPUT, F-format	IBMDSOF
OUTPUT, U-format	IBMDSOU
OUTPUT, V-format	IBMDSOV
PRINT, U-format	IBMDSTU
PRINT, V-format	IBMDSTV
| OUTPUT, CICS	IBMFSTV

Figure 7.1 Stream I/O Transmitter Modules

LOADING STREAM I/O TRANSMITTERS

Each stream I/O transmitter has a 16-byte prefix containing the following information:

Bytes 0 to 3	Length of module
Bytes 4 to 7	Chain back field
Bytes 8 to 10	5th, 6th, and 7th letters of module name
Byte 11	Responsibility count
Bytes 12 to 15	Not used

The start of the transmitter chain is addressed from field TLMC in the task communications area. The chain includes all stream I/O transmitters, record I/O transmitters, and record I/O error modules that are currently in store.

When a PL/I STREAM file is being opened, the associated first level open module scans the transmitter chain, searching for the required transmitter. If the transmitter is found, its responsibility count is incremented by one; otherwise the module is loaded and added to the chain. Similarly, when a PL/I file is being closed, the close module IBMDOCA scans the chain to find the associated transmitter and reduces its responsibility count by one. If the responsibility count is thus reduced to zero, the transmitter is deleted from the chain and the storage containing it is freed.

LINKAGE FOR STREAM I/O TRANSMITTERS

All the stream I/O transmitters are passed the address of the relevant file control block (FCB) in register R1.

MODULE DESCRIPTIONS

IBMDSOF - OUTPUT F-format Transmitter

Function

To write one F-format record from an output buffer onto a data set accessed by a STREAM OUTPUT file.

Method

A flowchart of the module is given in chart DSOF.

Error and Exceptional Conditions

The TRANSMIT condition or the uncorrectable error in output error (error message IBM144I) may be raised by this module.

Linkage

R1 = A(file control block)

Calls

IBMDERR - Resident error-handler

Called By

IBMDSDO - Data-directed output.
IBMDSEO - Edit-directed output.
IBMDSLO - List-directed output.
IBMDSPL - PAGE, LINE, and SKIP formats and options module.
IBMDSXC - X and COLUMN formats module.
IBMDSED - Edit-directed Input/Output.
IBMDSCP - COPY module.

IBMDSOU - OUTPUT U-format Transmitter

Function

To write one U-format record from an output buffer onto a data set accessed by a STREAM OUTPUT file.

Method

A flowchart of the module is given in chart DSOU.

Error and Exceptional Conditions

The TRANSMIT condition or the uncorrectable error in output error (error message IBM144I) may be raised by this module.

Linkage

R1 = A(file control block)

Calls

IBMDERR - Resident error-handler

Called By

IBMDSDO - Data-directed output.
IBMDSEO - Edit-directed output.
IBMDSLO - List-directed output.
IBMDSPL - PAGE, LINE, and SKIP formats and options module.
IBMDSXC - X and COLUMN formats module.
IBMDSED - Edit-directed Input/Output.
IBMDSCP - COPY module.

IBMDSOV - OUTPUT V-format Transmitter

Function

To write one V-format record from an output buffer onto a data set accessed by a STREAM OUTPUT file.

Method

A flowchart of the module is given in chart DSOV.

Error and Exceptional Conditions

The TRANSMIT condition or the uncorrectable error in output error (error message IBM144I) may be raised by this module.

Linkage

R1 = A(file control block)

Calls

IBMDERR - Resident error-handler

Called By

IBMDSDO - Data-directed output.
IBMDSEO - Edit-directed output.
IBMDSLO - List-directed output.
IBMDSPL - PAGE, LINE, and SKIP formats and options module.
IBMDSXC - X and COLUMN formats module.
IBMDSED - Edit-directed Input/Output.
IBMDSCP - COPY module.

IBMDSTU - PRINT U-format Transmitter

Function

To write one U-format record from an output buffer onto a data set accessed by a STREAM PRINT file. The module has two entry points:

IBMDSTUA: Normal output.

IBMDSTUB: Output of error messages when transmitter is being used for SYSPRINT.

Method

A flowchart of the module is given in chart DSTU.

Error and Exceptional Conditions

The TRANSMIT condition or the uncorrectable error in output errcr (error message IBM144I) may be raised by this mcdule.

The ENDPAGE condition is raised if the line number rises abcve the pagesize.

Linkage

Entry Point IBMDSTUA:

R1 = A(file control block)

Entry Point IBMDSTUB:

R1 = A(string locator for message)
R2 = A(request control flag byte)

Calls

IBMDERR - Resident error-handler.

Called By

IBMDSDO - Data-directed output.
IBMDSEO - Edit-directed output.
IBMDSLO - List-directed output.
IBMDSPL - PAGE, LINE, and SKIP formats and options module.
IBMDSXC - X and COLUMN formats mcdule.
IBMDSED - Edit-directed Input/Output.
IBMDSCP - COPY module.
IBMDESN - Error message module.
IBMDPEP - Housekeeping diagnostic message module.
IBMDPES - Storage management diagnostic message module.

IBMDSTV - PRINT V-format Transmitter

Function

To write one V-format record from an output buffer onto a data set accessed by a STREAM PRINT file. The module has two entry points:

IBMDSTVA: Normal output.

IBMDSTVB: Output of error messages when transmitter is being used for SYSPRINT.

Method

A flowchart of the module is given in chart DSTV.

Error and Exceptional Conditions

The TRANSMIT condition or the uncorrectable error in output error (error message IBM144I) may be raised by this module. The ENDPAGE condition is raised if the line number rises above the pagesize.

Linkage

Entry Point IBMDSTVA:

R1 = A(file control block)

Entry Point IBMDSTVB:

R1 = A(string locator for message)
R2 = A(request control flag byte)

Calls

IBMDERR - Resident error-handler.

Called By

IBMDSDO - Data-directed output.
IBMDSEO - Edit-directed output.
IBMDSLO - List-directed output.
IBMDSPL - PAGE, LINE, and SKIP formats and options module.
IBMDSXC - X and COLUMN formats module.
IBMDSED - Edit-directed Input/Output.
IBMDSCP - COPY module.
IBMDESN - Error message module.
IBMDPEP - Housekeeping diagnostic message module.
IBMDPES - Storage management diagnostic message module.

IBMBSTA - Tab Table

Function

This module is a table containing the default pagesize and linesize values and the default tab positions for list directed and data-directed output on PRINT files.

The module does not contain any executable code.

IBMFSTV - CICS Stream Output Routine

Function

This module is the output transmitter for CICS systems. It is used for SYSPRINT, REPORT, COUNT, and messages. In addition to the transmitter it also incorporates OPEN and CLOSE functions for SYSPRINT on CICS.

Method (chart FSTV)

The module is loaded by the bootstrap in IBMFPCCA. For open it builds the various control blocks required and then transmits records to the transient data queue CPLI, having first placed heading information on each record.

Linkage

R15 = Points at required entry point
R14 = Return point
R1 = 'Usual' parameter list for function

Entry points

IBMFSTVA	Stream output transmitter
IBMFSTVB	Message transmitter
IBMFSTVC	Count/Report transmitter
IBMBOCLA	Explicit open
IBMBOCLB	Implicit open
IBMBOCLC	Explicit close

External Modules

DFHTD - In CICS

Error Condition

If any errors occur in the transmission an abend is issued using a code of APLI.

CHAPTER 8: CONVERSICN ROUTINES

The transient litrary contains four conversion routines: IBMBCCL, IBMBCCR, IBMDCCR, and IBMDSCT. Module IBMBCCR is included to ensure compatibility with earlier versicns of the DOS PL/I Resident Litrary. The remainder of the conversion routines are in the DOS PL/I resident library and are described in DOS PL/I Resident Litrary: Program Logic.

MODULE DESCRIPTIONS

IBMBCCL - Conversion Director (Complex Strings)

Function

To direct complex string conversions.

Method

The source is first inspected to determine the implied arithmetic data type (assumed to be complex). The implied data types are then combined to give the dominant one, which then becomes the intermediate target to which the source is converted. The intermediate target is then converted to the final target.

For a character string source, the string is scanned, noting decimal point, exponent, and binary indicator if any. A DED (data element descriptor) is then derived from this information.

If the source is p-format, the intermediate DED takes scale and precision from the picture DED and sets the data type to fixed or float decimal, according to the picture type. For exponential (e) or fixed (f) formats, the information for the DED can be derived from the FED (format element descriptor). However the precision and scale from the latter two formats can be overridden if a decimal point is found in the input stream. Therefore such a string must be scanned.

A bit string source has an intermediate form of float binary.

Linkage

```
R1    = A(PLIST)
PLIST = A(source or source locator)
        A(source DED)
        A(target or target locator)
        A(target DED)
R4    = A(list of conversion package entry points)
```

Calls

Conversion modules in the DOS PL/I resident library.

Called By

IBMDCCS - String conversion director bootstrap

IBMBCCR - Conversion Director (Non-complex Strings)

Function

To direct non-complex string conversions

Details of this module are as given for IBMDCCR below.

IBMDCCR - Conversion Director (Non-complex Strings)

Function

To direct non-complex string conversions.

Method

The source is first converted to the data type implied by its attributes and then to the target type.

For a character string source, the string is scanned, noting decimal point, exponent, and binary indicator if any. A DED is then derived from this information.

If the source is P-format, the intermediate DED takes scale and precision from the picture DED and sets the data type to fixed or float decimal, according to the picture type. For E or F formats, the information for the DED can be derived from the FED. However the precision and scale from the latter two formats can be overridden if a decimal point is found in the input stream. Therefore such a string must be scanned.

A bit string source has an intermediate form of float binary.

Linkage

```
R1    = A(PLIST)
PLIST = A(source or source locator)
        A(source DED)
        A(target or target locator)
        A(target DED)
R4    = A(list of conversion package entry points)
```

Error and Exceptional Conditions

CONVERSION is raised if the string is a complex arithmetic constant.

Calls

Conversion modules in the DOS PL/I resident library.

Called By

IBMDCCS - String conversion director bootstrap

IBMDSCT - Conversion Condition Inteıface

Function

To preserve an item in an input buffer and to change the SIOCB when CONVERSION is raised so that, if the file is re-used in an on-unit, the ONSOURCE string will not be lost.

Method

A flowchart of the module is given in chart DSCT.

Linkage

The module retrieves information frcm the ONCA and from the DSA chain which defines the path from compiled code to the conversion package.

Calls

IBMDERR - Error handler.

Called By

IBMDSCV - Conversion Fix-up bootstrap (resident).

APPENDIX A: LIST OF MODULES

Name	Function	Size (approx)
IBMBCCL	Conversion director (complex strings)	1830 bytes
IBMBCCR	Conversion director (non-complex strings)	940 bytes
IBMBEOC	On-code translate	230 bytes
IBMBETA	Miscellaneous non-ON messages (1)	710 bytes
IBMBETB	Miscellaneous non-ON messages (2)	1140 bytes
IBMBETC	Misc. and computational non-ON messages	1000 bytes
IBMBETI	I/O non-ON messages	1340 bytes
IBMBETO	ON messages (1)	1380 bytes
IBMBETP	ON messages (2)	760 bytes
IBMBETQ	ON messages (3)	1020 bytes
IBMBETT	EVENT messages	1140 bytes
IBMBSTA	Tab table	40 bytes
IBMDCCR	Conversion director (non-complex strings)	910 bytes
IBMDEDO	Open diagnostic file	210 bytes
IBMDEDW	Console transmitter	190 bytes
IBMDESM	Error message module phase 1	1010 bytes
IBMDESN	Error message module phase 2	2250 bytes
IBMDESY	Error system action	180 bytes
IBMDKDD	Hexadecimal dump	1360 bytes
IBMDKDR	Report Option Module	1100 bytes
IBMDKDT	Dump File Transmitter	870 bytes
IBMDKFA	Dump file atrributes	2470 bytes
IBMDKMR	Dump control	2110 bytes
IBMDKPT	Dump parameter translate	310 bytes
IBMDKTB	Save Area Control Block printout	2330 bytes
IBMDKTC	Save Area chain validity checker	120 bytes
IBMDKTR	Dump trace	3360 bytes
IBMDOCA	Close	1740 bytes
IBMDOCV	Close (VSAM)	460 bytes
IBMDOPM	OPEN - consecutive unbuffered files	1250 bytes
IBMDOPP	OPEN - consecutive buffered files	1000 bytes
IBMDOPQ	OPEN - consecutive buffered files (level 2)	870 bytes
IBMDOPS	OPEN - stream files	1030 bytes
IBMDOPT	OPEN - stream files (level 2)	1080 bytes
IBMDOPU	OPEN - consecutive buffered/stream files (level 3)	780 bytes
IBMDOPW	OPEN - indexed files (level 4)	740 bytes
IBMDOPX	OPEN - regional and indexed files	1130 bytes
IBMDOPY	OPEN - regional/indexed files (level 2)	760 bytes
IBMDOPV	Open (VSAM)	510 bytes
IBMDOPZ	OPEN - regional indexed files (level 3)	550 bytes
IBMDPEP	Housekeeping Diagnostic message module	780 bytes
IBMDPES	Storage Management Diagnostic message module	1580 bytes
IBMDPIF	Operation Exception checking, (no floating-point hardware)	130 bytes
IBMDPII	Program ISA initialization	910 bytes
IBMDPJI	Program ISA initialization from caller	740 bytes
IBMDRAW	Regional(3) sequential unbuffered output transmitter	710 bytes
IBMDRAX	Regional(3) sequential buffered output transmitter	650 bytes
IBMDRAY	Regional(1) sequential unbuffered output transmitter	770 bytes
IBMDRAZ	Regional(1) sequential buffered output transmitter	680 bytes
IBMDRBW	Regional(1) sequential buffered input/update transmitter	860 bytes
IBMDRBX	Regional(3) sequential buffered input/update transmitter	860 bytes
IBMDRBY	Regional(3) sequential unbuffered input/update transmitter	1040 bytes
IBMDRBZ	Regional(1) sequential unbuffered input/update transmitter	1020 bytes
IBMDRCY	Consecutive sequential unbuffered transmitter, U-format	1020 bytes
IBMDRCZ	Consecutive sequential unbuffered transmitter, F-format	1030 bytes

IBMDRDY	Regional(3) direct transmitter	950	bytes
IBMDRDZ	Regional(1) direct transmitter	850	bytes
IBMDREF	ENDFILE module	170	bytes
IBMDREV	Error Module for VSAM files	730	bytes
IBMDREX	Error handler for indexed files	580	bytes
IBMDREY	Error handler for regional and unbuffered consecutive files	690	bytes
IBMDREZ	Error handler for buffered consecutive files	490	bytes
IBMDRJZ	Indexed sequential input/update transmitter	1410	bytes
IBMDRKZ	Indexed direct input/update transmitter	960	bytes
IBMDRLZ	Indexed sequential output transmitter	660	bytes
IBMDRRR	Consecutive buffered exit module	300	bytes
IBMDRRT	Consecutive sequential buffered OMR transmitter, F-format	400	bytes
IBMDRRU	Consecutive sequential buffered associate files, U-format	360	bytes
IBMDRRV	Consecutive sequential buffered associate files, V-format	380	bytes
IBMDRRW	Consecutive sequential buffered associate files, F-format	620	bytes
IBMDRRX	Consecutive sequential buffered transmitter, U-format	520	bytes
IBMDRRY	Consecutive sequential buffered transmitter, V-format	620	bytes
IBMDRRZ	Consecutive sequential buffered transmitter, F-format	620	bytes
IBMDRVR	KSDS Direct Transmitter	1380	bytes
IBMDRVS	KSDS Sequential Input/Update Transmitter	1820	bytes
IBMDRVT	KSDS Sequential Output Transmitter	970	bytes
IBMDRVZ	ESDS Transmitter	1470	bytes
IBMDSCT	Conversion condition interface	470	bytes
IBMDSOF	Stream output transmitter, F-format	210	bytes
IBMDSOU	Stream output transmitter, U-format	170	bytes
IBMDSOV	Stream output transmitter, V-format	210	bytes
IBMDSTU	Stream print transmitter, U-format	410	bytes
IBMDSTV	Stream print transmitter, V-format	430	bytes
IBMFEFC	Print count tables	100	bytes
IBMFERR	Error handler	100	bytes
IBMFESM	Error message routine 1	100	bytes
IBMFESN	Error message routine 2	100	bytes
IBMFKCS	Print CICS control blocks	1220	bytes
IBMFKMR	Invoke dump modules and transmit output	1900	bytes
IBMFKPT	Convert options parameter into flags	460	bytes
IBMFKTB	Print PL/I blocks for PLIDUMP under CICS	3100	bytes
IBMFKTC	Detect errors in DSA chain	350	bytes
IBMFKTR	Print trace part of PLIDUMP under CICS	4550	bytes
IBMFPCC	CICS nucleus director	100	bytes
IBMFPGD	Get storage with report	100	bytes
IBMFPGR	Get storage	100	bytes
IBMFPIR	Initialization/termination routine	100	bytes
IBMFPMR	Print report table	100	bytes
IBMFSTV	Stream output transmitter	100	bytes

APPENDIX B: LIST OF LIBRARY MACRO INSTRUCTIONS

Name Function

IBMBXCH Chains a specified element to a specified element.

|IBMBXCIC Describes CICS appendage

IBMBXDBG Debugging macro instruction (see appendix C).

IBMBXDBL Debugging macro instruction (see appendix C).

IBMBXDBM Debugging macro instruction (see appendix C).

IBMBXDC Dechains a specified element.

IBMBXDD Provides a DSECT map of a data element descriptor (DED).

IBMDXDM Provides a description of the options for the dump modules and a description of the dump control routine's Dynamic Storage Area (DSA).

IBMBXDP Provides a description of the picture part of the Data Element Descriptor (DED).

IBMBXEC Gives the code required by routines which raise CONVERSION or set up information for data conversion conditions.

IBMBXER Provides DSECT maps of the following blocks:

Dynamic Storage Area (DSA).
Dynamic On Control Block (ONCB).
Static On Control Block (ONCB).
Diagnostic File Block (DFB).
Dump Block (DUB).
Dynamic Storage Area (DSA) for IBMDERR.
Interrupt Control Block.

IBMBXET Generates the message text modules.

IBMBXEV Provides a DSECT map of an Event Variable (EV).

IBMBXFLT Provides DSECT maps of the headings for the flow statement table and for the meanings of the bits in the flag byte of each statement number entry.

IBMBXFV Frees a Variable Data Area (VDA).

IBMDXGC Generates GOTO code that is copied into the TCA by IBMDPII.

IBMBXGV Gets a Variable Data Area (VDA) and returns its address.

IBMBXIC Initializes two adjacent chain fields to zero for use by macros IBMBXCH and IBMBXDC.

IBMBXIN Performs the initialization functions required by a library module.

IBMBXIOS Provides a DSECT map of the Request Control Block and the parameter list for record I/O statements.

IBMBXKY Checks the key given in the KEYFROM or KEY option for regional files and converts it into the "search address" for the record. For Regional(3) files the macro also moves the key to the buffer area.

IBMBXLB Defines all symbolic register and offset names and provides DSECT maps of the Task Communications Area (TCA), library workspace (LWS), and the On Communications Area (ONCA).

IBMBXML Moves any number of bytes from one area to another.

IBMBXPL Describes PLISTART parameter list.

IBMBXPLC Describes initialization plist for CICS

IBMBXRFC Performs updating of region number for Regional files.

IBMBXRKM Performs KEYTO processing for Regional files or moves key from KEYFROM variable to buffer area.

IBMBXRRI Provides a record checking table for reference in library modules.

IBMBXRRT Generates code to set up the registers for record ckecking or generates an offset table for use in record ckecking.

IBMBXRT Provides the code necessary to return from a library module to a caller.

IBMBXRWS Defines the workspace of all record I/O transmitters to enable them to pass information to the record I/O error modules.

IBMBXSIO Provides a DSECT map of the Stream I/O Control Block (SIOCB).

IBMBXSY Provides a DSECT map of the Symbol Table.

IBMBXTAB Provides a DSECT map of the tab table.

IBMBXVKD Provides a DSECT map of the Key Descriptor.

IBMBXVRD Provides a DSECT map of the Record Descriptor.

IBMBXWT Provides a DSECT map of an EVTAB element.

IBMDXCOM Provides a DSECT map of the DOS communications region.

IBMDXDGT Debugging macro instruction (see appendix C).

IBMDXDTF Provides a DSECT map of the Define The File Block (DTF) for indexed files.

IBMDXFCB Provides a DSECT map of the File Control Block (FCB).

IBMDXNVB Provides a DSECT map of the DOS Environment Block.

IBMDXOSA Defines a DSECT for Open/Close workspace.

IBMDXTA Provides a DSECT map of the implementation defined appendage of the Task Communications Area (TCA).

IBMFXDM Provides a description of the options for the CICS dump modules and a description of the dump control's Dynamic Storage Area (DSA).

APPENDIX C: REACTIVATION OF DEBUGGING MACRO INSTRUCTIONS

The program listings of many library modules contain debugging macro instructions which were used during the development of the library. The release version of the library does not contain any instructions corresponding to these macro instructions; the debugging facilities that they provide are therefore normally inactive. This appendix describes how the debugging macro instructions can be reactivated.

Debugging Facilities

The facilities provided by the debugging macro instructions are as follows:

1. Module trace: This facility is provided by macro instruction IBMBXDBM, which is included once in every library module. When the instruction is excuted it causes the 5th, 6th and 7th letters of the module name to be entered in a push-down stack, thereby providing confirmation that the module has been entered. The push-down stack holds the names of the last twelve modules entered.

2. Label Trace and General Register Dump: This facility is provided by macro instruction IBMBXDBG. When this macro instruction is expanded, it generates a label of the form DBXYZnn, where X, Y, and Z are the 5th, 6th, and 7th characters of the module name, and nn is a numeric value unique to the particular appearance of the macro instruction. When the macro instruction is executed, it stores this label in a trace table. Optionally, the contents of specified registers can be stored in the trace table following the generated label.

3. Bit Setting: This facility is provided by macro instruction IBMBXDBG as an alternative to the label trace facility. When the macro instruction is executed, it sets a bit in a known position in a bit table, thereby providing confirmation that control has passed through the section of code containing the macro instruction.

The debugging information generated by these macro instructions is stored in a table known as BUGTAB. Storage for this table is obtained at program initialization by macro instruction IBMDXDGT.

The debugging facilities are controlled by operands on a "debugging level" macro instruction: IBMBXDBL. This macro instruction appears at least once in every module that contains other debugging macro instructions. It has the following format:

```
                |NIL|,
IBMBXDBL  LEVEL=|BIT|, REG1=n, REG2=m, TYPE=CHANGE, PHASE=|DEV    |
                |LAB|,                                    |RELEASE|
                |REG|
```

The operands are optional and can be coded in any order. The default values are: LEVEL=BIT, REG1=13, REG2=11, PHASE=RELEASE.

LEVEL=NIL causes macro instruction IBMBXDBG to generate no executable instructions.

LEVEL=BIT selects the bit-setting facility provided by macrc instruction IBMBXDBG.

LEVEL=LAB selects the label trace facility provided by macrc instruction IBMBXDBG.

LEVEL=REG selects the label trace facility with the general register dump option. Registers REG1 through REG2 are dumped.

If an IBMBXDBL macro instruction appears more than once in a library module, TYPE=CHANGE is coded on the second and subsequent appearances.

PHASE=RELEASE deactivates all the debugging facilities. This option overrides all other options.

Reactivating the debugging macro instructions

In order to reactivate the library debugging facilities you will require a source module for each module that you wish to reactivate, scurce modules for modules IBMDPIR (PL/I resident library) and IBMDPII (PL/I transient library), and a library macro library.

The source modules must be modified as follows:

1. Locate the IBMBXBDL macro instructions in modules IBMDPIR and IBMDPII and change the PHASE cperand to PHASE=DEV. This is necessary to obtain storage for the BUGTAB.

2. Locate the IBMBXDBL macro instructions in the modules whose debugging facilities you wish to reactivate, change the PHASE operand to PHASE=DEV, and code the LEVEL operand for the required debugging facility.

The modified modules must now be reassembled and link-edited cnto the appropriate system library. The fcllowing methods are suggested.

For modules of the PL/I resident library, reassemble the module and catalog it onto the relocatable library with a new, unique, name. Specify the new name on an INCLUDE statement in the link edit step of your PL/I test program; since the entry point names of the modified module are unchanged, references to them will be resolved to the modified module rather than to the unmodified module.

For modules of the PL/I transient library, the name of the modified module cannot be changed, since the modules are loaded by name. It is therefore necessary to rename the unmodified version of the module on the core image library, and then tc link edit the modified version onto the core image library.

The modules can now be executed by means of a PL/I program.

Recovering the debugging informaticn

The information in the BUGTAB is recoverable only in a dump. Your test program must therefore produce a dump at the required point in its execution.

The BUGTAB is located immediately after the program management area, and its address is located in field TBUG in the TCA. Field TBUG is at offset 3C (hexadecimal) from the start of the TCA.

Module trace information is located at the start of the BUGTAB. Each module name in the trace table is prefixed by a number between 1 and 12. Modules 1 through 6 are at the start of the BUGTAB; modules 7 through 12 are immediately before the start of the BUGTAB. Module 1 is always the most recently entered module.

If LEVEL=LAB or LEVEL=REG has been coded on an IBMBXDBG macro instruction, the label trace will appear in the BUGTAB as a series of labels of the form DBXYZnn, where X, Y, and Z are the 5th, 6th, and 7th characters of the module name, and nn is a numeric value unique to a particular appearance of an IBMBXDBG macro instruction. The macro that has generated the particular label can be found by examining the program listing of the reassembled module.

If LEVEL=REG has been coded, each label in the BUGTAB is followed by a dump of the contents of the registers specified in the REG1 and REG2 operands.

If LEVEL=BIT has been coded, then each IBMBXDBG macro instruction will set a specific bit in the BUGTAB when it is executed. The particular bit set by IBMBXDBG is referenced by a note in the macro expansion. A typical note appears as:

```
SET BIT 2 IN BYTE 0 OF FIELD BZJDS OFFSET HEX 136 FROM BIT-TAB
```

The bit table (BIT-TAB) starts at the first fullword boundary after character string 'BIT TAB' in the BUGTAB.

Part 2: Flowcharts

This part contains flowcharts of all the DOS transient library modules that contain executable instructions. Chart references are formed from the last four letters of the module name; for example, the flowchart for module IBMDKTB is on chart DKTB. The charts are arranged in alphabetical order.

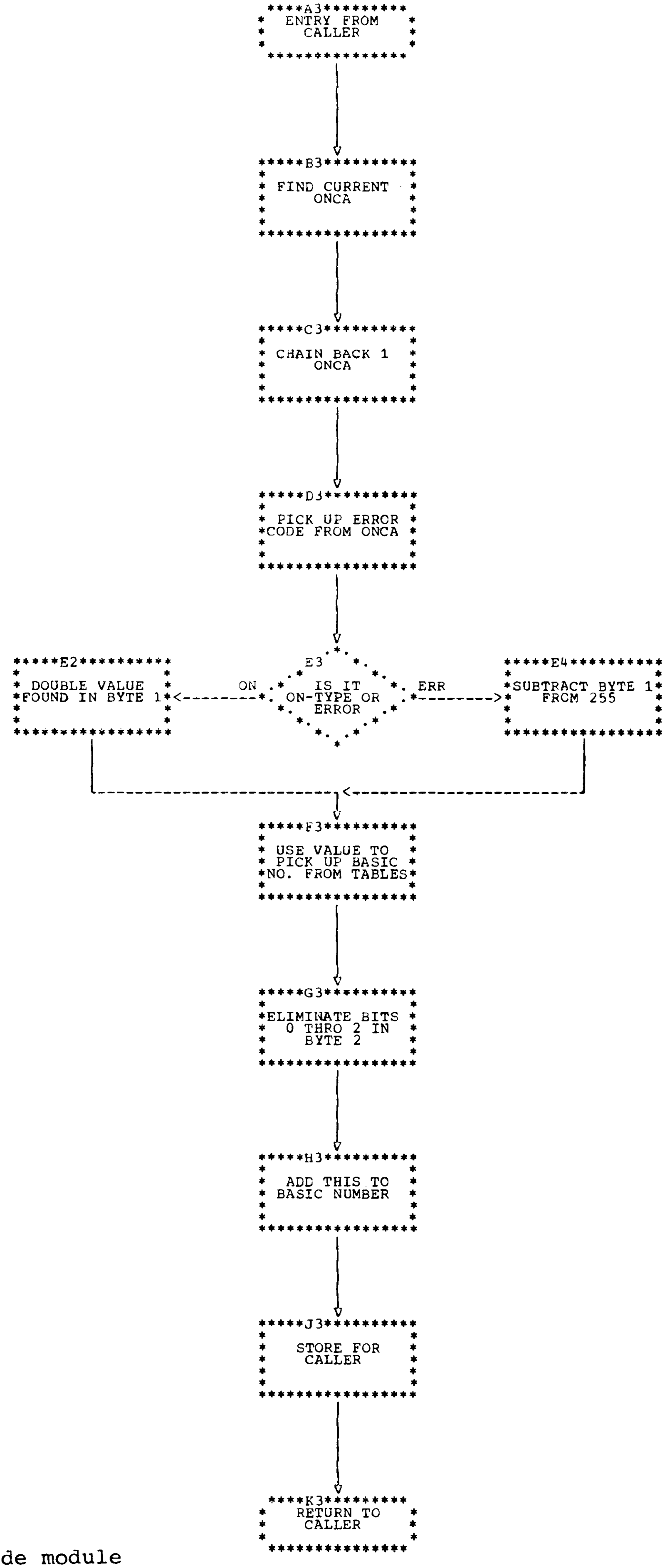


Chart BEOC. On-code module

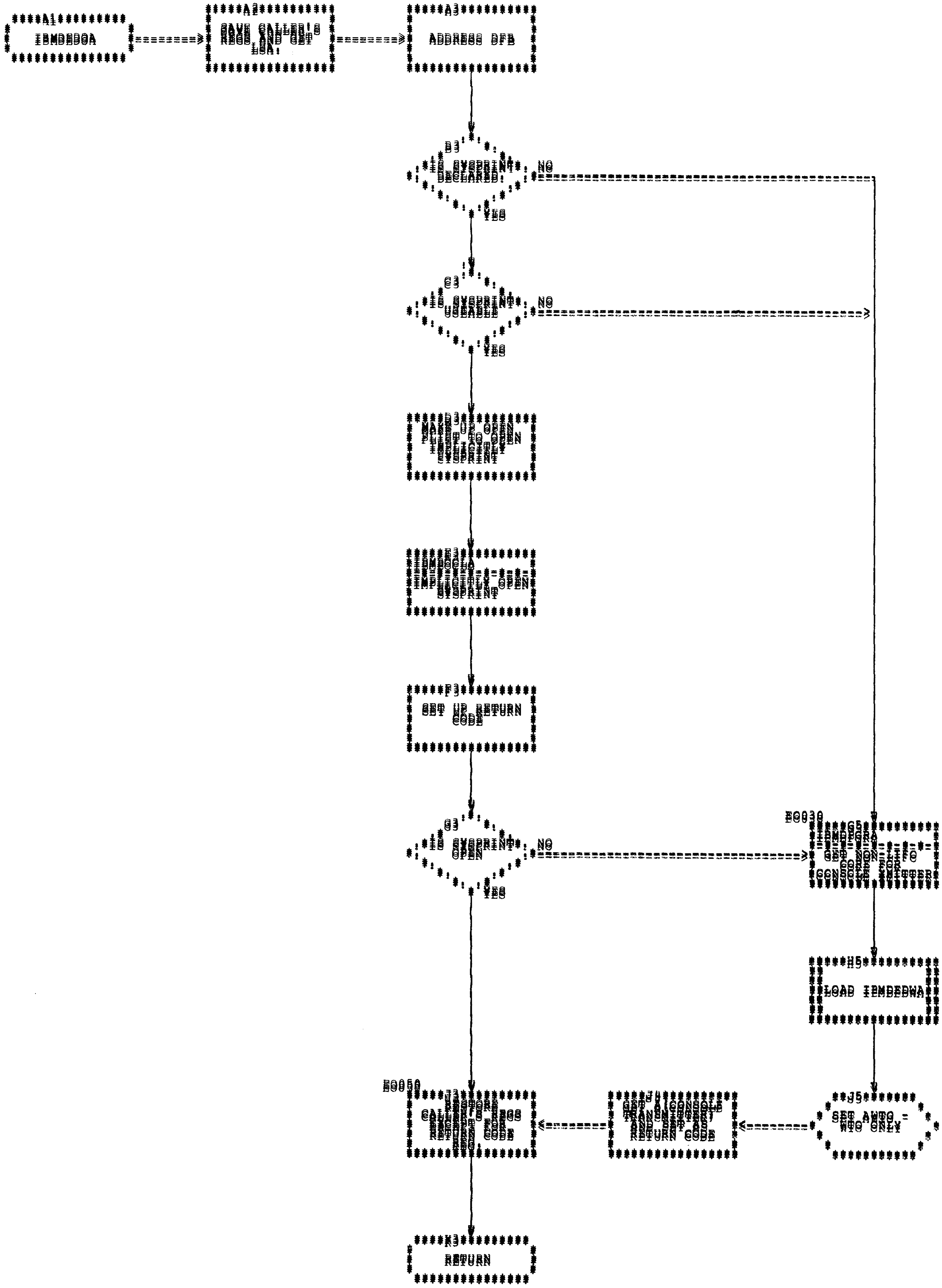


Chart DLDO. Open diagnostic file

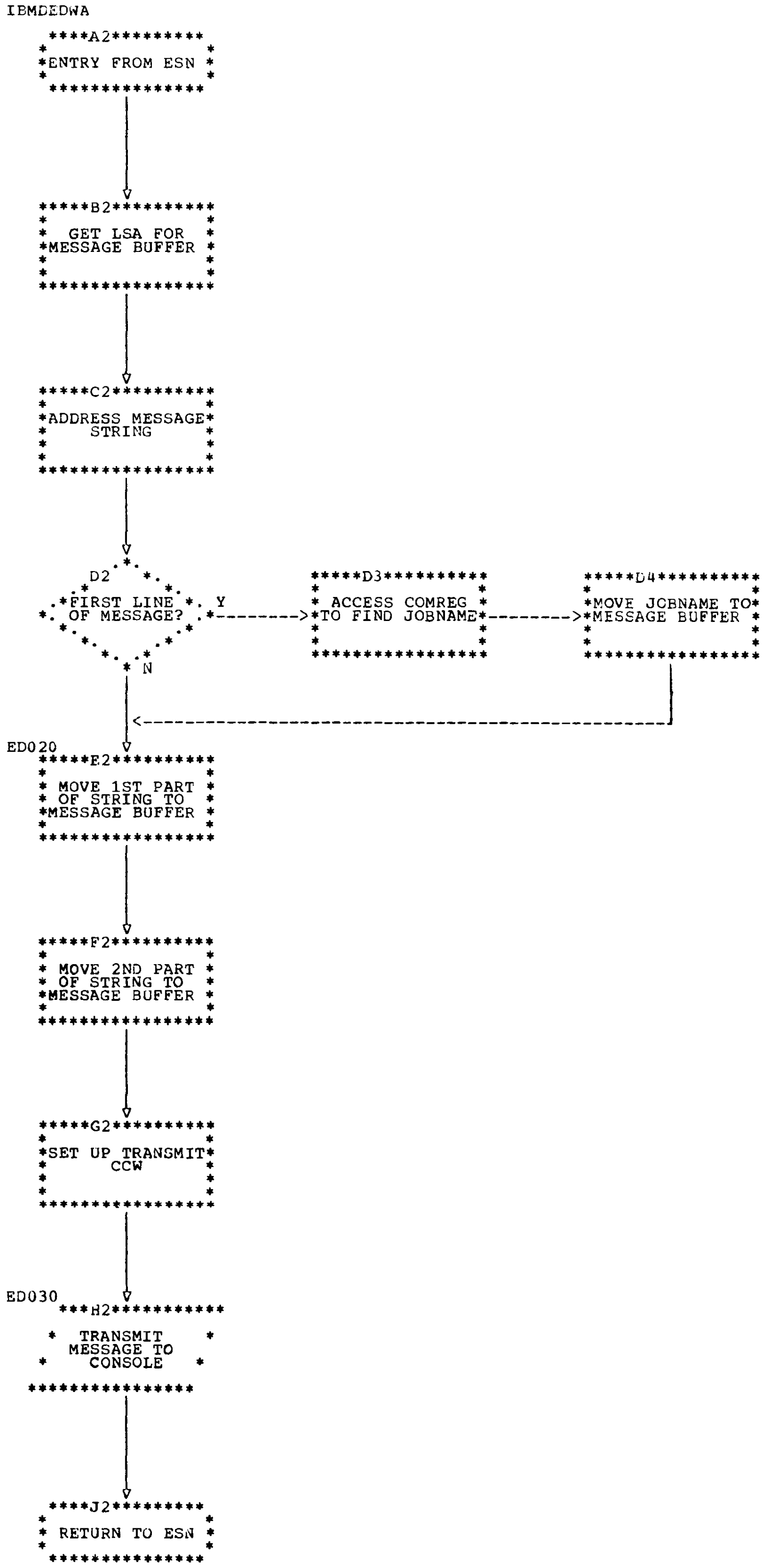

IBMDEDWA
A2
ENTRY FROM ESN
B2
GET LSA FOR MESSAGE BUFFER
C2
ADDRESS MESSAGE STRING
D2
FIRST LINE OF MESSAGE?
Y
N
D3
ACCESS COMREG TO FIND JOBNAME
D4
MOVE JOBNAME TO MESSAGE BUFFER
ED020
E2
MOVE 1ST PART OF STRING TO MESSAGE BUFFER
F2
MOVE 2ND PART OF STRING TO MESSAGE BUFFER
G2
SET UP TRANSMIT CCW
ED030
H2
TRANSMIT MESSAGE TO CONSOLE
J2
RETURN TO ESN

Chart DEDW. Console transmitter

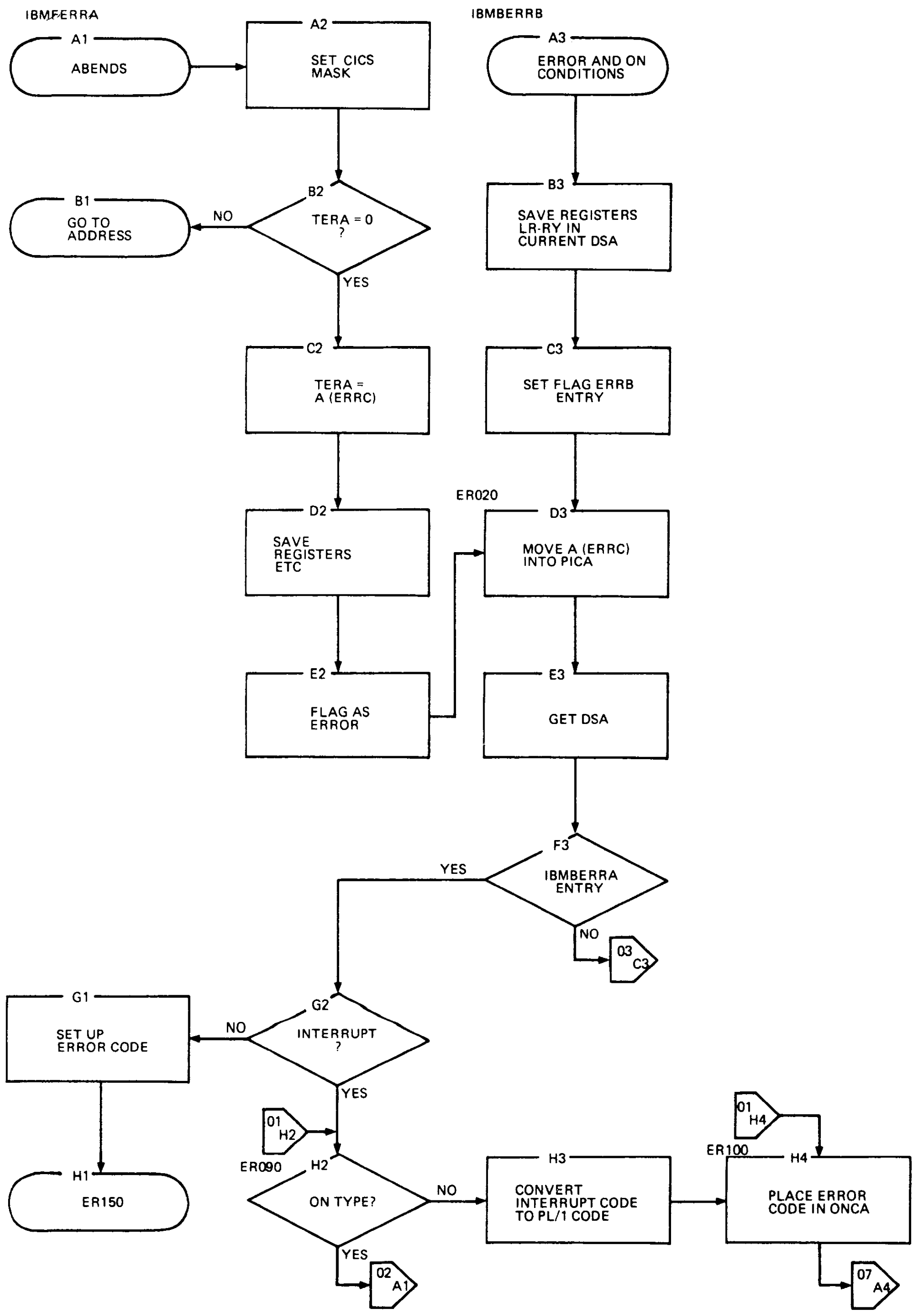


Chart FERR. Error handler - CICS (part 1 of 8)

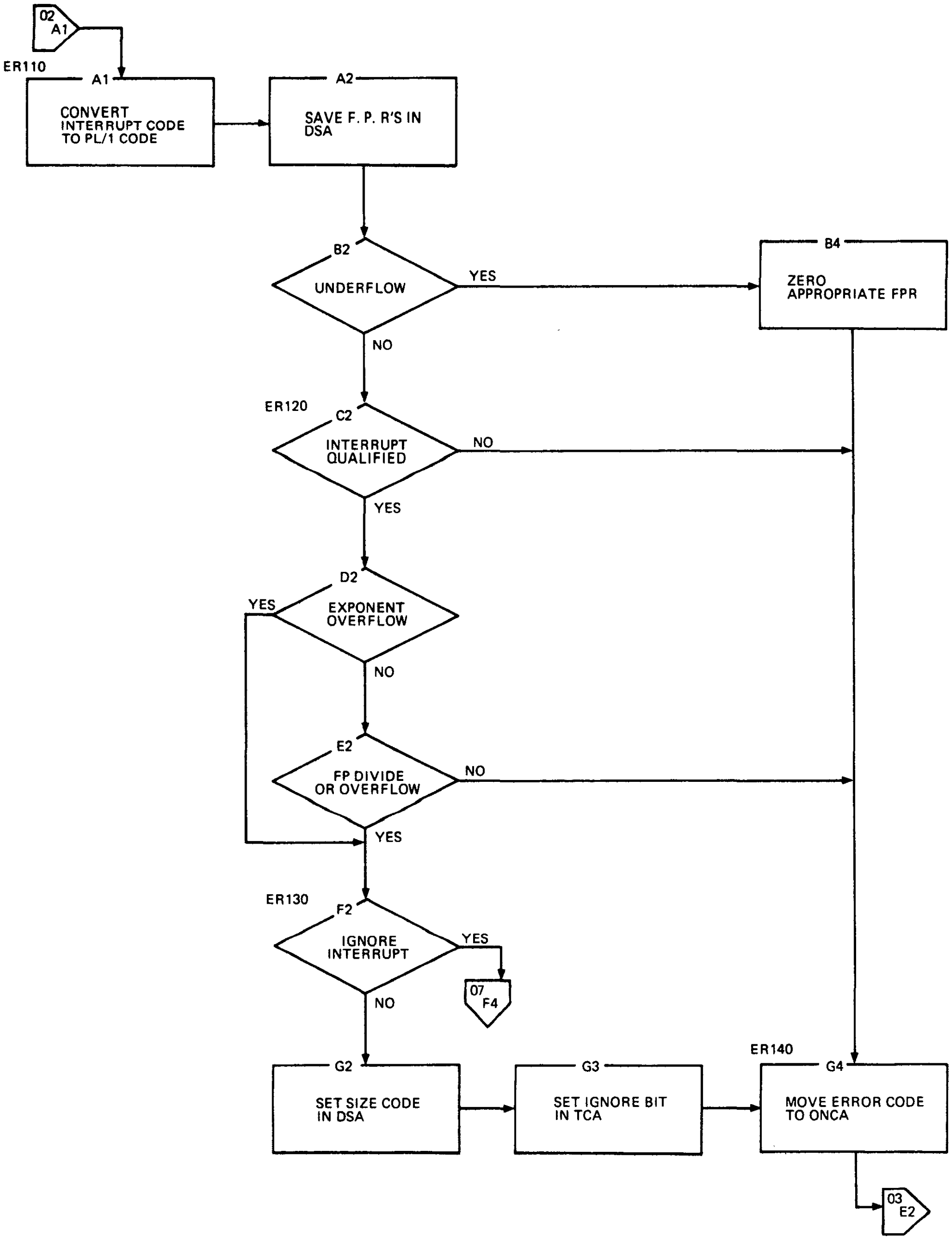
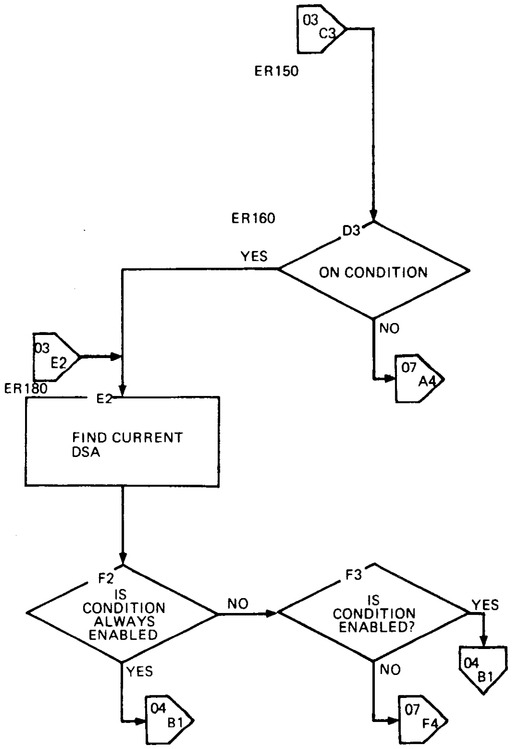
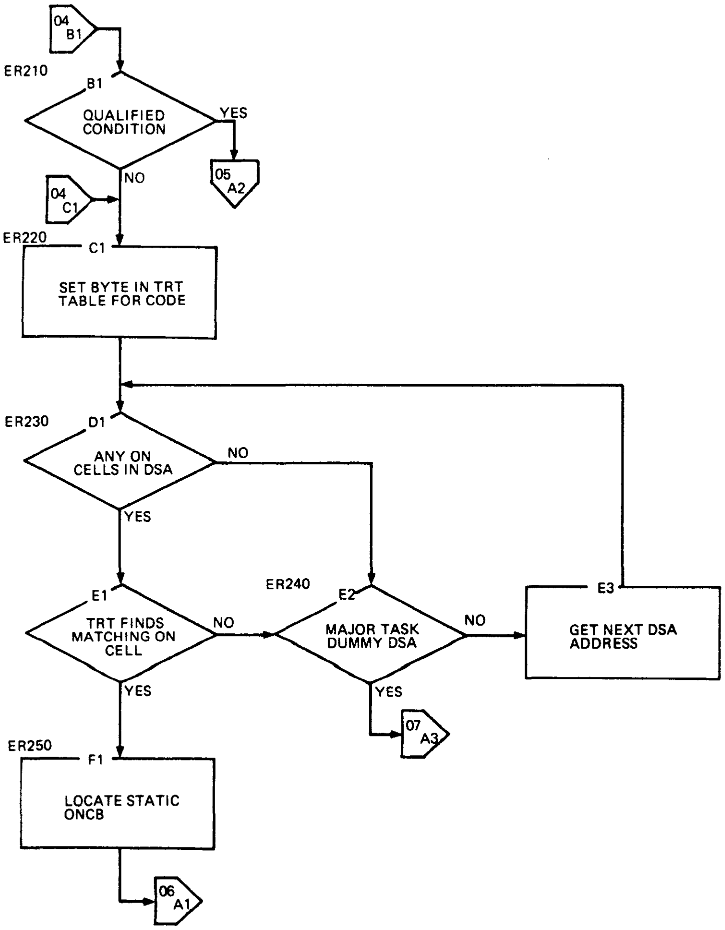


Chart FERR. Error handler - CICS (part 2 of 8)



|Chart FERP. Error handler - CICS (part 3 of 8)



|Chart FERR. Error handler - CICS (part 4 of 8)

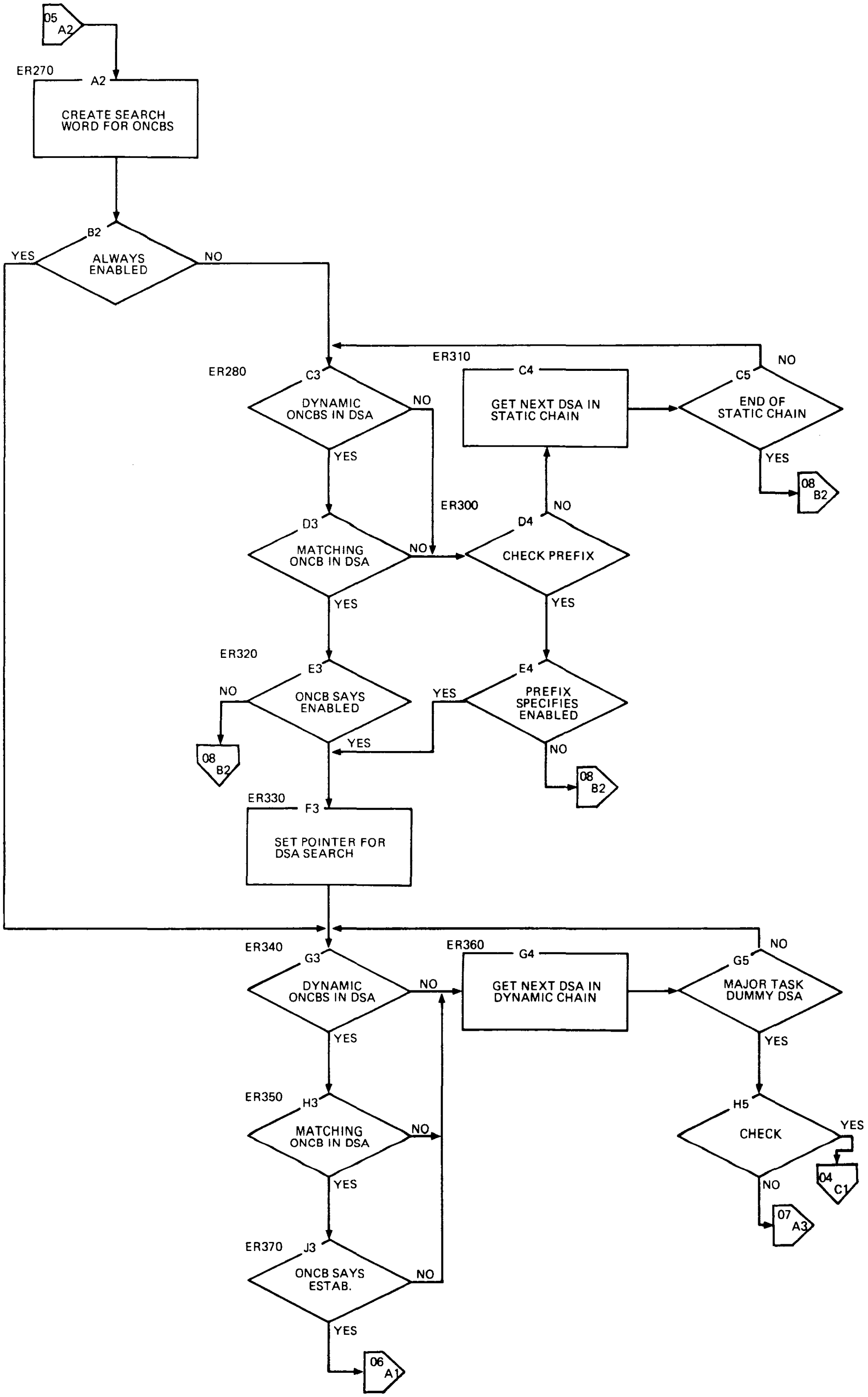


Chart FERR. Error handler - CICS (part 5 of 8)

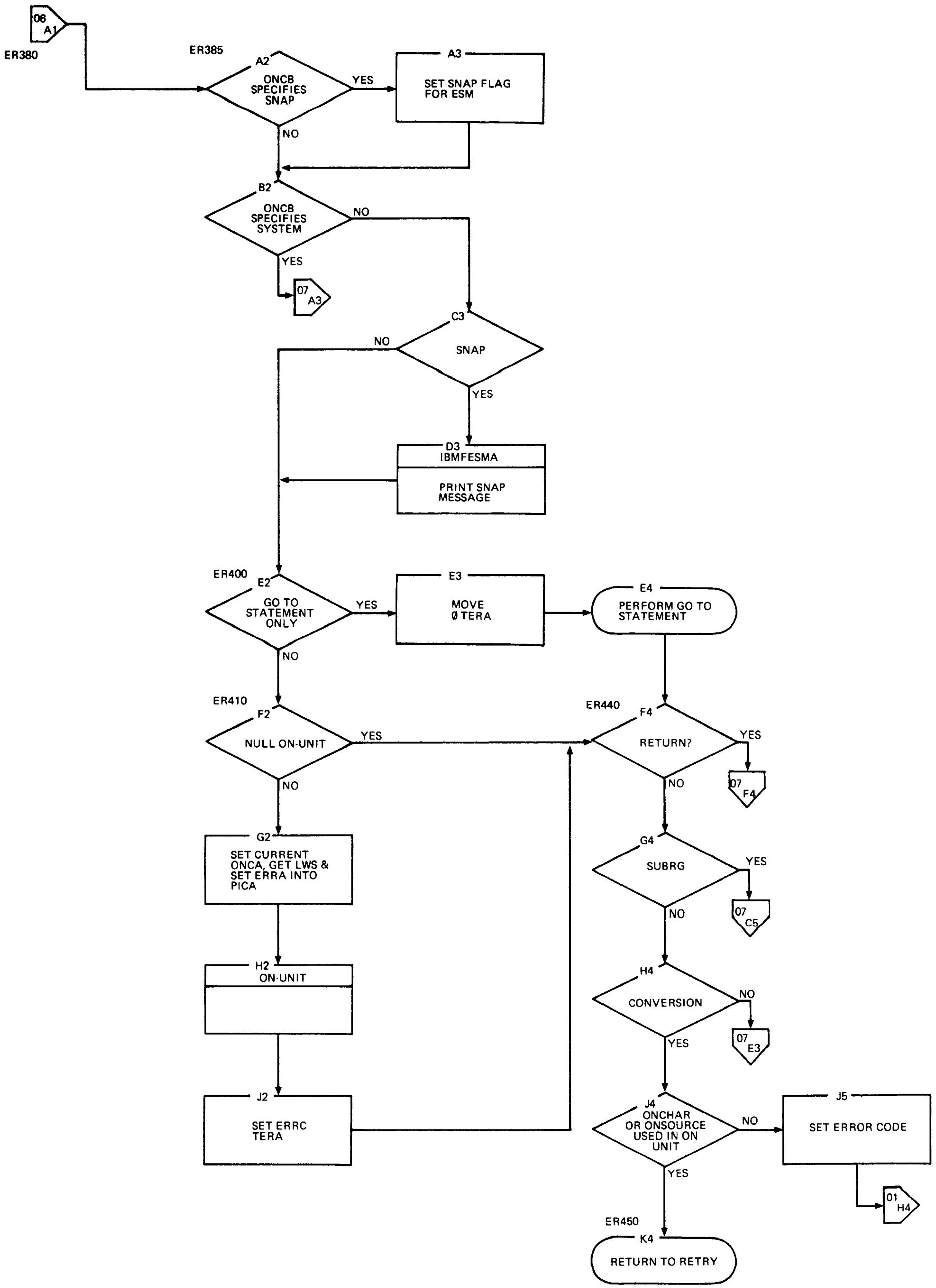


Chart FERR. Error handler - CICS (part 6 of 8)

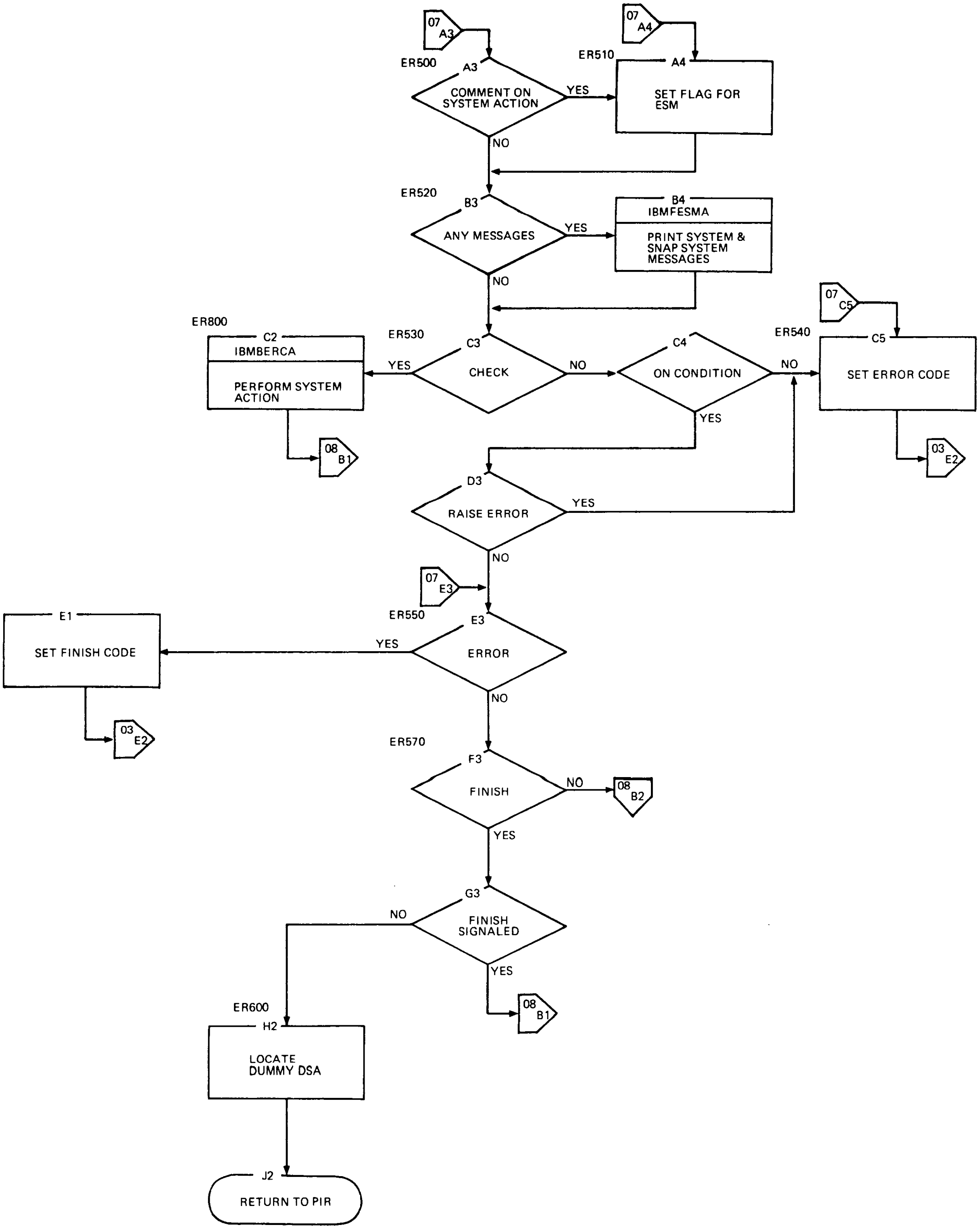


Chart FERR. Error handler - CICS (part 7 of 8)

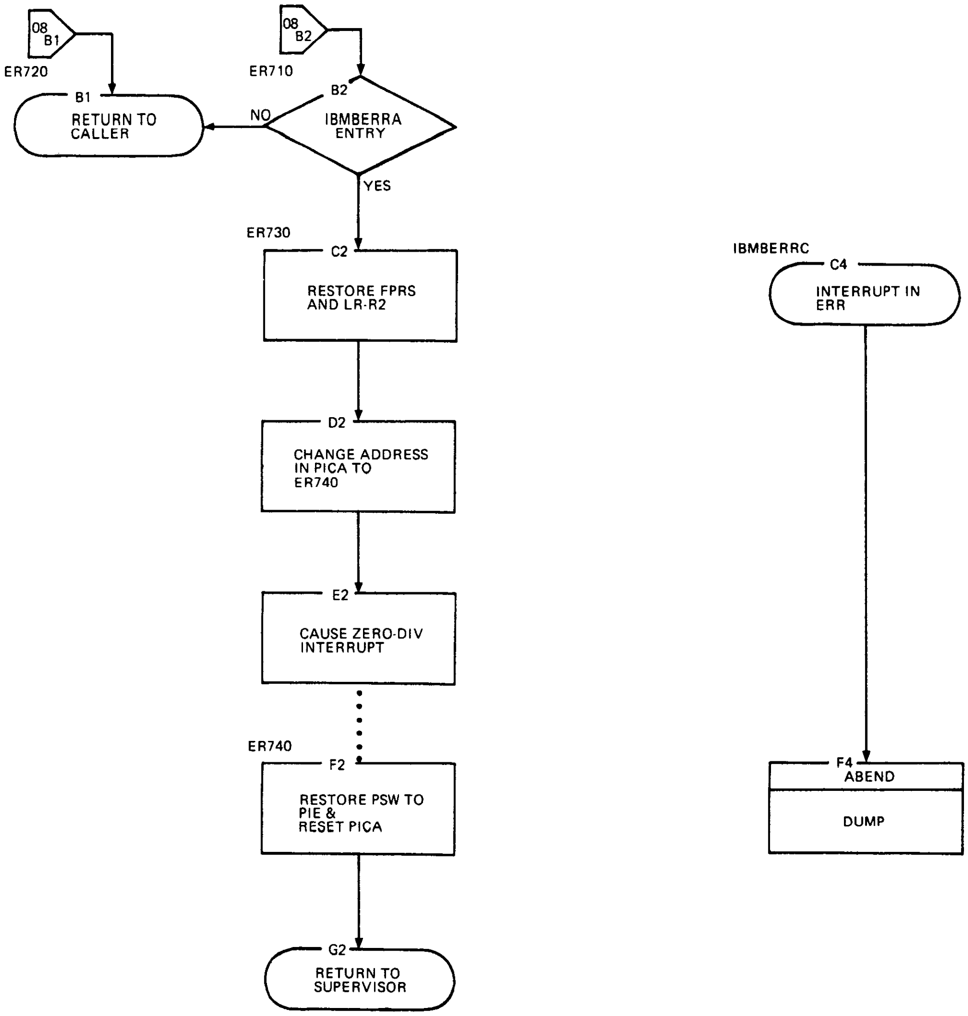


Chart FERR. Error handler - CICS (part 8 of 8)

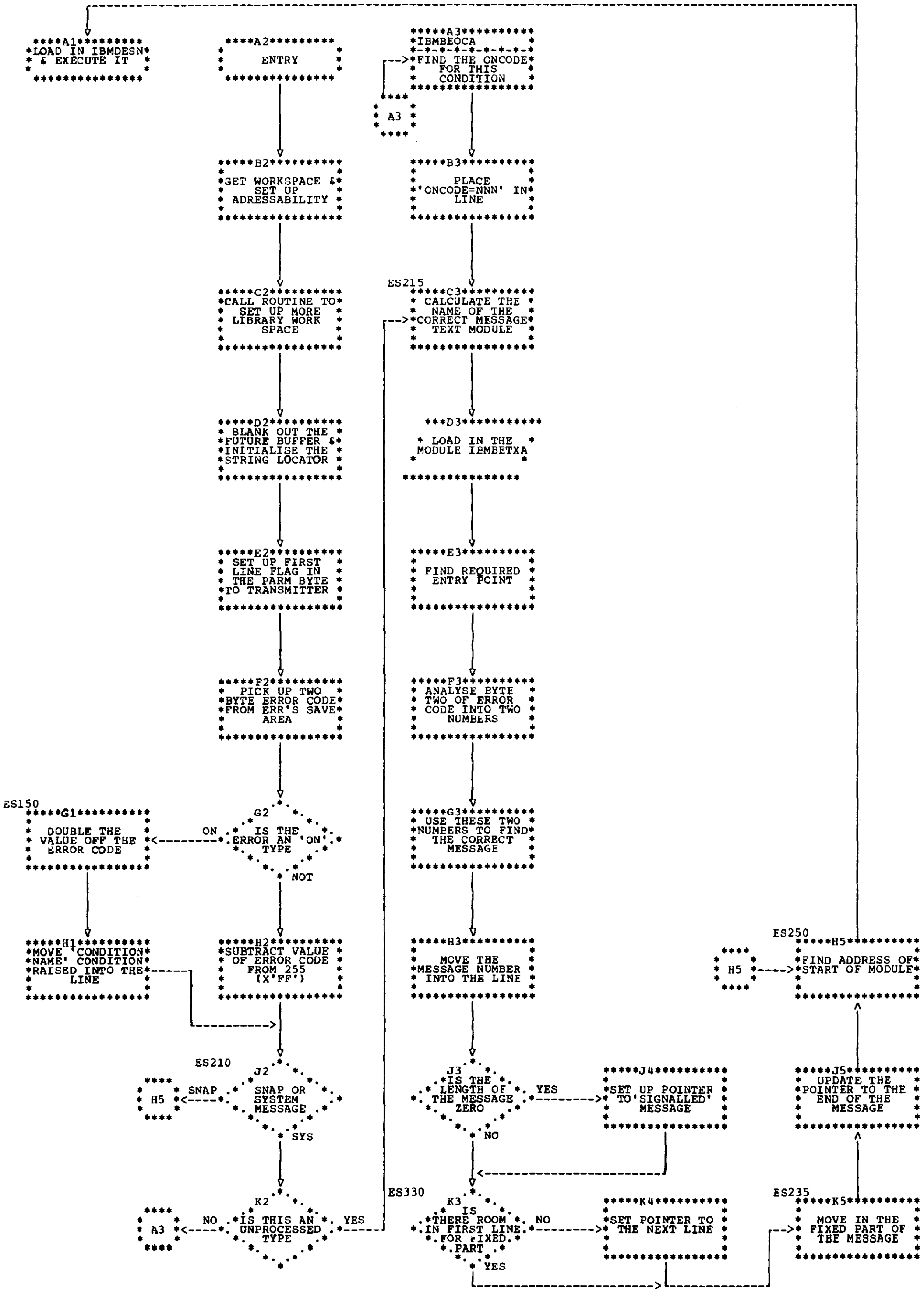

A1
LOAD IN IBMDESN & EXECUTE IT
A2
ENTRY
A3
IBMBEOCA
FIND THE ONCODE FOR THIS CONDITION
A3
B2
GET WORKSPACE & SET UP ADRESSABILITY
B3
PLACE 'ONCODE=NNN' IN LINE
C2
CALL ROUTINE TO SET UP MORE LIBRARY WORK SPACE
ES215
C3
CALCULATE THE NAME OF THE CORRECT MESSAGE TEXT MODULE
D2
BLANK OUT THE FUTURE BUFFER & INITIALISE THE STRING LOCATOR
D3
LOAD IN THE MODULE IBMBETXA
E2
SET UP FIRST LINE FLAG IN THE PARM BYTE TO TRANSMITTER
E3
FIND REQUIRED ENTRY POINT
F2
PICK UP TWO BYTE ERROR CODE FROM ERR'S SAVE AREA
F3
ANALYSE BYTE TWO OF ERROR CODE INTO TWO NUMBERS
ES150
G1
DOUBLE THE VALUE OFF THE ERROR CODE
G2
IS THE ERROR AN 'ON' TYPE
ON
NOT
G3
USE THESE TWO NUMBERS TO FIND THE CORRECT MESSAGE
H1
MOVE 'CONDITION NAME' CONDITION RAISED INTO THE LINE
H2
SUBTRACT VALUE OF ERROR CODE FROM 255 (X'FF')
H3
MOVE THE MESSAGE NUMBER INTO THE LINE
ES250
H5
H5
FIND ADDRESS OF START OF MODULE
ES210
J2
SNAP OR SYSTEM MESSAGE
SNAP
SYS
H5
J3
IS THE LENGTH OF THE MESSAGE ZERO
YES
NO
J4
SET UP POINTER TO 'SIGNALLED' MESSAGE
J5
UPDATE THE POINTER TO THE END OF THE MESSAGE
K2
IS THIS AN UNPROCESSED TYPE
NO
YES
A3
ES330
K3
IS THERE ROOM IN FIRST LINE FOR FIXED PART
NO
YES
K4
SET POINTER TO THE NEXT LINE
ES235
K5
MOVE IN THE FIXED PART OF THE MESSAGE

Chart DESM. Error message module phase 1

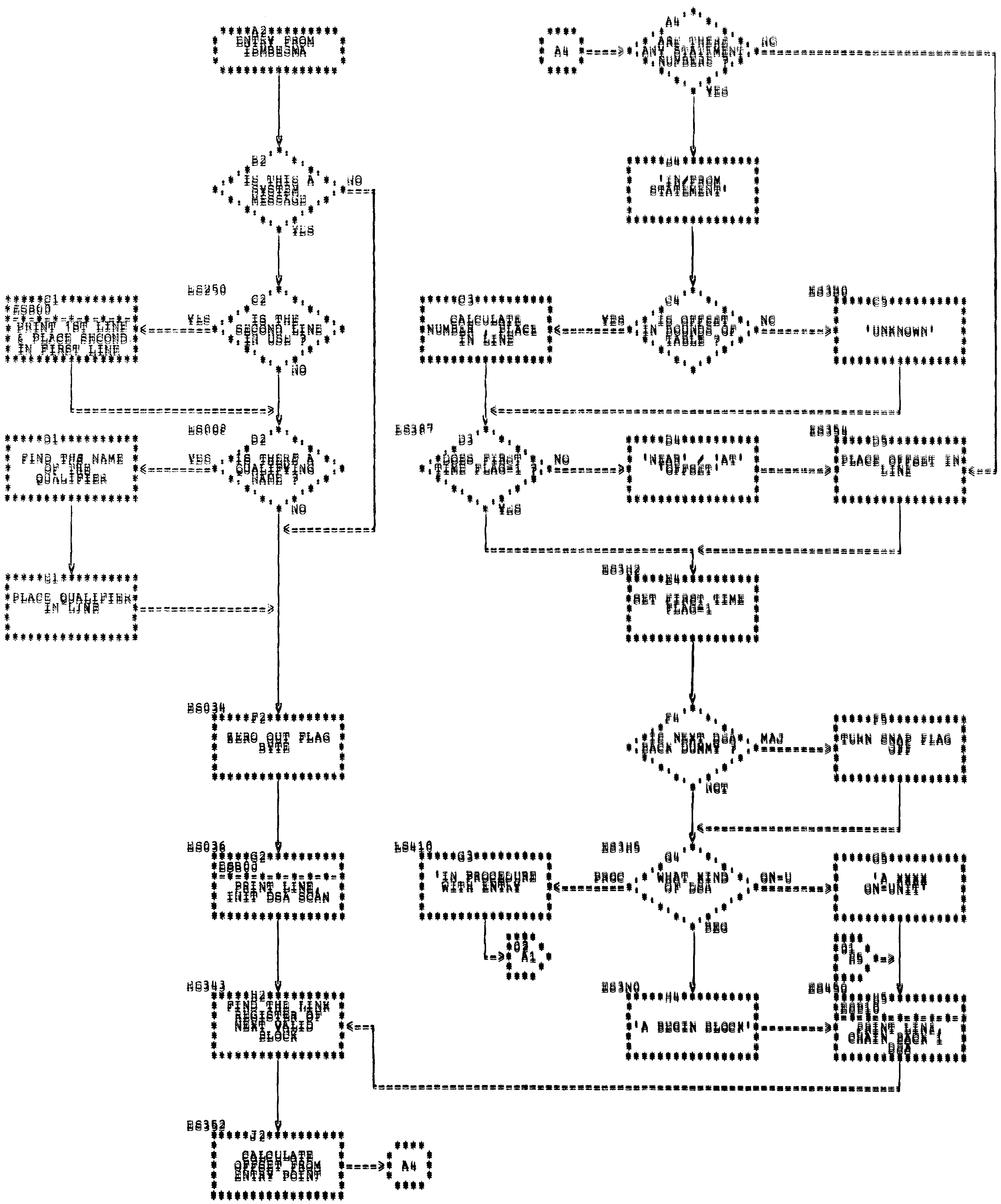


Chart DESN. Error message module phase 2 (part 1 of 3)

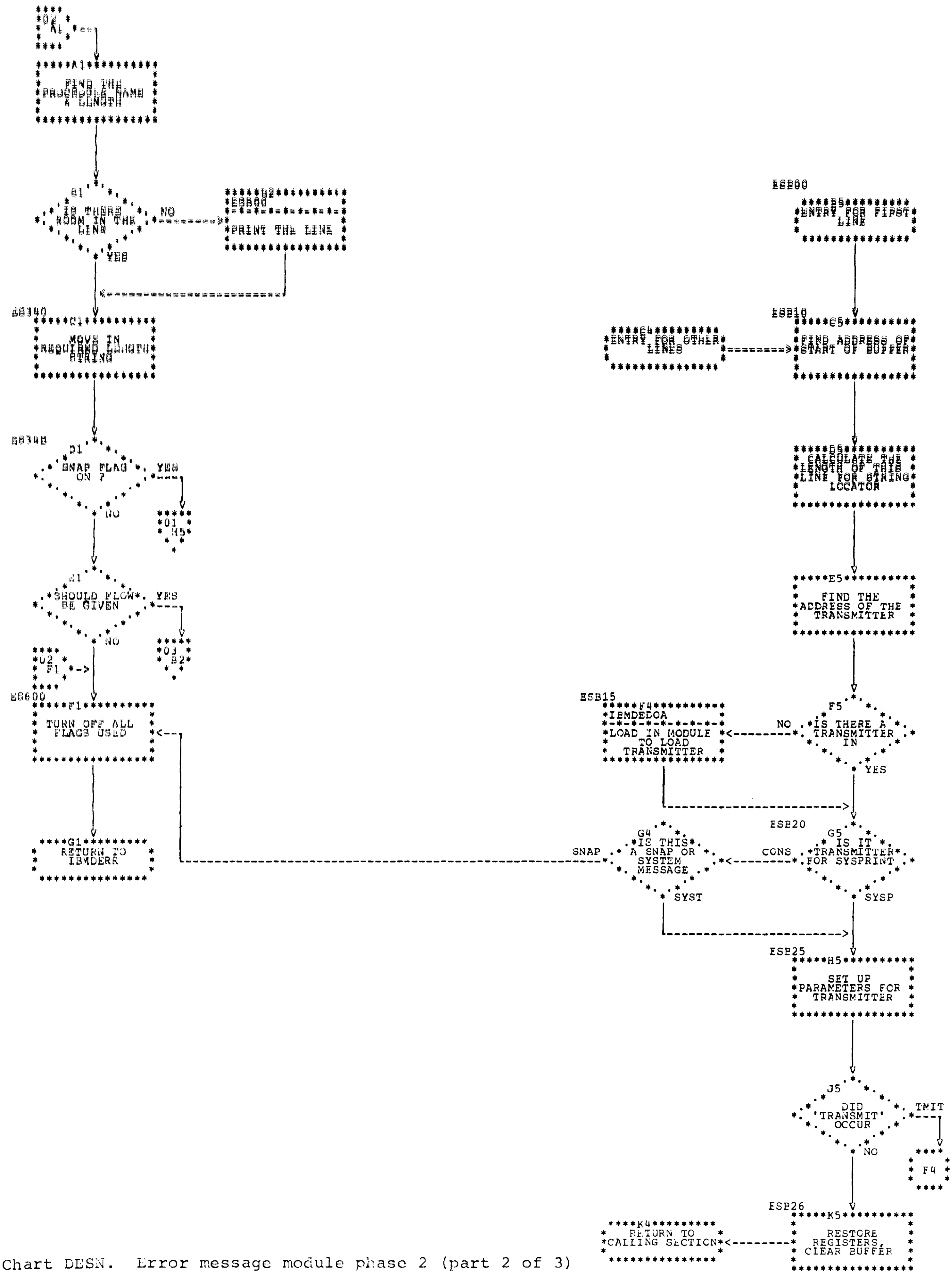

02 A1
A1 FIND THE PROCEDURE NAME & LENGTH
B1 IS THERE ROOM IN THE LINE
NO
YES
B2 ESB00 PRINT THE LINE
ES340
C1 MOVE IN REQUIRED LENGTH STRING
ES34B
D1 SNAP FLAG ON ?
YES
NO
01 H5
E1 SHOULD FLOW BE GIVEN
YES
NO
03 B2
02 F1
ES600
F1 TURN OFF ALL FLAGS USED
G1 RETURN TO IBMDERR
ESB00
B5 ENTRY FOR FIPST LINE
ESB10
C4 ENTRY FOR OTHER LINES
C5 FIND ADDRESS OF START OF BUFFER
D5 CALCULATE THE LENGTH OF THIS LINE FOR STRING LOCATOR
E5 FIND THE ADDRESS OF THE TRANSMITTER
ESB15
F4 IBMDEDOA LOAD IN MODULE TO LOAD TRANSMITTER
F5 IS THERE A TRANSMITTER IN
NO
YES
G4 IS THIS A SNAP OR SYSTEM MESSAGE
SNAP
SYST
ESB20
G5 IS IT TRANSMITTER FOR SYSPRINT
CONS
SYSP
ESB25
H5 SET UP PARAMETERS FOR TRANSMITTER
J5 DID 'TRANSMIT' OCCUR
TMIT
NO
F4
ESB26
K4 RETURN TO CALLING SECTICN
K5 RESTORE REGISTERS, CLEAR BUFFER

Chart DESN. Error message module phase 2 (part 2 of 3)

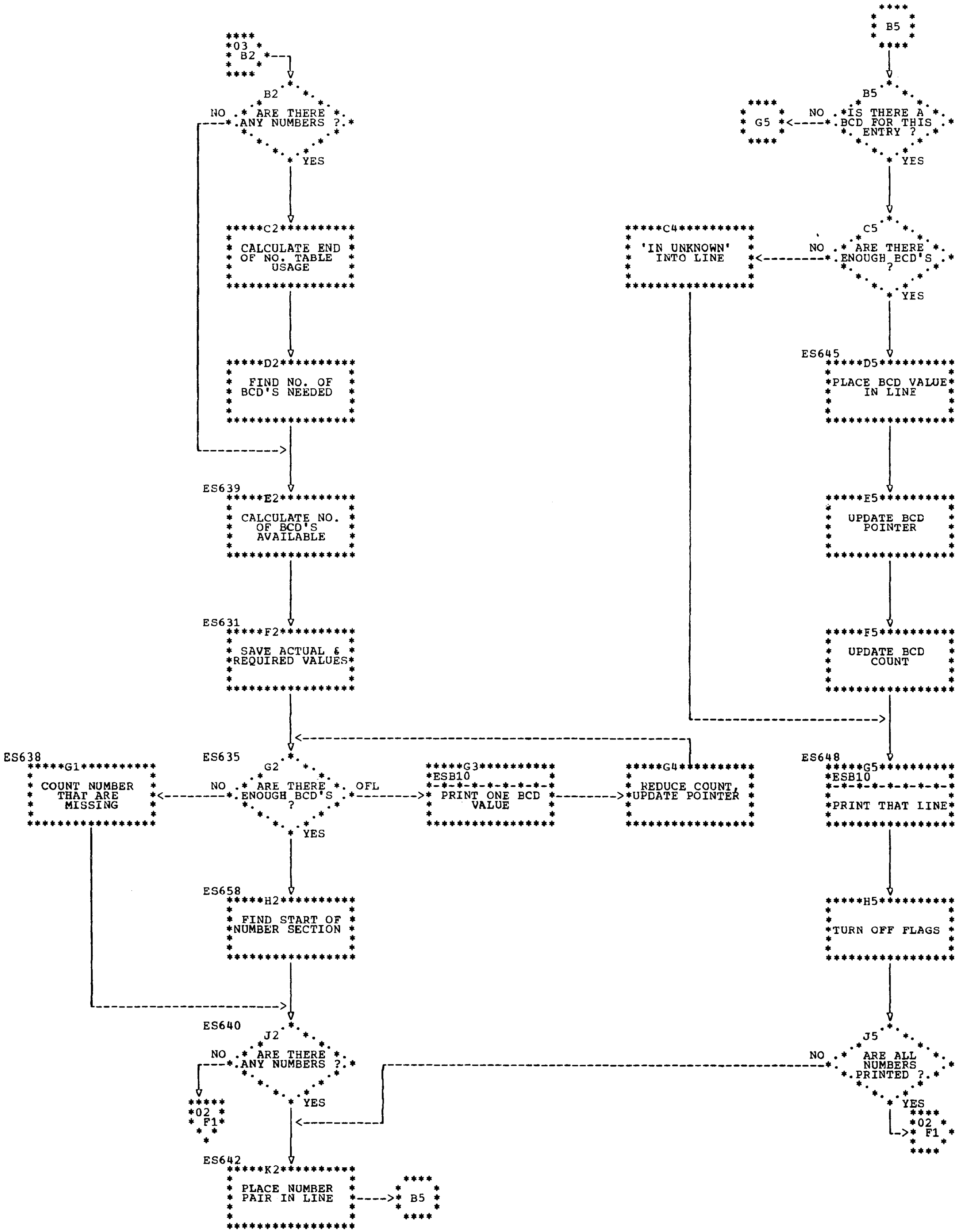
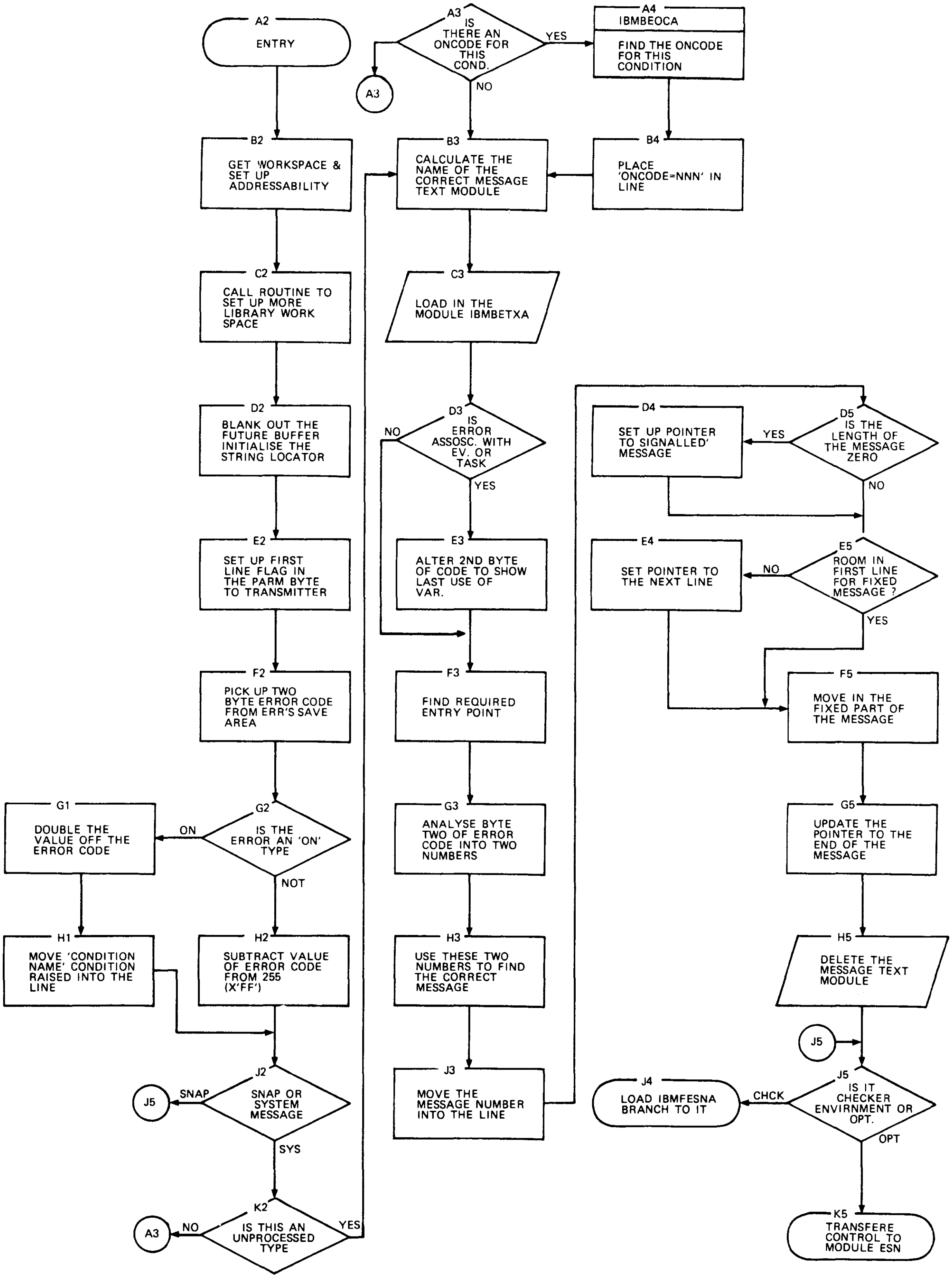

03 B2
B2 ARE THERE ANY NUMBERS ?
NO
YES
C2 CALCULATE END OF NO. TABLE USAGE
D2 FIND NO. OF BCD'S NEEDED
ES639 E2 CALCULATE NO. OF BCD'S AVAILABLE
ES631 F2 SAVE ACTUAL & REQUIRED VALUES
ES638 G1 COUNT NUMBER THAT ARE MISSING
ES635 G2 ARE THERE ENOUGH BCD'S ?
NO
OFL
YES
G3 ESB10 PRINT ONE BCD VALUE
G4 REDUCE COUNT, UPDATE POINTER
ES658 H2 FIND START OF NUMBER SECTION
ES640 J2 ARE THERE ANY NUMBERS ?
NO
YES
02 F1
ES642 K2 PLACE NUMBER PAIR IN LINE
B5
B5
B5 IS THERE A BCD FOR THIS ENTRY ?
NO
G5
YES
C5 ARE THERE ENOUGH BCD'S ?
NO
C4 'IN UNKNOWN' INTO LINE
YES
ES645 D5 PLACE BCD VALUE IN LINE
E5 UPDATE BCD POINTER
F5 UPDATE BCD COUNT
ES648 G5 ESB10 PRINT THAT LINE
H5 TURN OFF FLAGS
J5 ARE ALL NUMBERS PRINTED ?
NO
YES
02 F1

Chart DESN. Error message module phase 2 (part 3 of 3)


A2
ENTRY
B2
GET WORKSPACE & SET UP ADDRESSABILITY
C2
CALL ROUTINE TO SET UP MORE LIBRARY WORK SPACE
D2
BLANK OUT THE FUTURE BUFFER INITIALISE THE STRING LOCATOR
E2
SET UP FIRST LINE FLAG IN THE PARM BYTE TO TRANSMITTER
F2
PICK UP TWO BYTE ERROR CODE FROM ERR'S SAVE AREA
G1
DOUBLE THE VALUE OFF THE ERROR CODE
ON
G2
IS THE ERROR AN 'ON' TYPE
NOT
H1
MOVE 'CONDITION NAME' CONDITION RAISED INTO THE LINE
H2
SUBTRACT VALUE OF ERROR CODE FROM 255 (X'FF')
J5
SNAP
J2
SNAP OR SYSTEM MESSAGE
SYS
A3
NO
K2
IS THIS AN UNPROCESSED TYPE
YES
A3
A3
IS THERE AN ONCODE FOR THIS COND.
YES
NO
A4
IBMBEOCA
FIND THE ONCODE FOR THIS CONDITION
B3
CALCULATE THE NAME OF THE CORRECT MESSAGE TEXT MODULE
B4
PLACE 'ONCODE=NNN' IN LINE
C3
LOAD IN THE MODULE IBMBETXA
D3
IS ERROR ASSOSC. WITH EV. OR TASK
NO
YES
E3
ALTER 2ND BYTE OF CODE TO SHOW LAST USE OF VAR.
F3
FIND REQUIRED ENTRY POINT
G3
ANALYSE BYTE TWO OF ERROR CODE INTO TWO NUMBERS
H3
USE THESE TWO NUMBERS TO FIND THE CORRECT MESSAGE
J3
MOVE THE MESSAGE NUMBER INTO THE LINE
D4
SET UP POINTER TO SIGNALLED' MESSAGE
YES
D5
IS THE LENGTH OF THE MESSAGE ZERO
NO
E4
SET POINTER TO THE NEXT LINE
NO
E5
ROOM IN FIRST LINE FOR FIXED MESSAGE ?
YES
F5
MOVE IN THE FIXED PART OF THE MESSAGE
G5
UPDATE THE POINTER TO THE END OF THE MESSAGE
H5
DELETE THE MESSAGE TEXT MODULE
J5
J4
LOAD IBMFESNA BRANCH TO IT
CHCK
J5
IS IT CHECKER ENVIRNMENT OR OPT.
OPT
K5
TRANSFERE CONTROL TO MODULE ESN

|Chart FESM. Error message routine Phase 1 - CICS

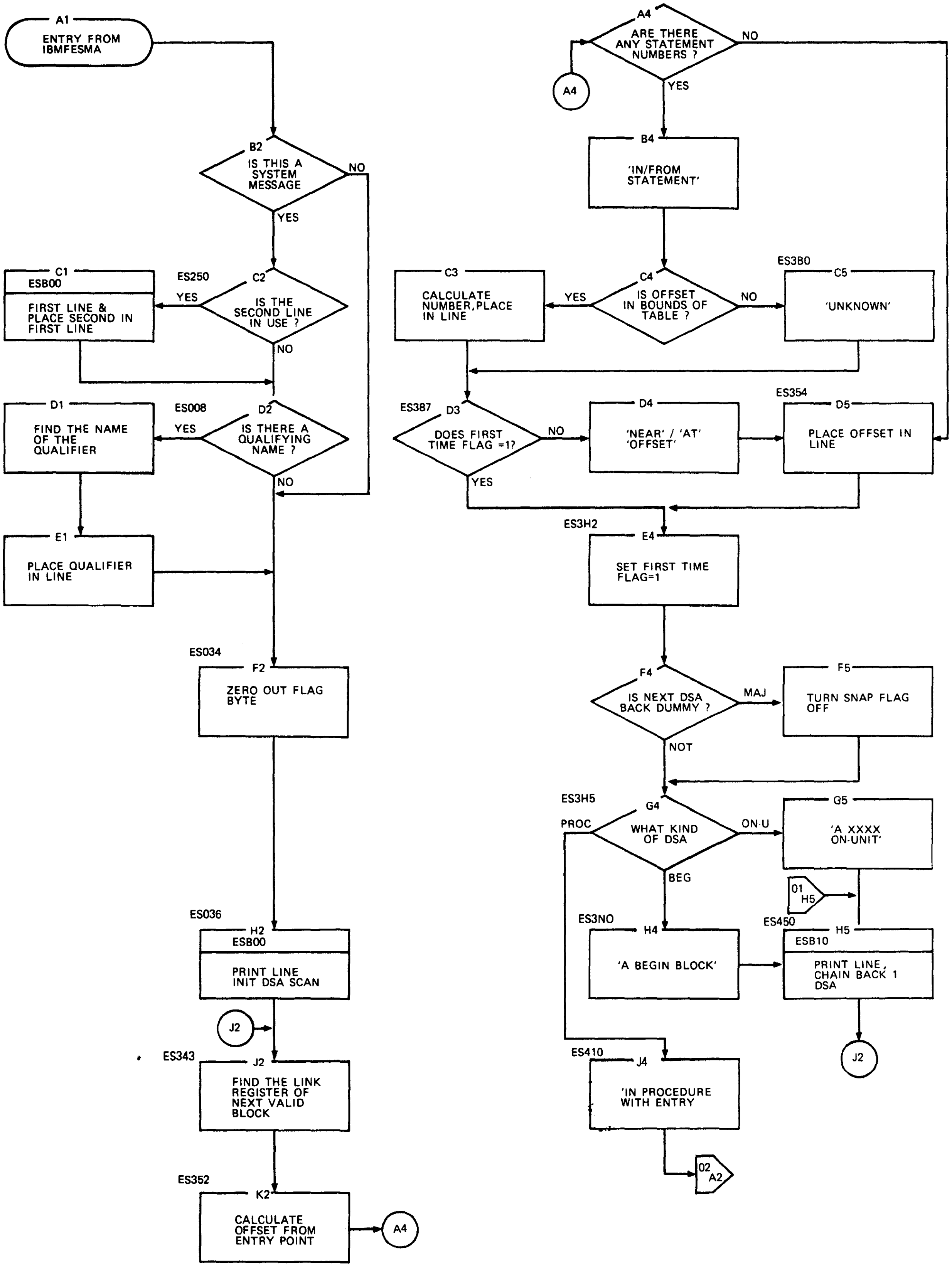


Chart FESN. Error message routine Phase 2 - CICS (part 1 of 3)

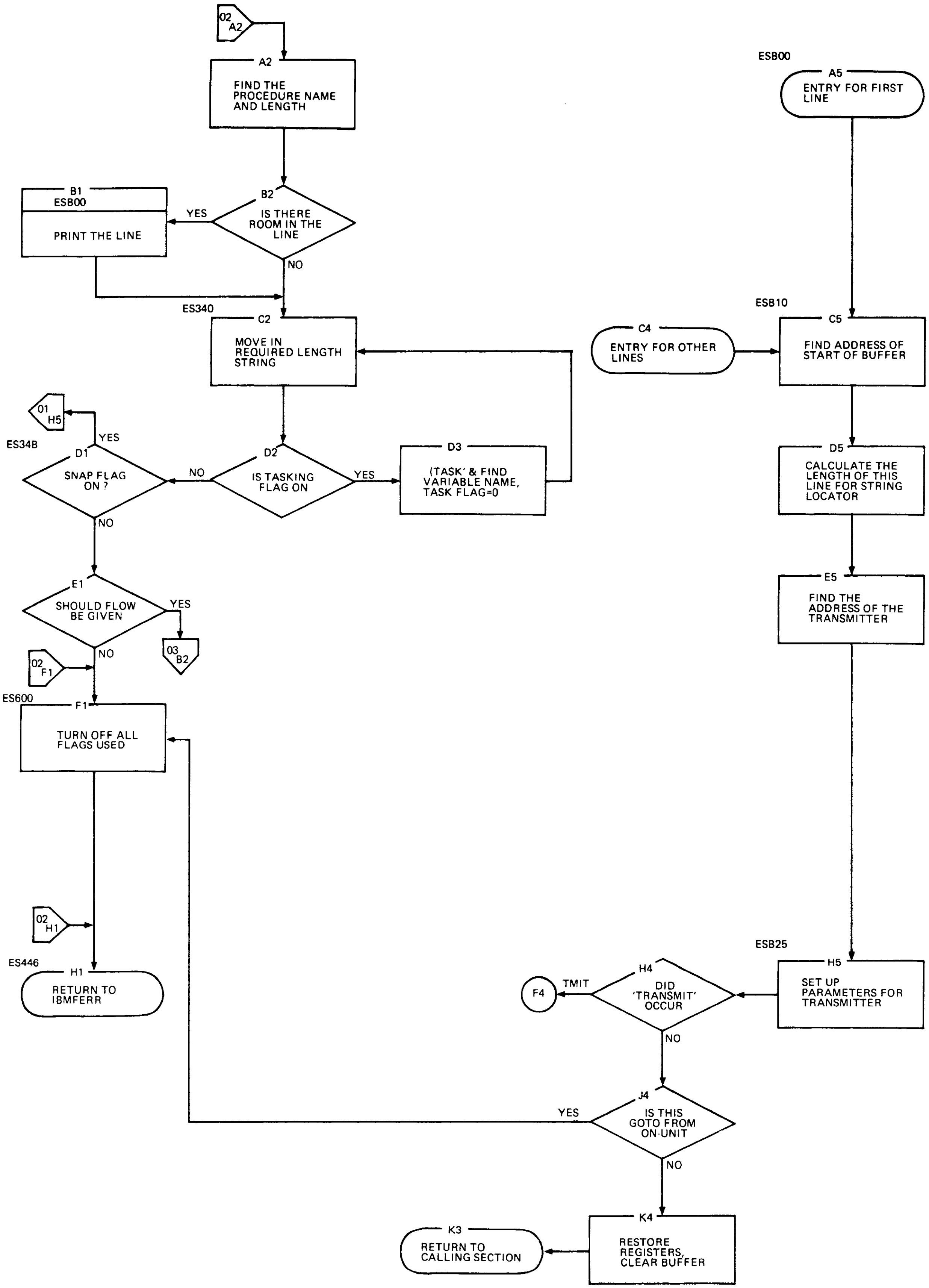
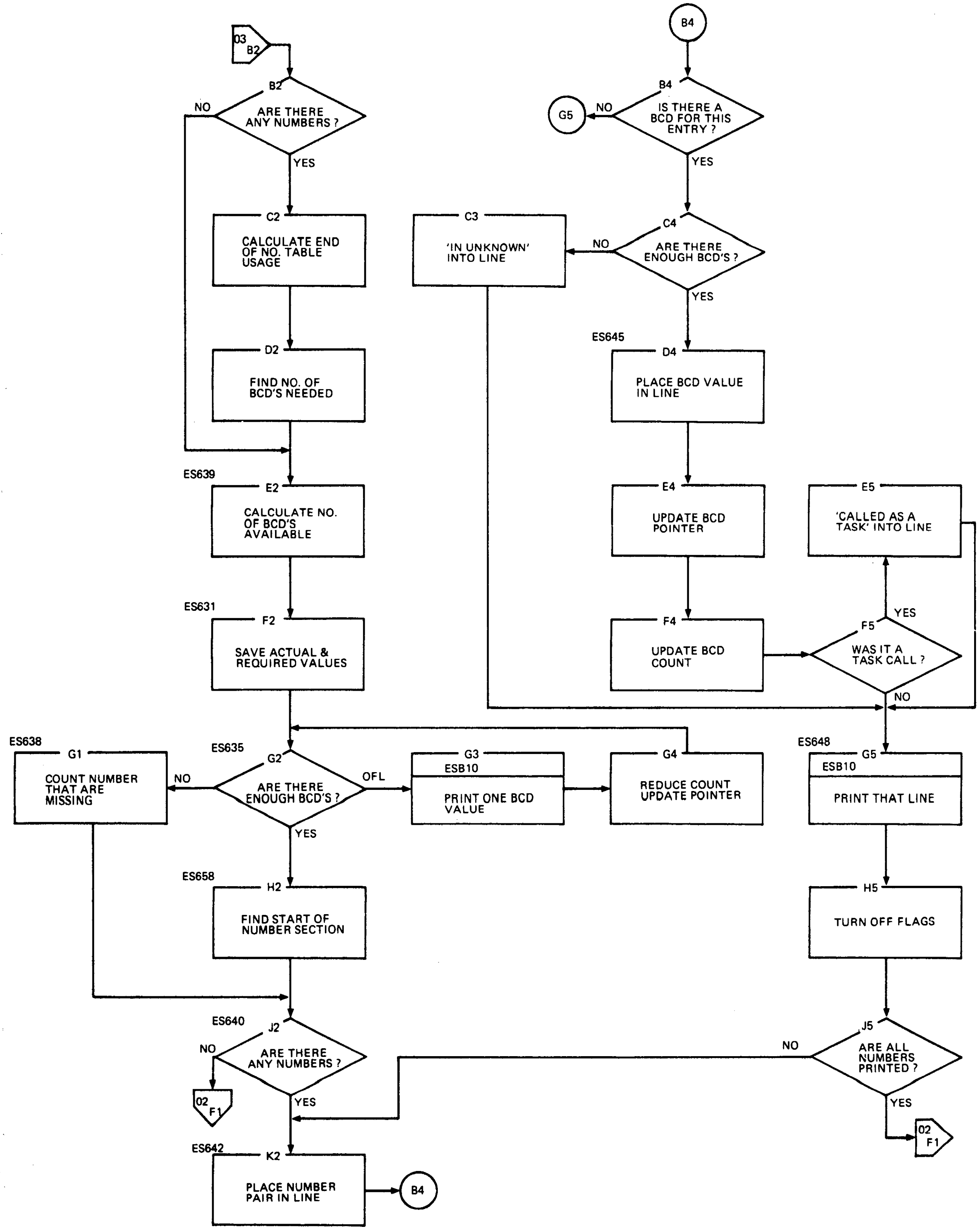

02
A2
A2
FIND THE PROCEDURE NAME AND LENGTH
B1
ESB00
PRINT THE LINE
YES
B2
IS THERE ROOM IN THE LINE
NO
ES340
C2
MOVE IN REQUIRED LENGTH STRING
01
H5
YES
ES34B
D1
SNAP FLAG ON ?
NO
D2
IS TASKING FLAG ON
YES
D3
(TASK' & FIND VARIABLE NAME, TASK FLAG=0
NO
E1
SHOULD FLOW BE GIVEN
YES
03
B2
NO
02
F1
ES600
F1
TURN OFF ALL FLAGS USED
02
H1
ES446
H1
RETURN TO IBMFERR
ESB00
A5
ENTRY FOR FIRST LINE
ESB10
C4
ENTRY FOR OTHER LINES
C5
FIND ADDRESS OF START OF BUFFER
D5
CALCULATE THE LENGTH OF THIS LINE FOR STRING LOCATOR
E5
FIND THE ADDRESS OF THE TRANSMITTER
ESB25
H5
SET UP PARAMETERS FOR TRANSMITTER
H4
DID 'TRANSMIT' OCCUR
TMIT
F4
NO
J4
IS THIS GOTO FROM ON-UNIT
YES
NO
K4
RESTORE REGISTERS, CLEAR BUFFER
K3
RETURN TO CALLING SECTION

Chart FESN. Error message routine Phase 2 - CICS (part 2 of 3)



|Chart FESN. Error message routine Phase 2 - CICS (part 3 of 3)

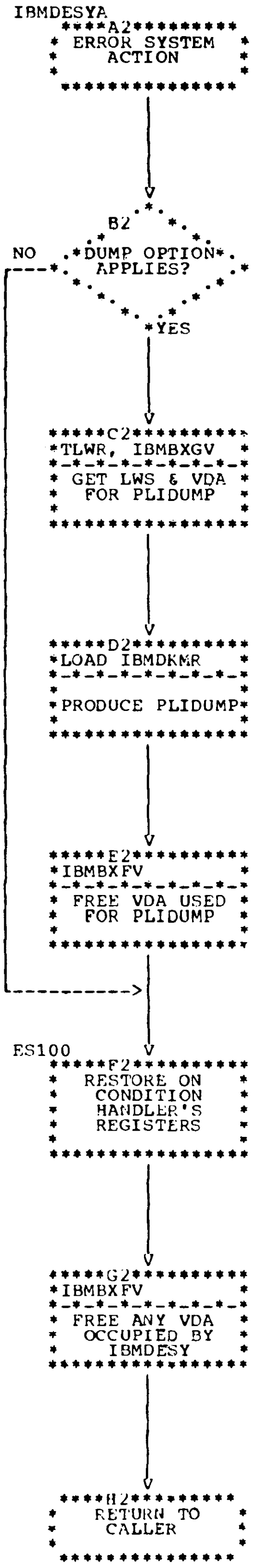


Chart DESY. Error system action

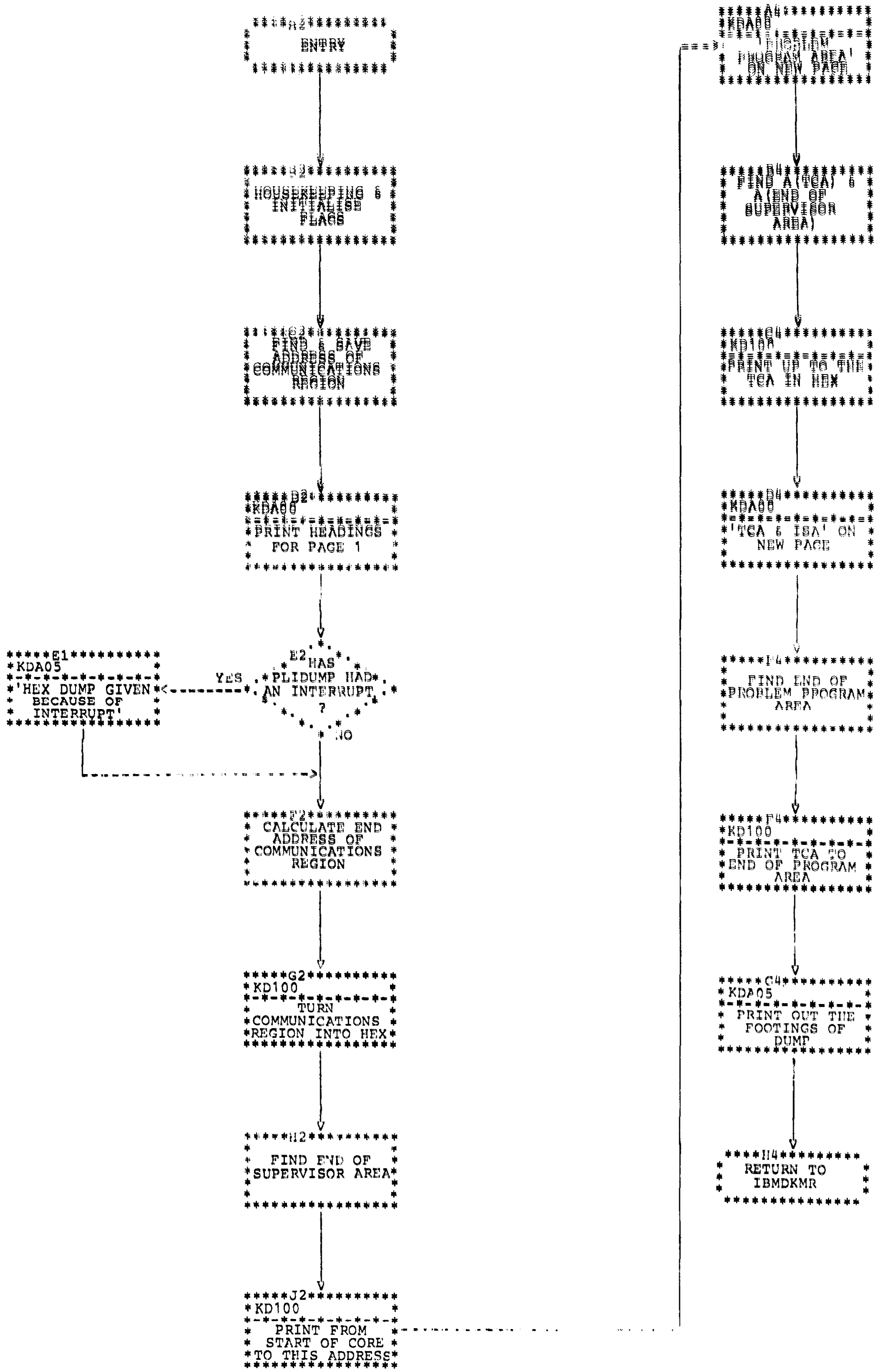


Chart DKDD. Hexadecimal dump module (part 1 of 2)

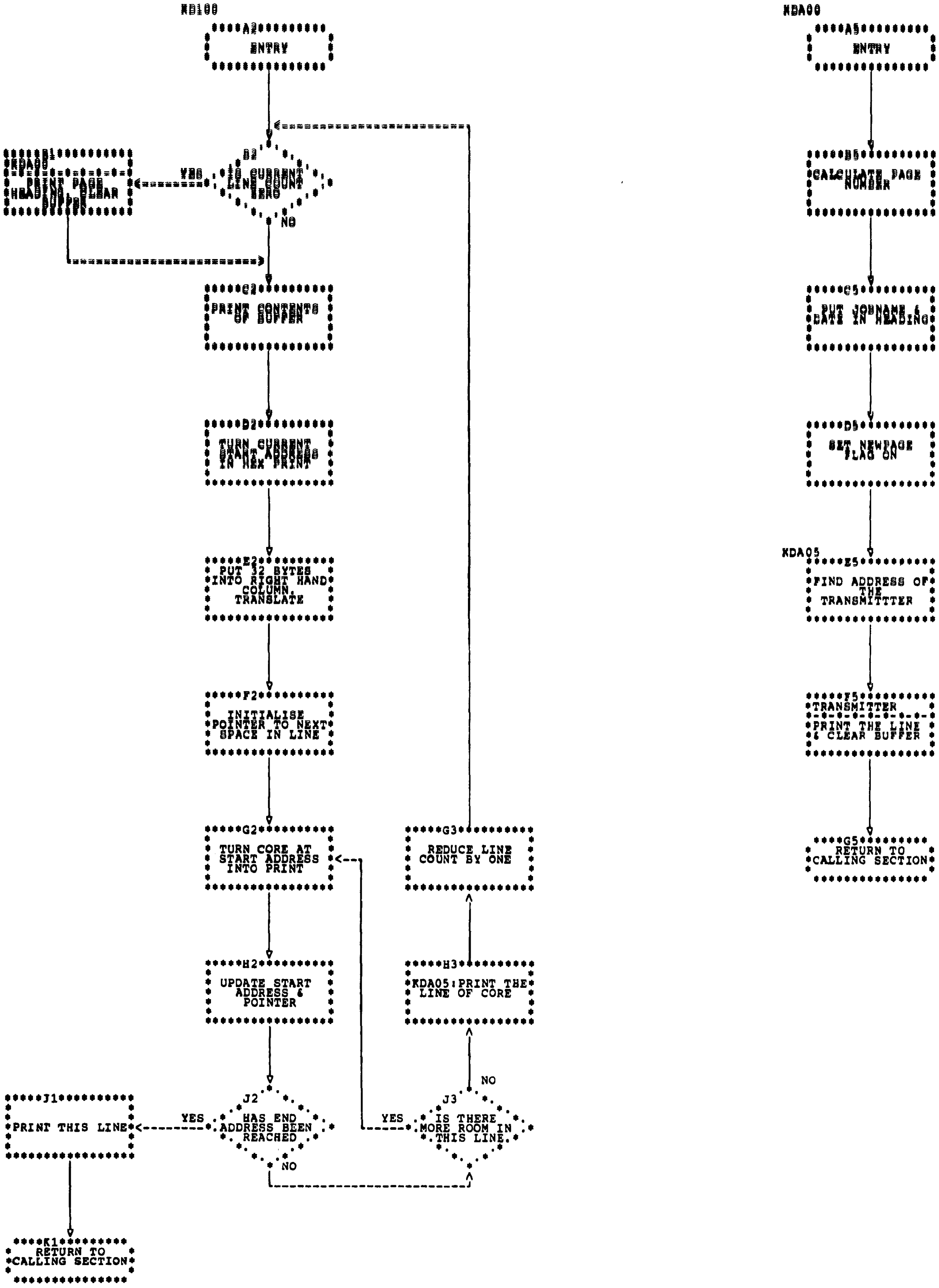

KD100
A2
ENTRY
B1
KDA00
PRINT PAGE HEADING, CLEAR BUFFER
YES
B2
IS CURRENT LINE COUNT ZERO
NO
C2
PRINT CONTENTS OF BUFFER
D2
TURN CURRENT START ADDRESS IN HEX PRINT
E2
PUT 32 BYTES INTO RIGHT HAND COLUMN. TRANSLATE
F2
INITIALISE POINTER TO NEXT SPACE IN LINE
G2
TURN CORE AT START ADDRESS INTO PRINT
H2
UPDATE START ADDRESS & POINTER
J1
PRINT THIS LINE
YES
J2
HAS END ADDRESS BEEN REACHED
NO
K1
RETURN TO CALLING SECTION
G3
REDUCE LINE COUNT BY ONE
H3
KDA05:PRINT THE LINE OF CORE
NO
YES
J3
IS THERE MORE ROOM IN THIS LINE
KDA00
A5
ENTRY
B5
CALCULATE PAGE NUMBER
C5
PUT JOBNAME & DATE IN HEADING
D5
SET NEWPAGE FLAG ON
KDA05
E5
FIND ADDRESS OF THE TRANSMITTTER
F5
TRANSMITTER
PRINT THE LINE & CLEAR BUFFER
G5
RETURN TO CALLING SECTION

Chart DKDD. Hexadecimal dump module (part 2 of 2)

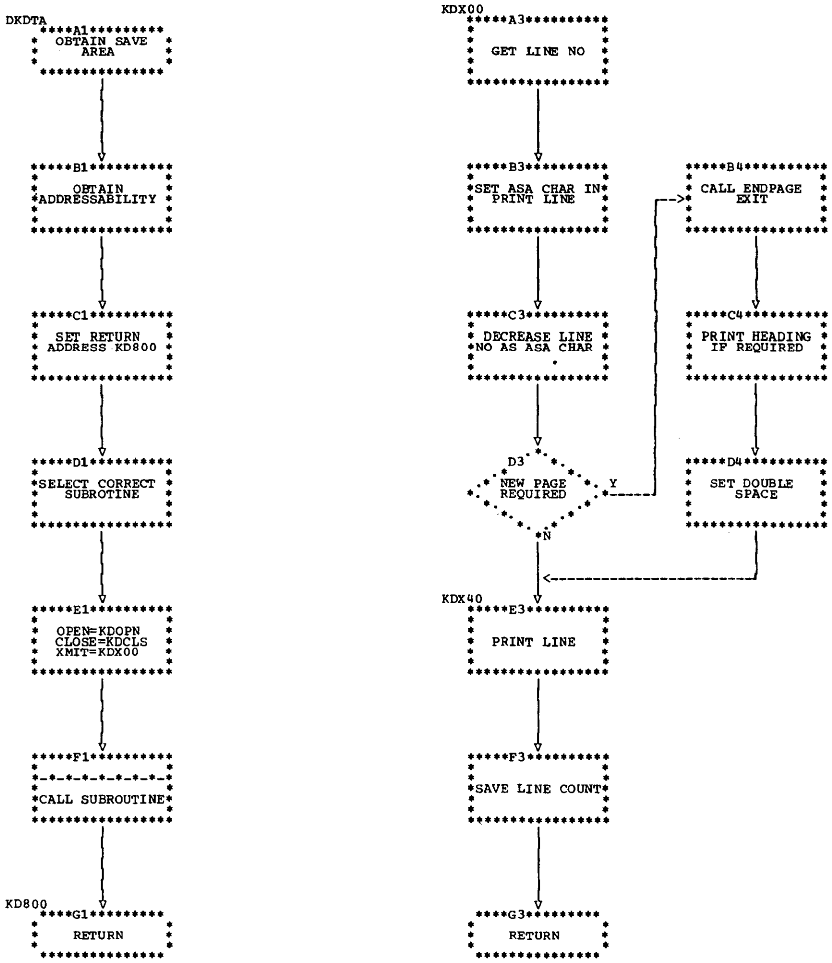


Chart DKDT. Dump file transmitter (part 1 of 2)

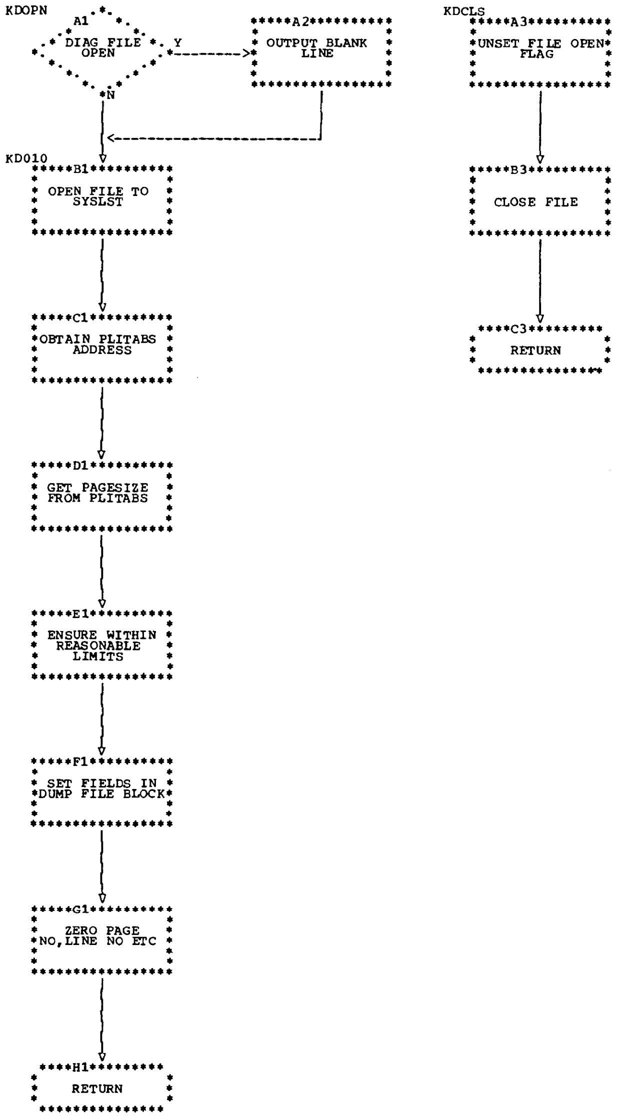


Chart DKDT. Dump file transmitter (part 2 of 2)

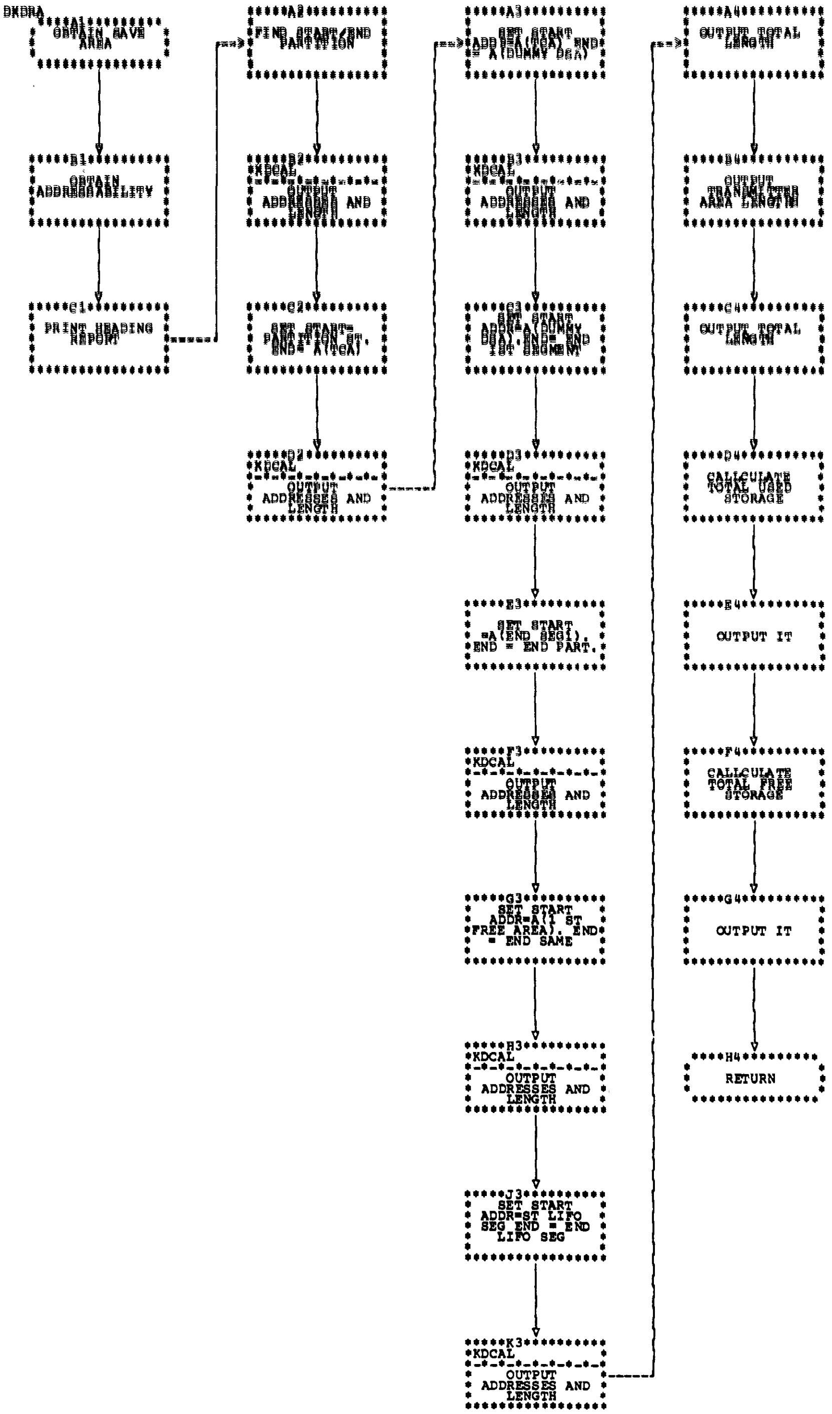

DKDRA
A1
OBTAIN SAVE AREA
B1
OBTAIN ADDRESSABILITY
C1
PRINT HEADING REPORT
A2
FIND START/END PARTITION
B2
KDCAL
OUTPUT ADDRESSES AND LENGTH
C2
SET START= PARTITION ST, END= A(TCA)
D2
KDCAL
OUTPUT ADDRESSES AND LENGTH
A3
SET START ADDR=A(TCA) END = A(DUMMY DSA)
B3
KDCAL
OUTPUT ADDRESSES AND LENGTH
C3
SET START ADDR=A(DUMMY DSA),END= END 1ST SEGMENT
D3
KDCAL
OUTPUT ADDRESSES AND LENGTH
E3
SET START =A(END SEG1), END = END PART.
F3
KDCAL
OUTPUT ADDRESSES AND LENGTH
G3
SET START ADDR=A(1 ST FREE AREA). END = END SAME
H3
KDCAL
OUTPUT ADDRESSES AND LENGTH
J3
SET START ADDR=ST LIFO SEG END = END LIFO SEG
K3
KDCAL
OUTPUT ADDRESSES AND LENGTH
A4
OUTPUT TOTAL LENGTH
B4
OUTPUT TRANSMITTER AREA LENGTH
C4
OUTPUT TOTAL LENGTH
D4
CALLCULATE TOTAL USED STORAGE
E4
OUTPUT IT
F4
CALLCULATE TOTAL FREE STORAGE
G4
OUTPUT IT
H4
RETURN

Chart DKDR. Report option (part 1 of 2)

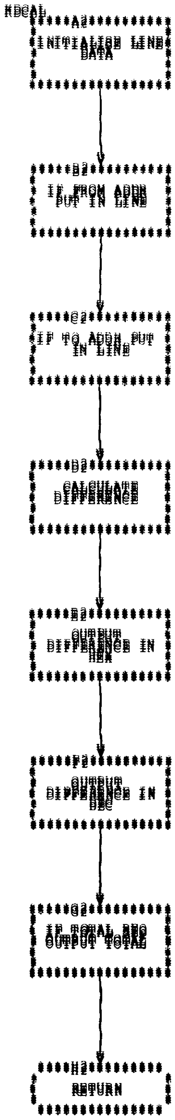


Chart DKDR. Report option (part 2 of 2)

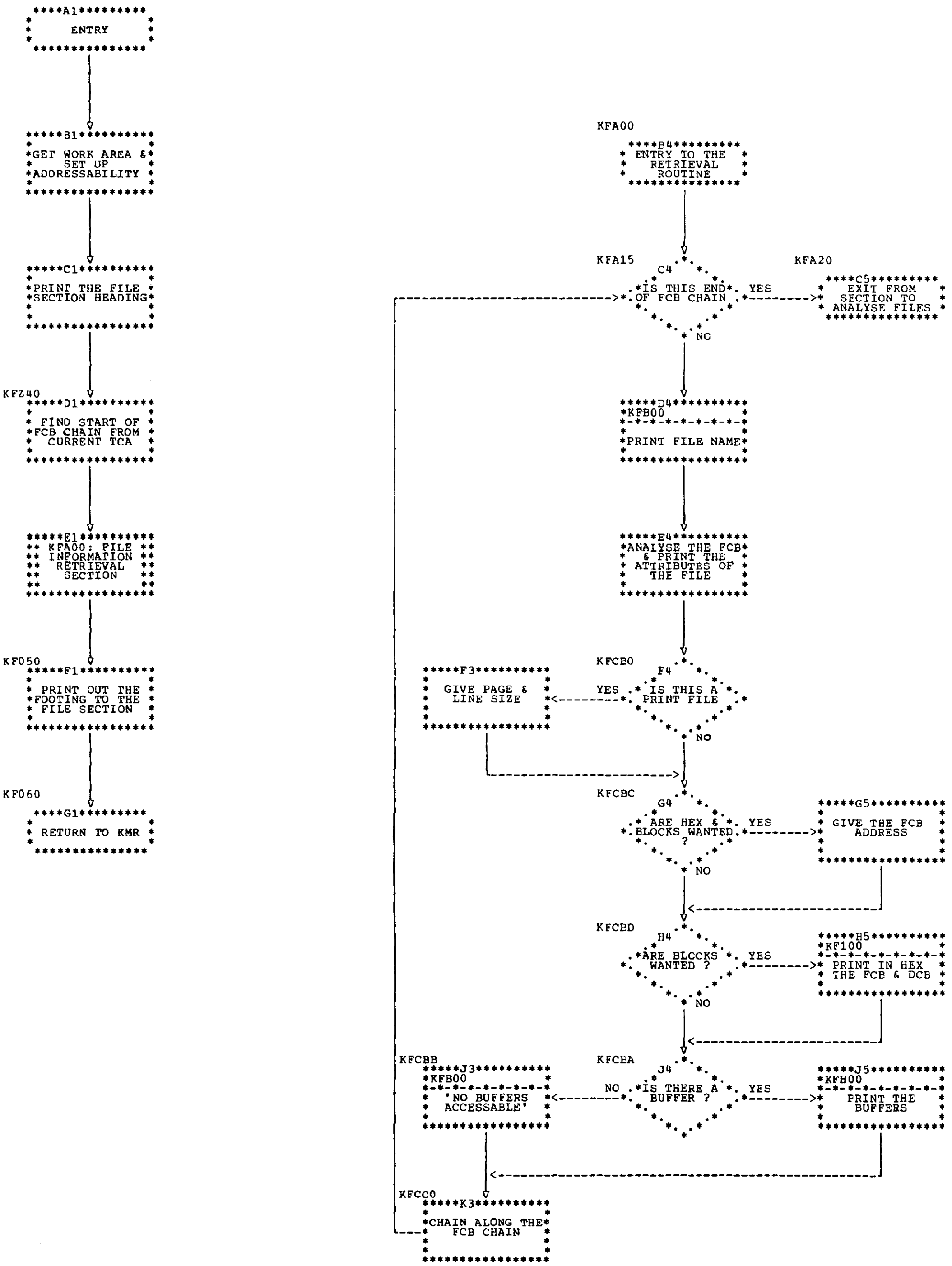

A1
ENTRY
B1
GET WORK AREA & SET UP ADDRESSABILITY
C1
PRINT THE FILE SECTION HEADING
KFZ40
D1
FIND START OF FCB CHAIN FROM CURRENT TCA
E1
KFA00: FILE INFORMATION RETRIEVAL SECTION
KF050
F1
PRINT OUT THE FOOTING TO THE FILE SECTION
KF060
G1
RETURN TO KMR
KFA00
B4
ENTRY TO THE RETRIEVAL ROUTINE
KFA15
C4
IS THIS END OF FCB CHAIN
YES
NO
KFA20
C5
EXIT FROM SECTION TO ANALYSE FILES
D4
KFB00
PRINT FILE NAME
E4
ANALYSE THE FCB & PRINT THE ATTRIBUTES OF THE FILE
KFCB0
F4
IS THIS A PRINT FILE
YES
NO
F3
GIVE PAGE & LINE SIZE
KFCBC
G4
ARE HEX & BLOCKS WANTED ?
YES
NO
G5
GIVE THE FCB ADDRESS
KFCBD
H4
ARE BLCCKS WANTED ?
YES
NO
H5
KF100
PRINT IN HEX THE FCB & DCB
KFCEA
J4
IS THERE A BUFFER ?
NO
YES
KFCBB
J3
KFB00
'NO BUFFERS ACCESSABLE'
J5
KFH00
PRINT THE BUFFERS
KFCC0
K3
CHAIN ALONG THE FCB CHAIN

Chart DKFA. Dump file attributes module (part 1 of 2)

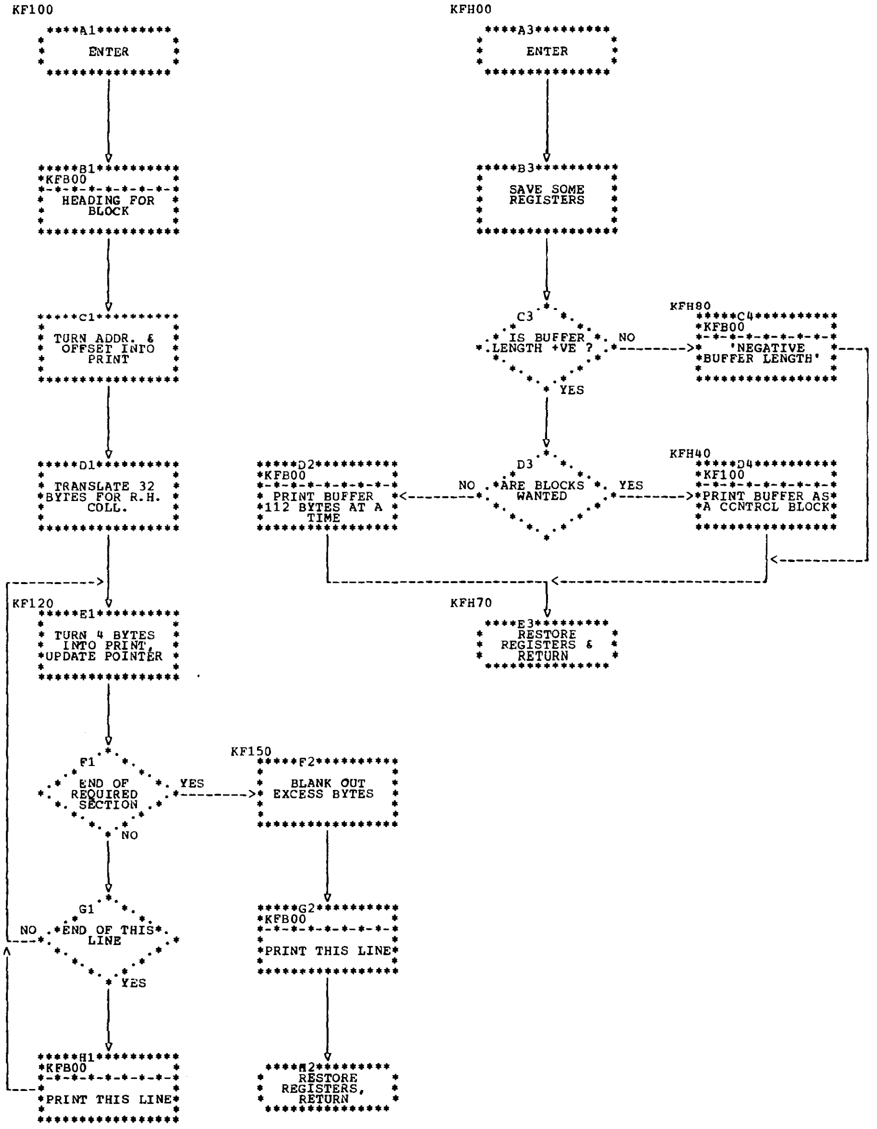


Chart DKFA. Dump file attributes module (part 2 of 2)

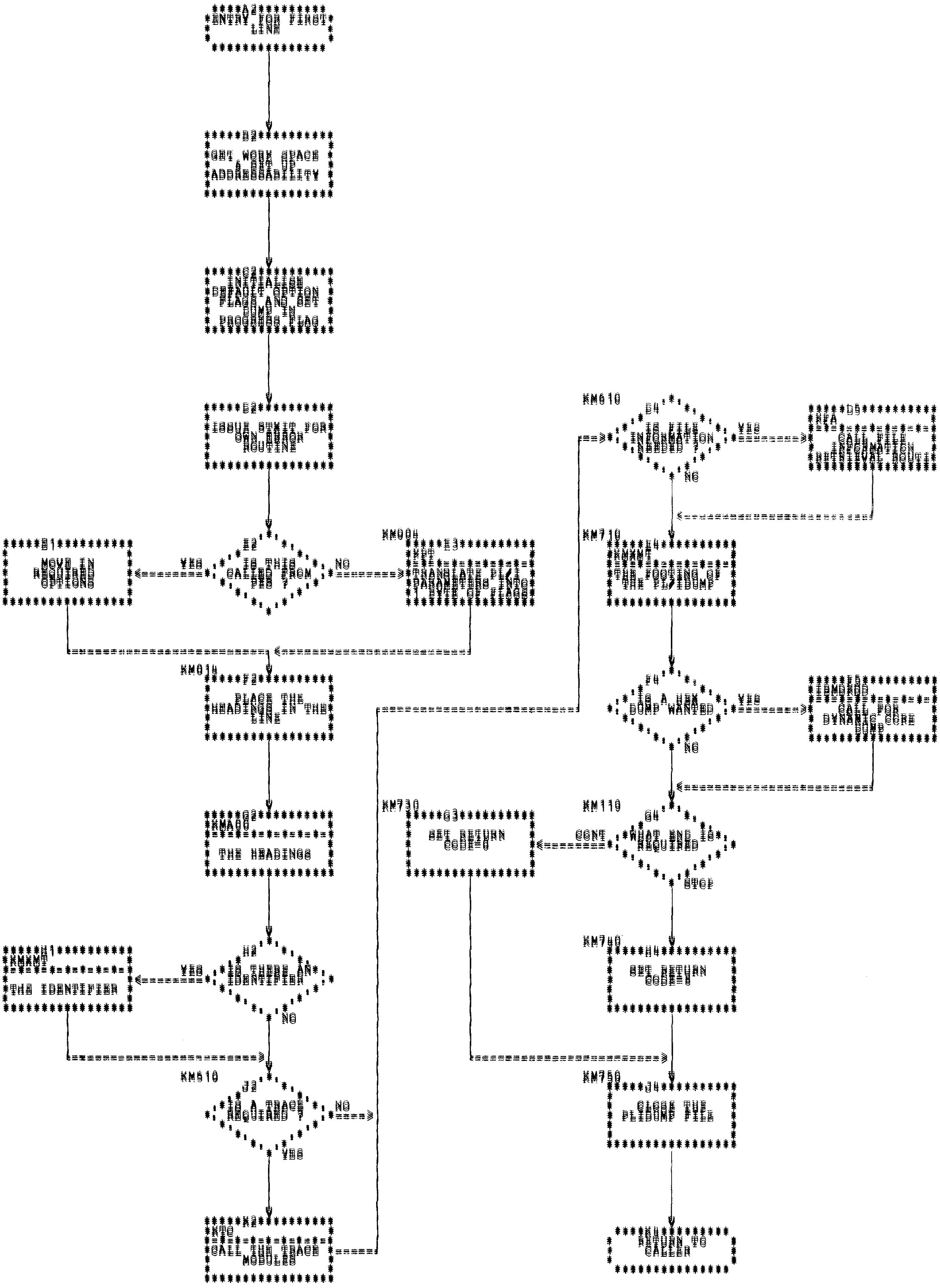


Chart DKMR. Dump control module (part 1 of 3)

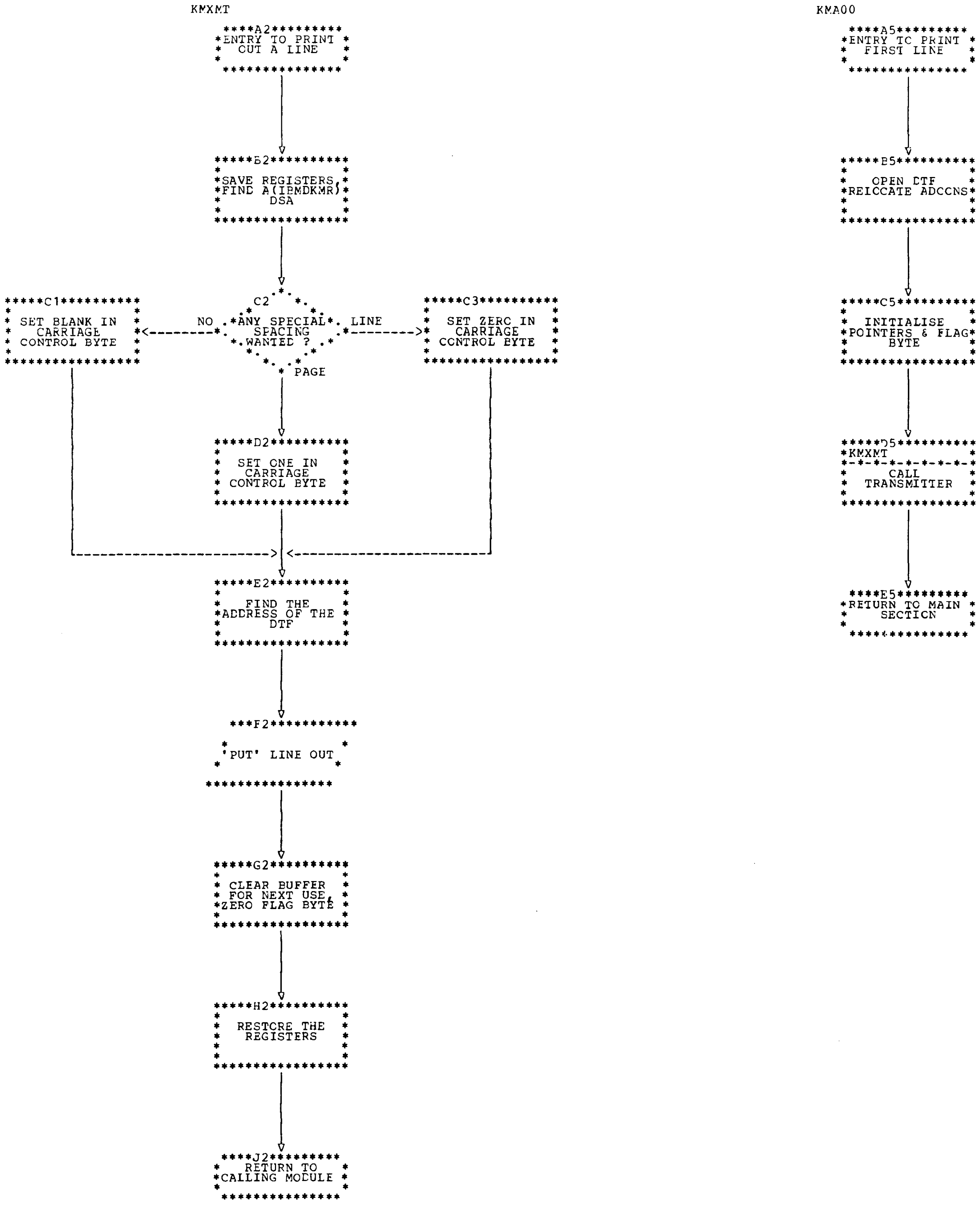

KMXMT
A2
ENTRY TO PRINT OUT A LINE
B2
SAVE REGISTERS, FIND A(IBMDKMR) DSA
C2
ANY SPECIAL SPACING WANTED ?
NO
LINE
PAGE
C1
SET BLANK IN CARRIAGE CONTROL BYTE
C3
SET ZERO IN CARRIAGE CONTROL BYTE
D2
SET ONE IN CARRIAGE CONTROL BYTE
E2
FIND THE ADDRESS OF THE DTF
F2
'PUT' LINE OUT
G2
CLEAR BUFFER FOR NEXT USE, ZERO FLAG BYTE
H2
RESTORE THE REGISTERS
J2
RETURN TO CALLING MODULE
KMA00
A5
ENTRY TO PRINT FIRST LINE
B5
OPEN DTF RELOCATE ADCONS
C5
INITIALISE POINTERS & FLAG BYTE
D5
KMXMT
CALL TRANSMITTER
E5
RETURN TO MAIN SECTION

Chart DKMR. Dump control module (part 2 of 3)

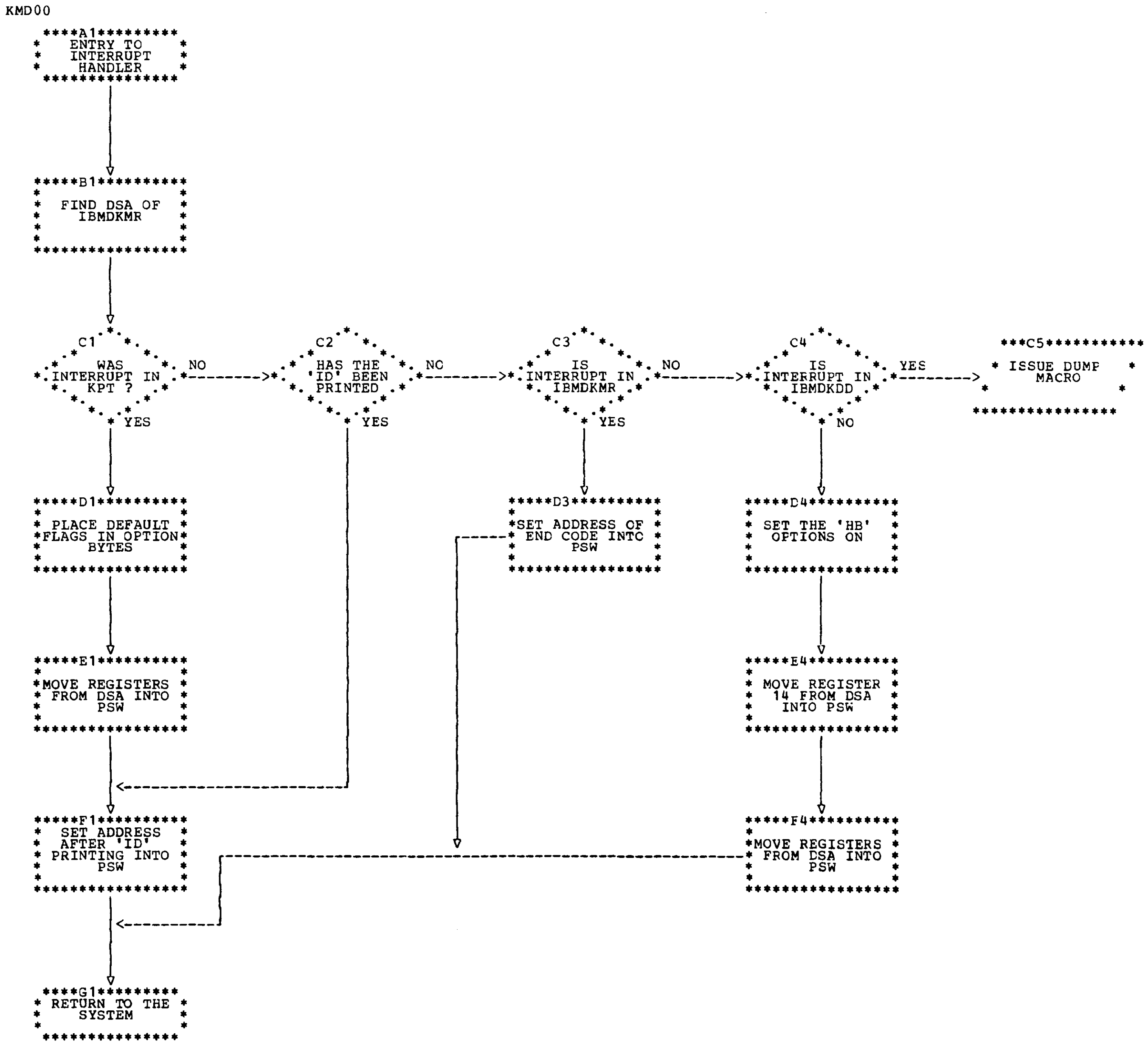

KMD00
A1 ENTRY TO INTERRUPT HANDLER
B1 FIND DSA OF IBMDKMR
C1 WAS INTERRUPT IN KPT ?
NO
YES
C2 HAS THE 'ID' BEEN PRINTED
NC
YES
C3 IS INTERRUPT IN IBMDKMR
NO
YES
C4 IS INTERRUPT IN IBMDKDD
YES
NO
C5 ISSUE DUMP MACRO
D1 PLACE DEFAULT FLAGS IN OPTION BYTES
D3 SET ADDRESS OF END CODE INTC PSW
D4 SET THE 'HB' OPTIONS ON
E1 MOVE REGISTERS FROM DSA INTO PSW
E4 MOVE REGISTER 14 FROM DSA INTO PSW
F1 SET ADDRESS AFTER 'ID' PRINTING INTO PSW
F4 MOVE REGISTERS FROM DSA INTO PSW
G1 RETURN TO THE SYSTEM

Chart DKMR. Dump control module (part 3 of 3)

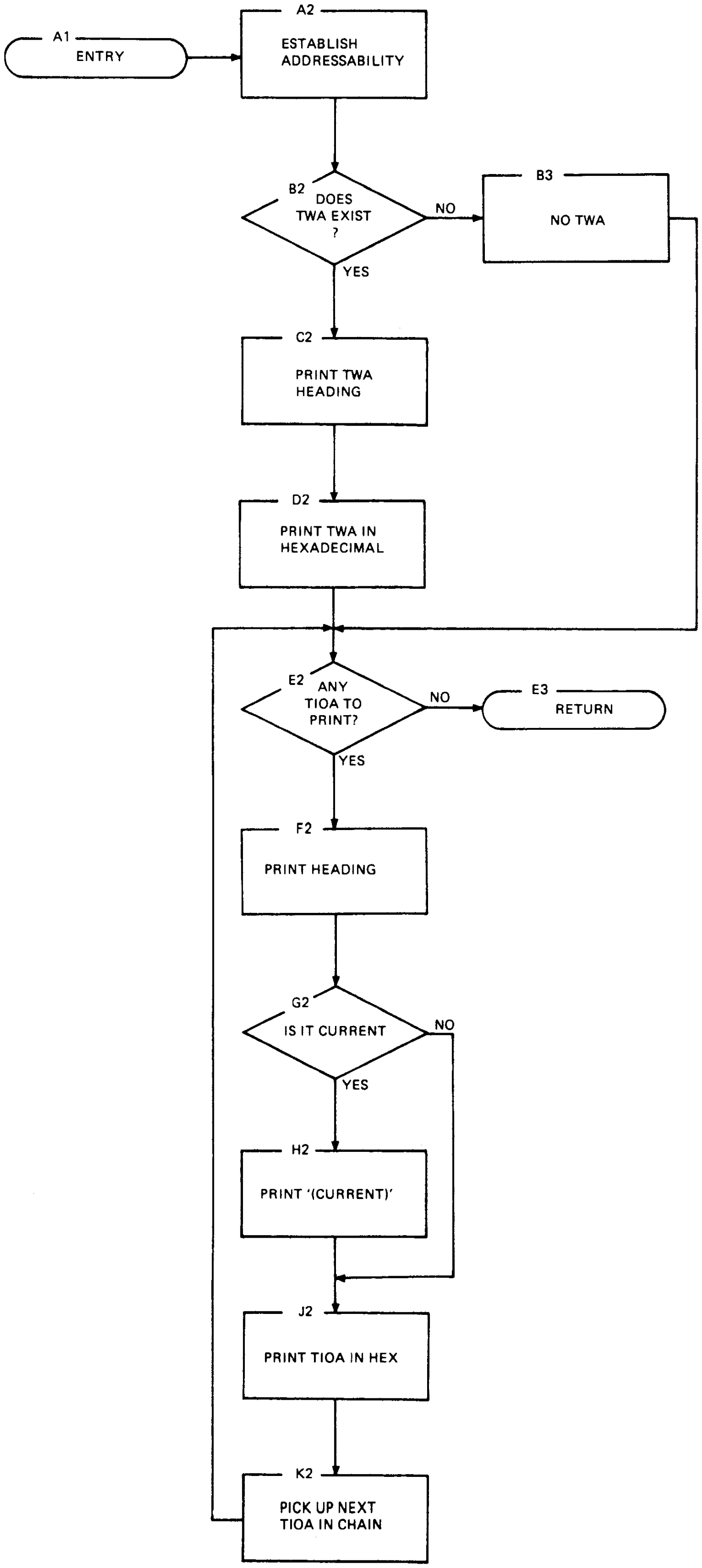
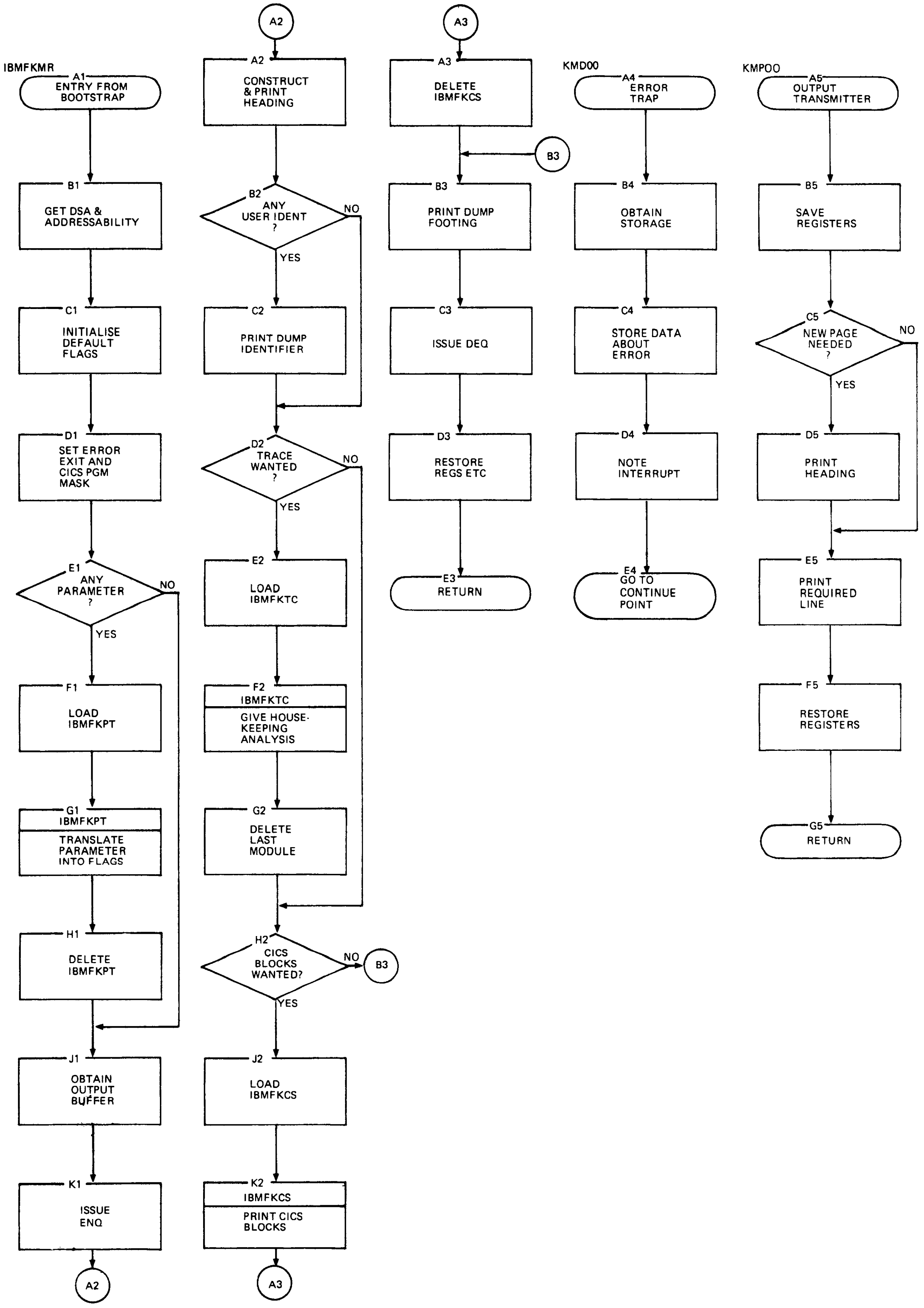
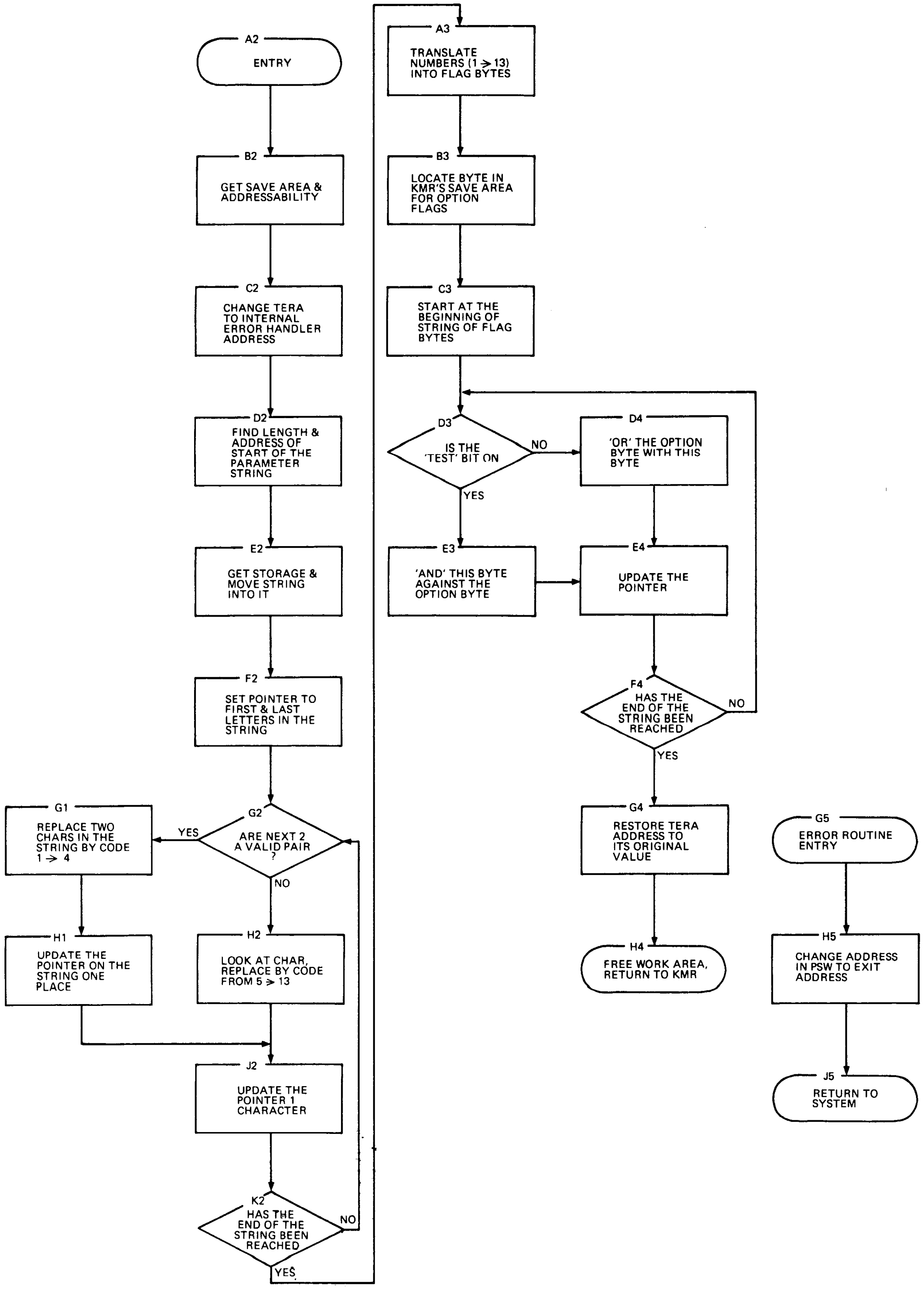


Chart FKCS. Print CICS control blocks



|Chart FKMR. Dump control module - CICS


A2
ENTRY
B2
GET SAVE AREA & ADDRESSABILITY
C2
CHANGE TERA TO INTERNAL ERROR HANDLER ADDRESS
D2
FIND LENGTH & ADDRESS OF START OF THE PARAMETER STRING
E2
GET STORAGE & MOVE STRING INTO IT
F2
SET POINTER TO FIRST & LAST LETTERS IN THE STRING
G1
REPLACE TWO CHARS IN THE STRING BY CODE 1 → 4
YES
G2
ARE NEXT 2 A VALID PAIR ?
NO
H1
UPDATE THE POINTER ON THE STRING ONE PLACE
H2
LOOK AT CHAR, REPLACE BY CODE FROM 5 → 13
J2
UPDATE THE POINTER 1 CHARACTER
K2
HAS THE END OF THE STRING BEEN REACHED
NO
YES
A3
TRANSLATE NUMBERS (1 → 13) INTO FLAG BYTES
B3
LOCATE BYTE IN KMR'S SAVE AREA FOR OPTION FLAGS
C3
START AT THE BEGINNING OF STRING OF FLAG BYTES
D3
IS THE 'TEST' BIT ON
NO
YES
D4
'OR' THE OPTION BYTE WITH THIS BYTE
E3
'AND' THIS BYTE AGAINST THE OPTION BYTE
E4
UPDATE THE POINTER
F4
HAS THE END OF THE STRING BEEN REACHED
NO
YES
G4
RESTORE TERA ADDRESS TO ITS ORIGINAL VALUE
H4
FREE WORK AREA, RETURN TO KMR
G5
ERROR ROUTINE ENTRY
H5
CHANGE ADDRESS IN PSW TO EXIT ADDRESS
J5
RETURN TO SYSTEM

| Chart FKPT. Dump parameter translate module - CICS

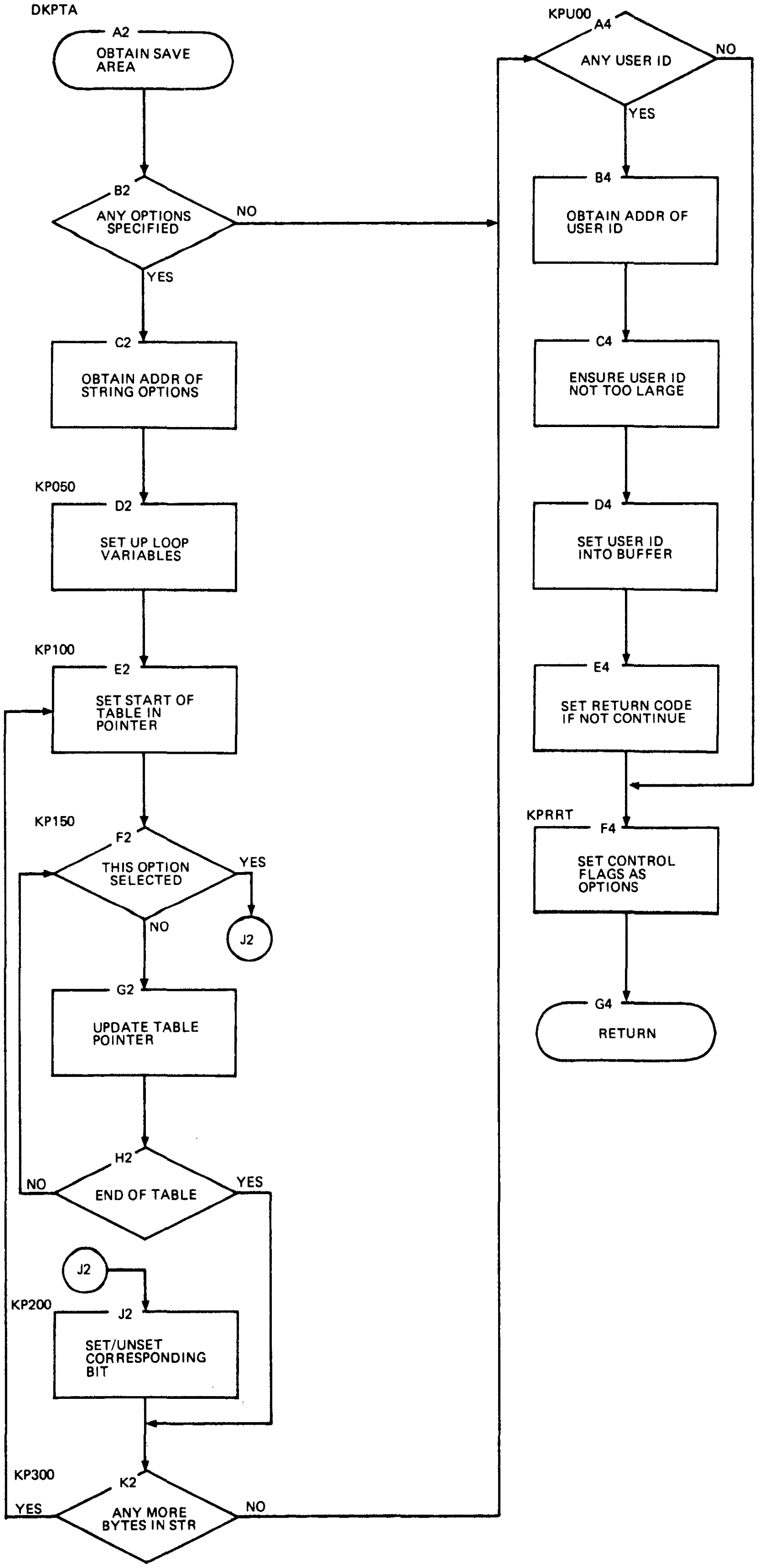


Chart DKPT. Dump parameter translate

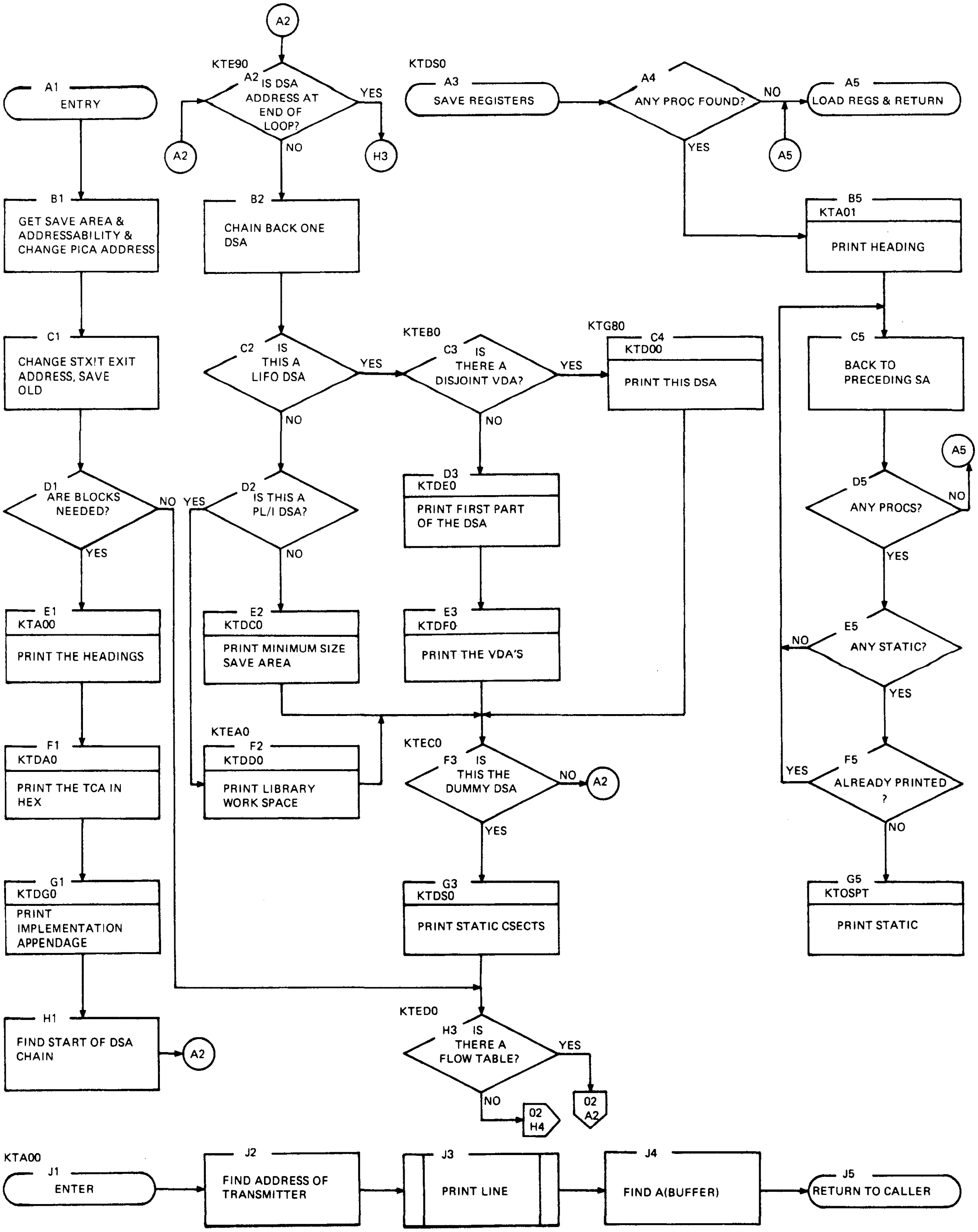


Chart DKTB. Save area control block printout (part 1 of 3)

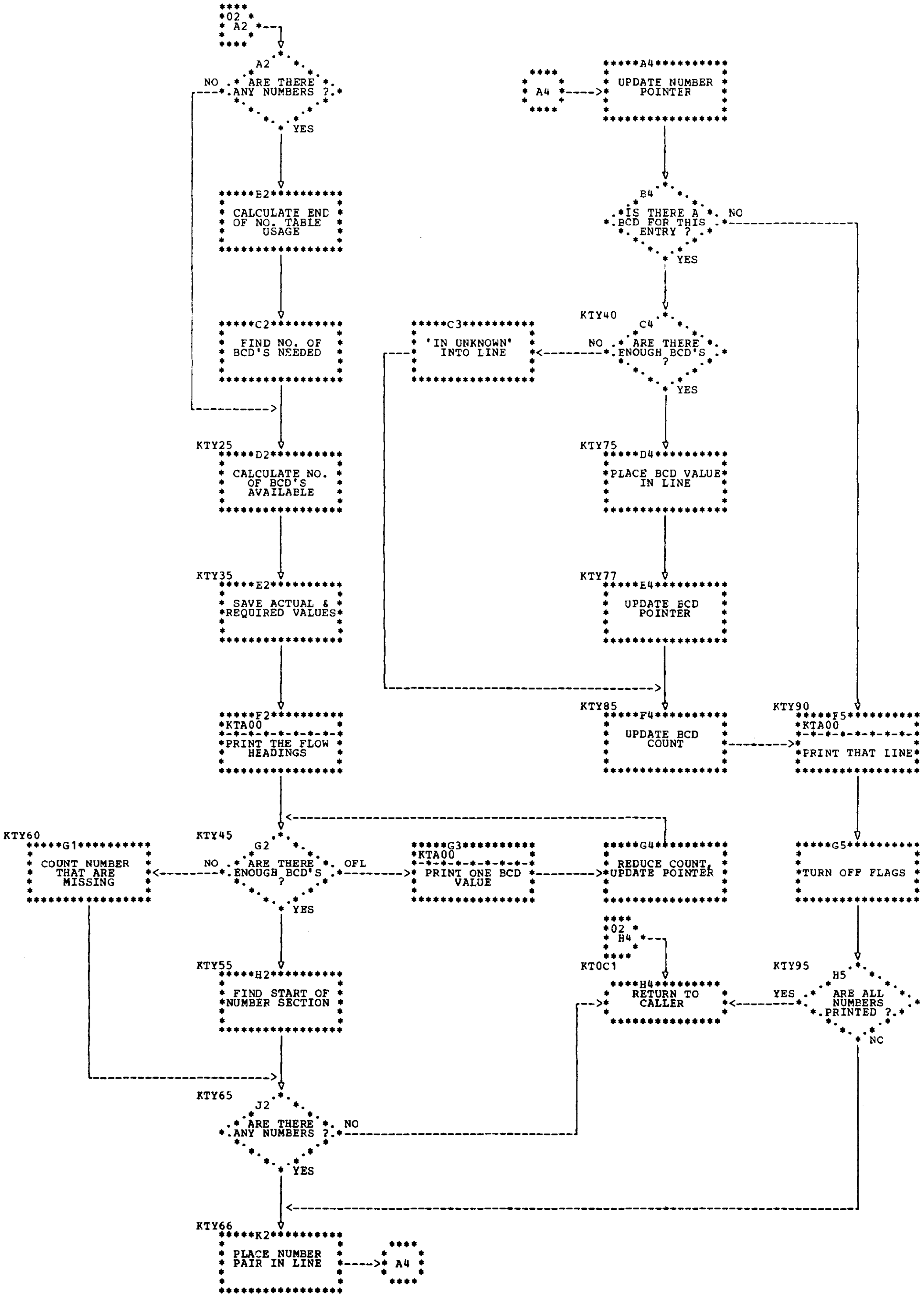


Chart DKTB. Save area control block printout (part 2 of 3)

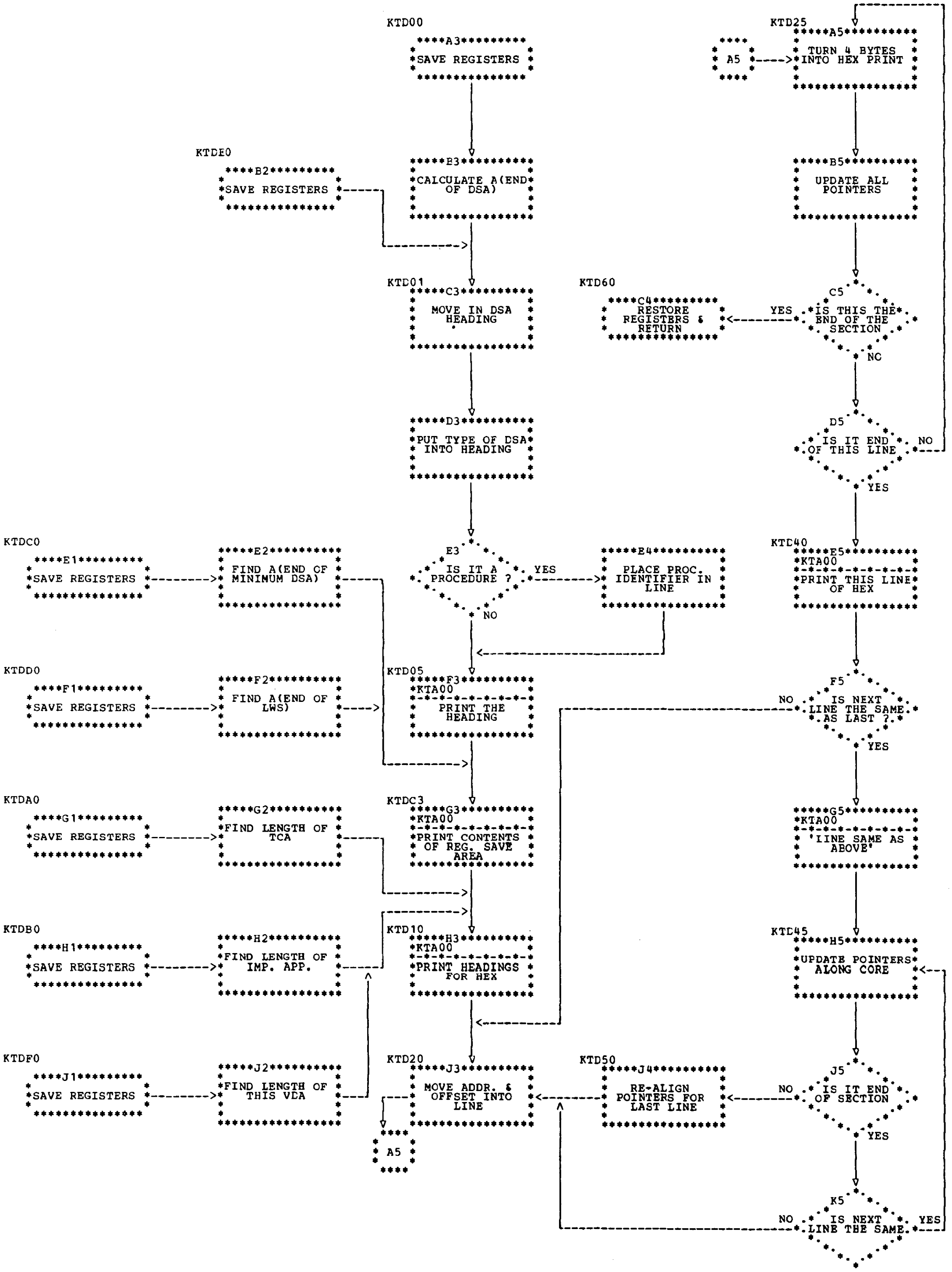
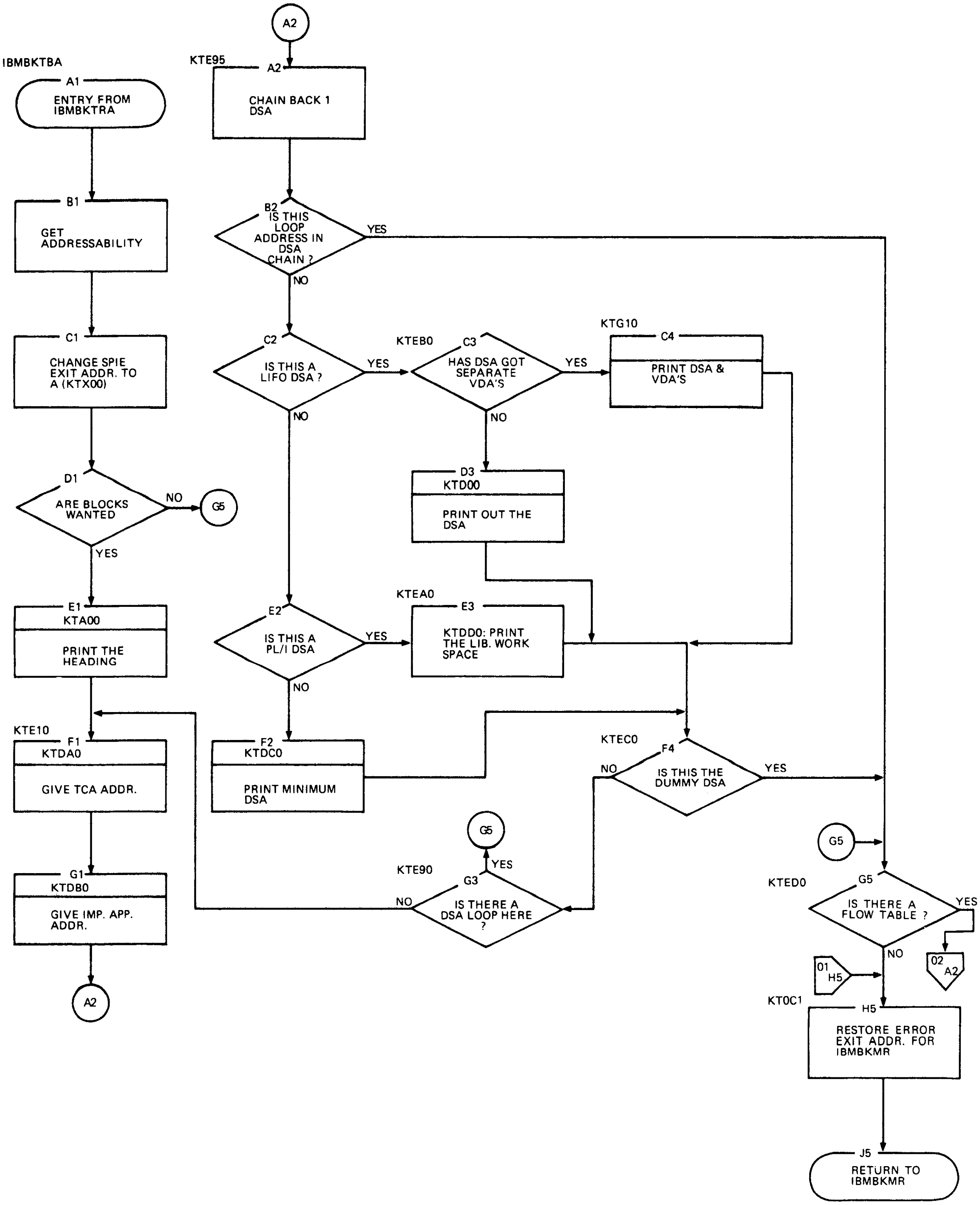
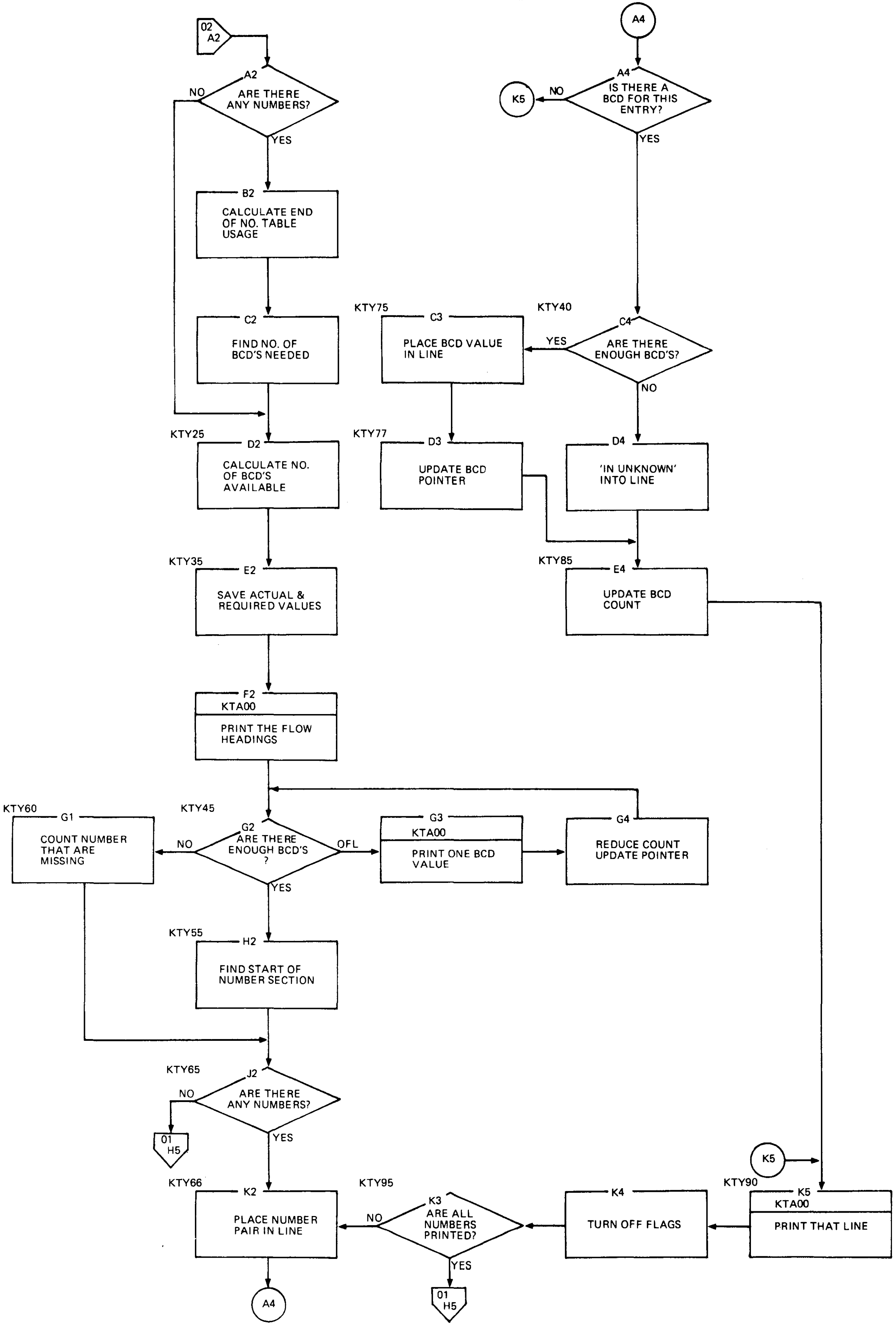


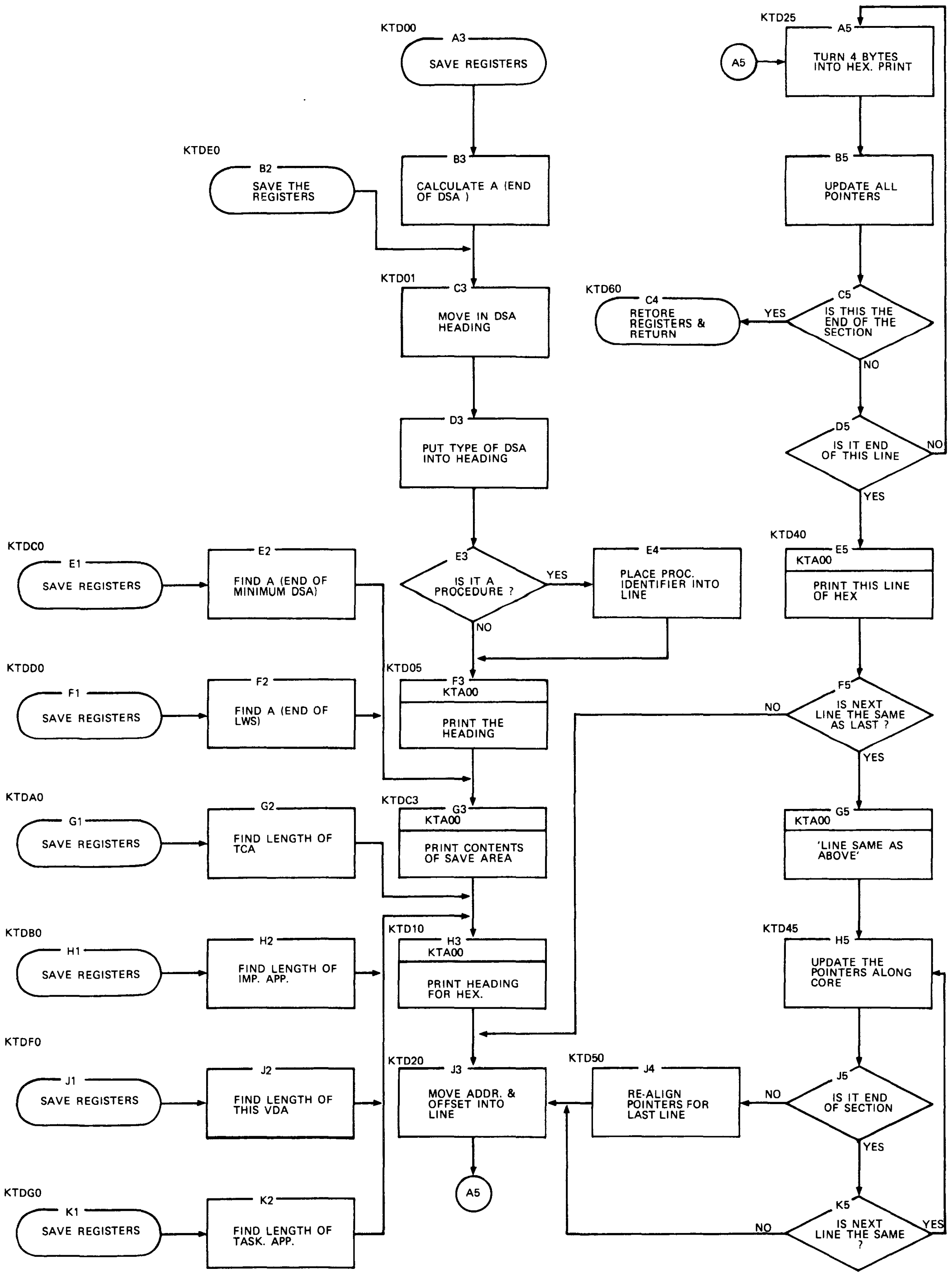
Chart DKTB. Save area control block printout (part 3 of 3)



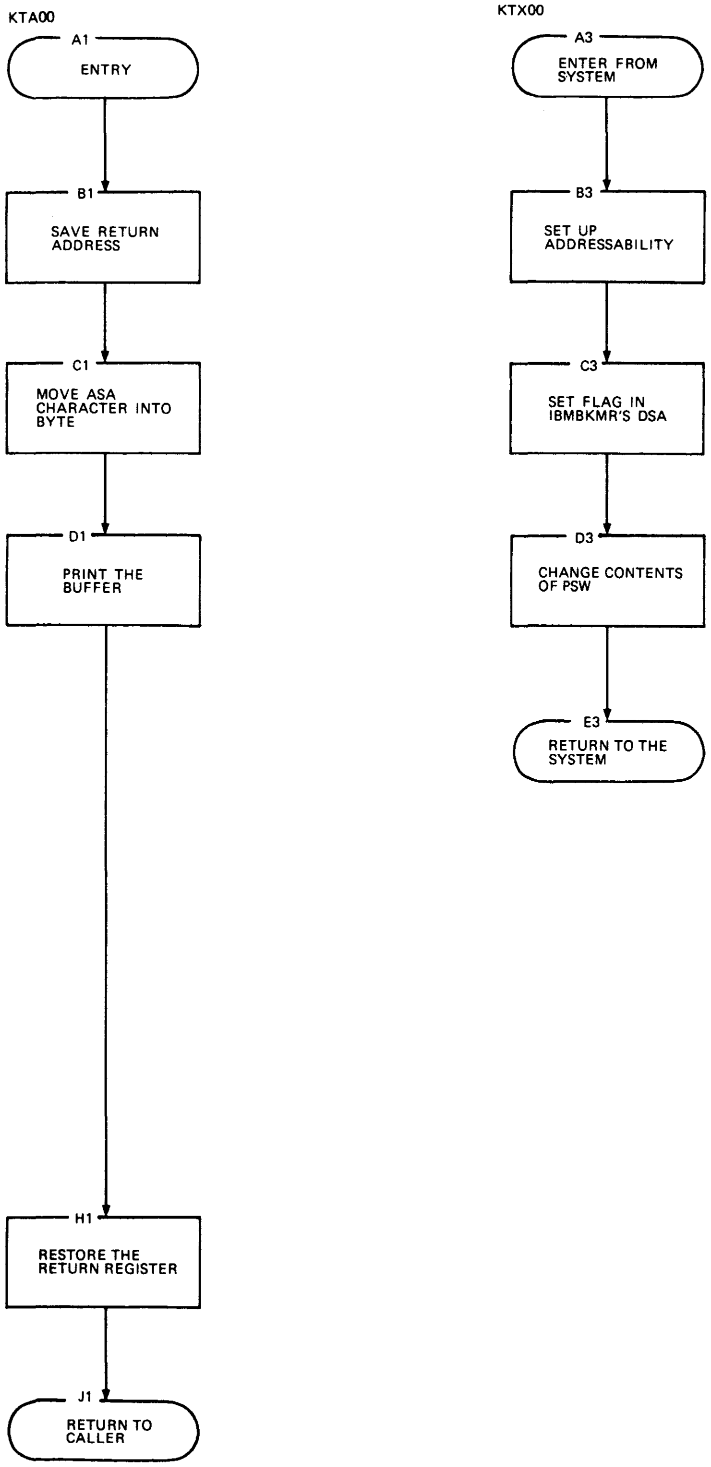
|Chart FKTB. Save area control block printout - CICS (part 1 of 4)



|Chart FKTB. Save area control block printout - CICS (part 2 of 4)



|Chart FKTB. Save area control block printout - CICS (part 3 of 4)



|Chart FKTB. Save area control block printout - CICS (part 4 of 4)

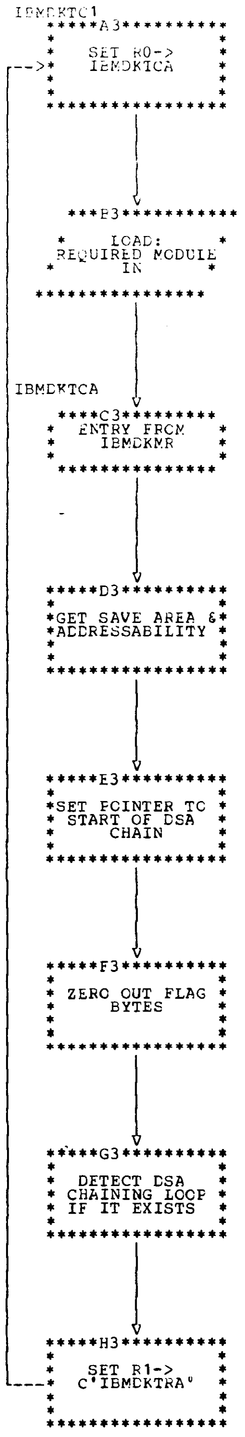


Chart DKTC. Save area chain validity checker

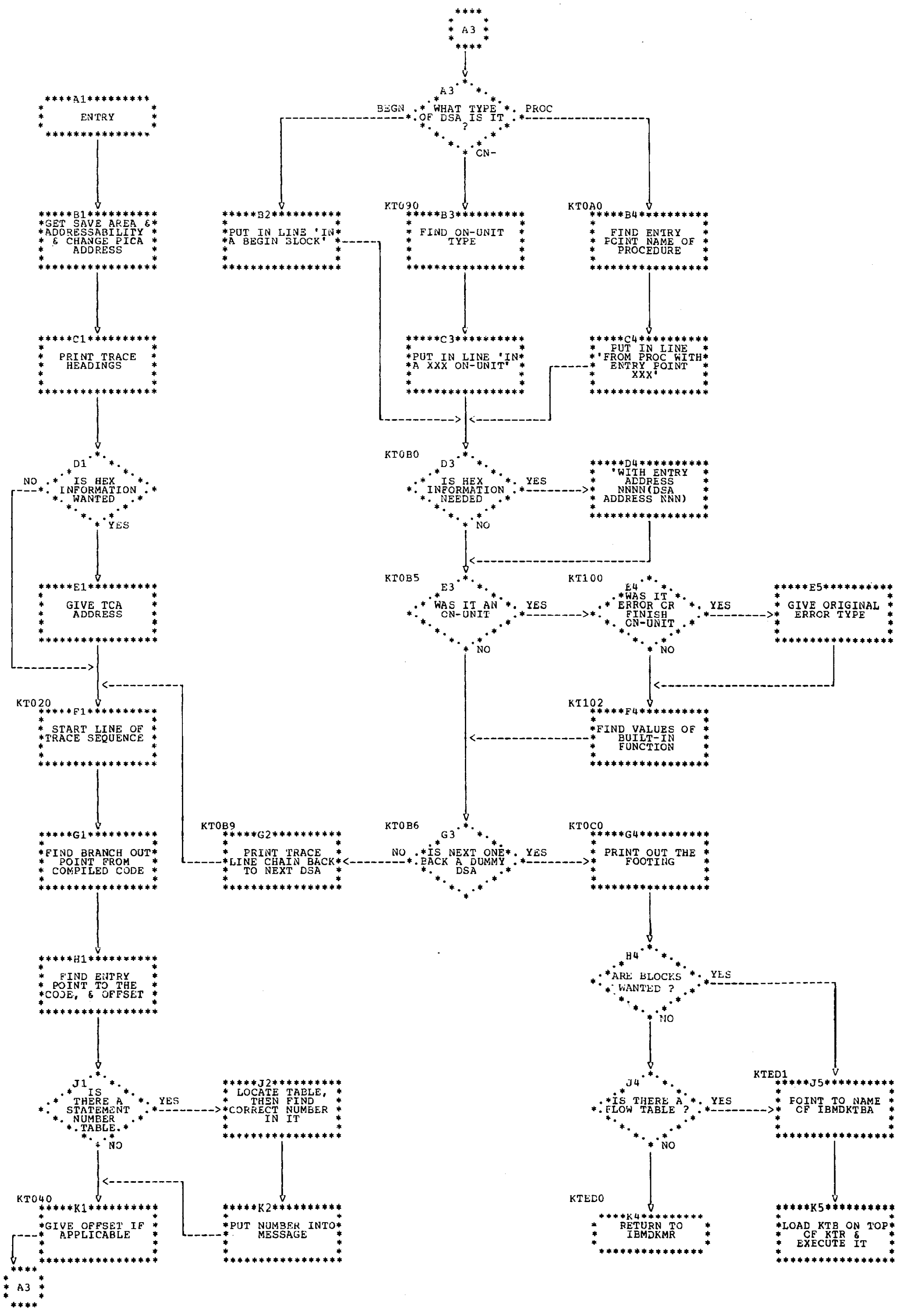

A3
A3
WHAT TYPE OF DSA IS IT ?
BEGN
PROC
ON-
A1
ENTRY
B1
GET SAVE AREA & ADDRESSABILITY & CHANGE PICA ADDRESS
B2
PUT IN LINE 'IN A BEGIN BLOCK'
KT090
B3
FIND ON-UNIT TYPE
KT0A0
B4
FIND ENTRY POINT NAME OF PROCEDURE
C1
PRINT TRACE HEADINGS
C3
PUT IN LINE 'IN A XXX ON-UNIT'
C4
PUT IN LINE 'FROM PROC WITH ENTRY POINT XXX'
D1
IS HEX INFORMATION WANTED
NO
YES
KT0B0
D3
IS HEX INFORMATION NEEDED
YES
NO
D4
'WITH ENTRY ADDRESS NNNN(DSA ADDRESS NNN)
E1
GIVE TCA ADDRESS
KT0B5
E3
WAS IT AN ON-UNIT
YES
NO
KT100
E4
WAS IT ERROR OR FINISH ON-UNIT
YES
NO
E5
GIVE ORIGINAL ERROR TYPE
KT020
F1
START LINE OF TRACE SEQUENCE
KT102
F4
FIND VALUES OF BUILT-IN FUNCTION
G1
FIND BRANCH OUT POINT FROM COMPILED CODE
KT0B9
G2
PRINT TRACE LINE CHAIN BACK TO NEXT DSA
KT0B6
G3
IS NEXT ONE BACK A DUMMY DSA
NO
YES
KT0C0
G4
PRINT OUT THE FOOTING
H1
FIND ENTRY POINT TO THE CODE, & OFFSET
H4
ARE BLOCKS WANTED ?
YES
NO
J1
IS THERE A STATEMENT NUMBER TABLE.
YES
NO
J2
LOCATE TABLE, THEN FIND CORRECT NUMBER IN IT
J4
IS THERE A FLOW TABLE ?
YES
NO
KTED1
J5
POINT TO NAME OF IBMDKTBA
KT040
K1
GIVE OFFSET IF APPLICABLE
K2
PUT NUMBER INTO MESSAGE
KTED0
K4
RETURN TO IBMDKMR
K5
LOAD KTB ON TOP OF KTR & EXECUTE IT
A3

Chart DKTR. Dump trace module

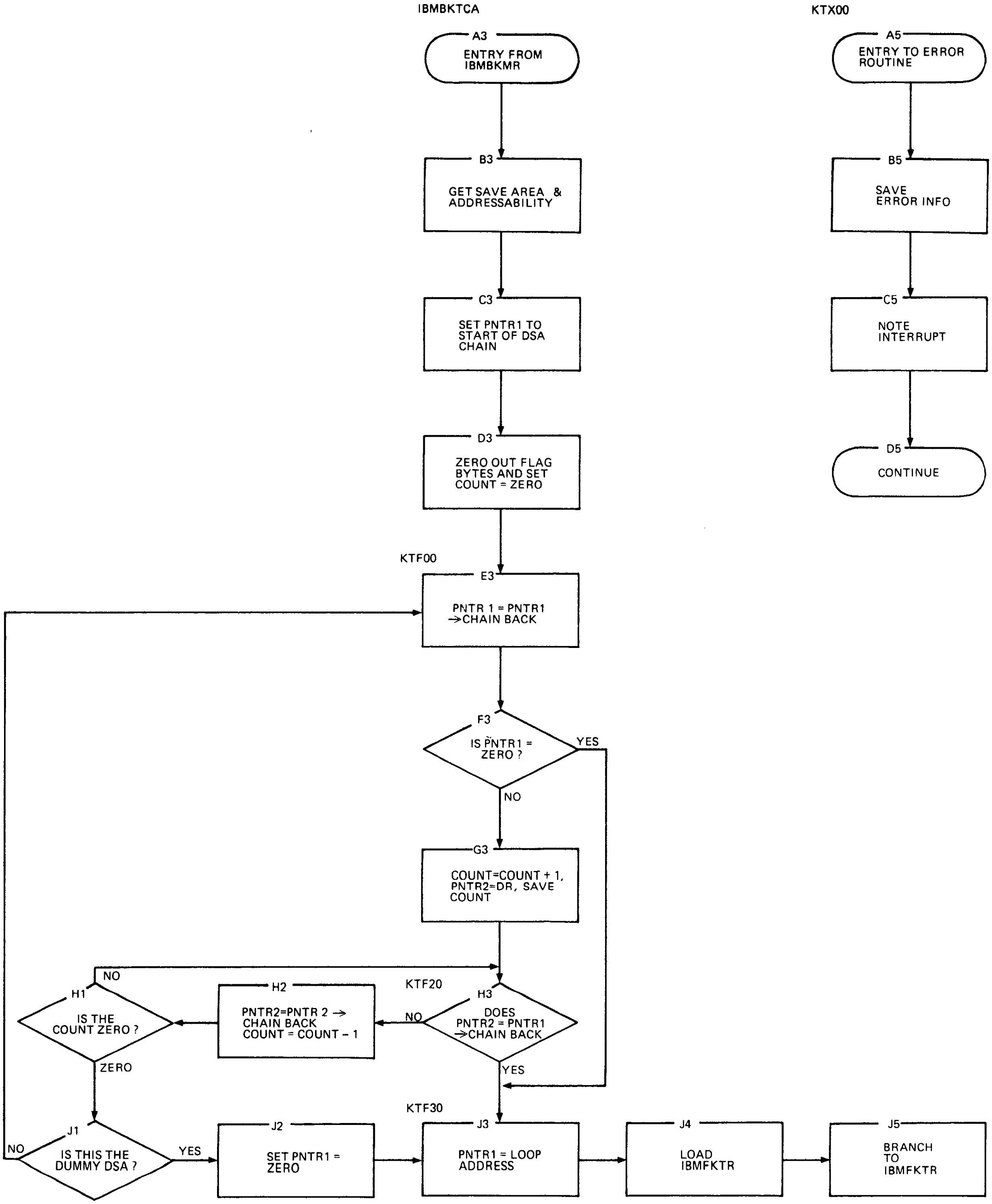


Chart FKTC. Trace check module - CICS

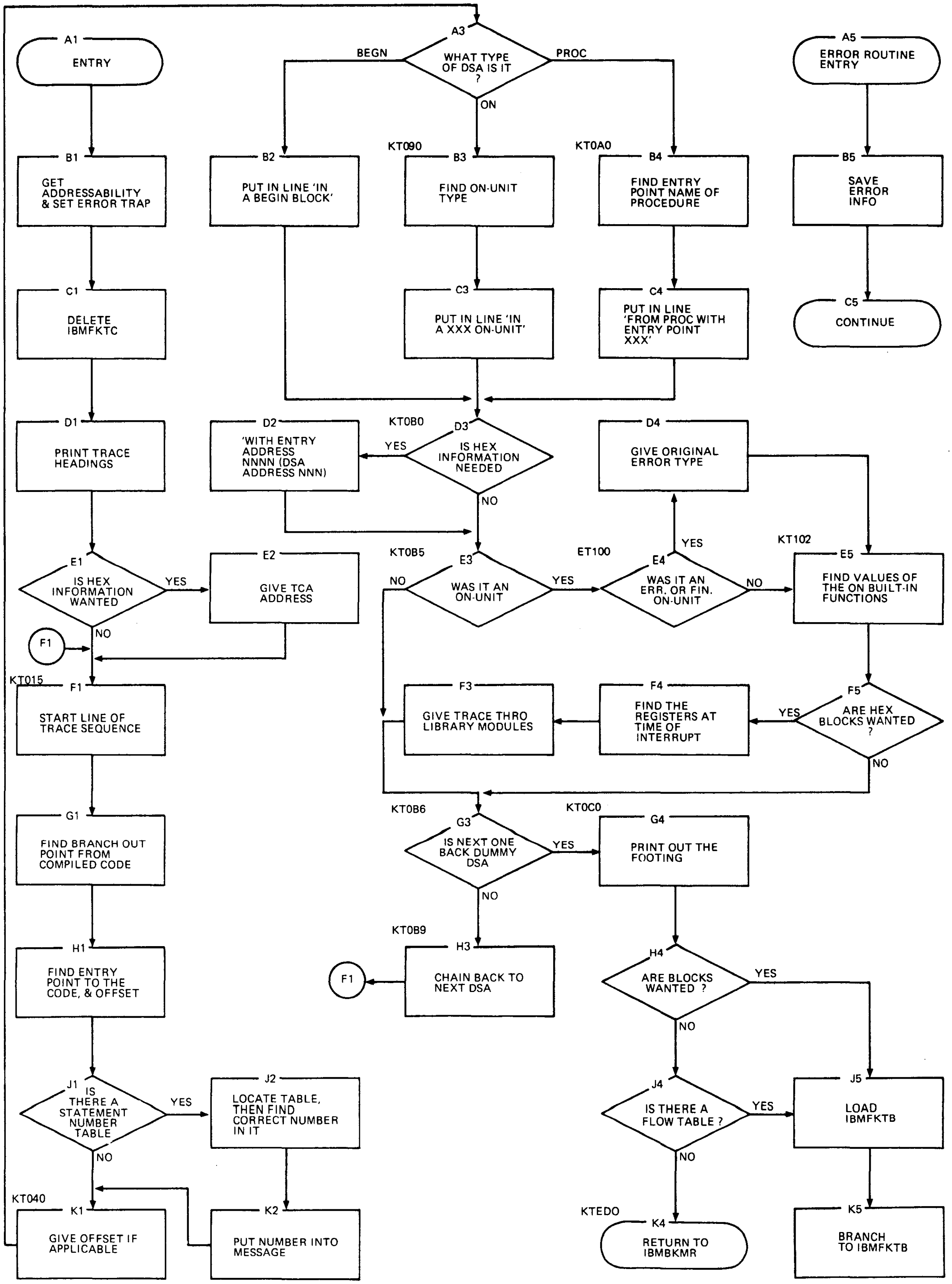

A1 ENTRY
B1 GET ADDRESSABILITY & SET ERROR TRAP
C1 DELETE IBMFKTC
D1 PRINT TRACE HEADINGS
E1 IS HEX INFORMATION WANTED
YES
NO
E2 GIVE TCA ADDRESS
F1
KT015
F1 START LINE OF TRACE SEQUENCE
G1 FIND BRANCH OUT POINT FROM COMPILED CODE
H1 FIND ENTRY POINT TO THE CODE, & OFFSET
J1 IS THERE A STATEMENT NUMBER TABLE
YES
NO
J2 LOCATE TABLE, THEN FIND CORRECT NUMBER IN IT
KT040
K1 GIVE OFFSET IF APPLICABLE
K2 PUT NUMBER INTO MESSAGE
A3 WHAT TYPE OF DSA IS IT ?
BEGN
PROC
ON
B2 PUT IN LINE 'IN A BEGIN BLOCK'
KT090
B3 FIND ON-UNIT TYPE
C3 PUT IN LINE 'IN A XXX ON-UNIT'
KT0A0
B4 FIND ENTRY POINT NAME OF PROCEDURE
C4 PUT IN LINE 'FROM PROC WITH ENTRY POINT XXX'
D2 'WITH ENTRY ADDRESS NNNN (DSA ADDRESS NNN)
KT0B0
D3 IS HEX INFORMATION NEEDED
YES
NO
D4 GIVE ORIGINAL ERROR TYPE
KT0B5
E3 WAS IT AN ON-UNIT
NO
YES
ET100
E4 WAS IT AN ERR. OR FIN. ON-UNIT
YES
NO
KT102
E5 FIND VALUES OF THE ON BUILT-IN FUNCTIONS
F3 GIVE TRACE THRO LIBRARY MODULES
F4 FIND THE REGISTERS AT TIME OF INTERRUPT
F5 ARE HEX BLOCKS WANTED ?
YES
NO
KT0B6
G3 IS NEXT ONE BACK DUMMY DSA
YES
NO
KT0C0
G4 PRINT OUT THE FOOTING
KT0B9
H3 CHAIN BACK TO NEXT DSA
F1
H4 ARE BLOCKS WANTED ?
YES
NO
J4 IS THERE A FLOW TABLE ?
YES
NO
KTED0
K4 RETURN TO IBMBKMR
A5 ERROR ROUTINE ENTRY
B5 SAVE ERROR INFO
C5 CONTINUE
J5 LOAD IBMFKTB
K5 BRANCH TO IBMFKTB

Chart FKTR. Dump trace module - CICS

]

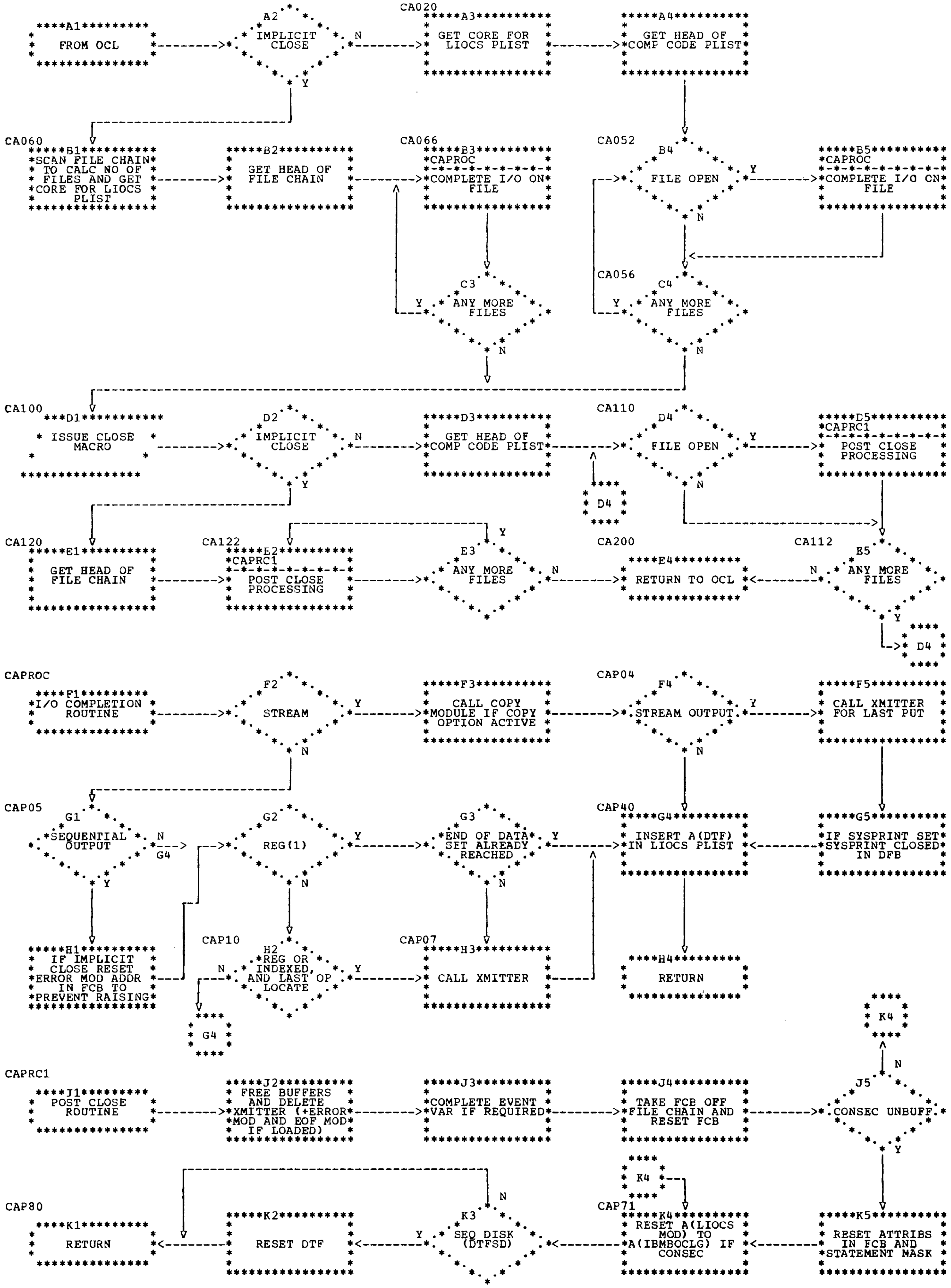


Chart DOCA. Close module

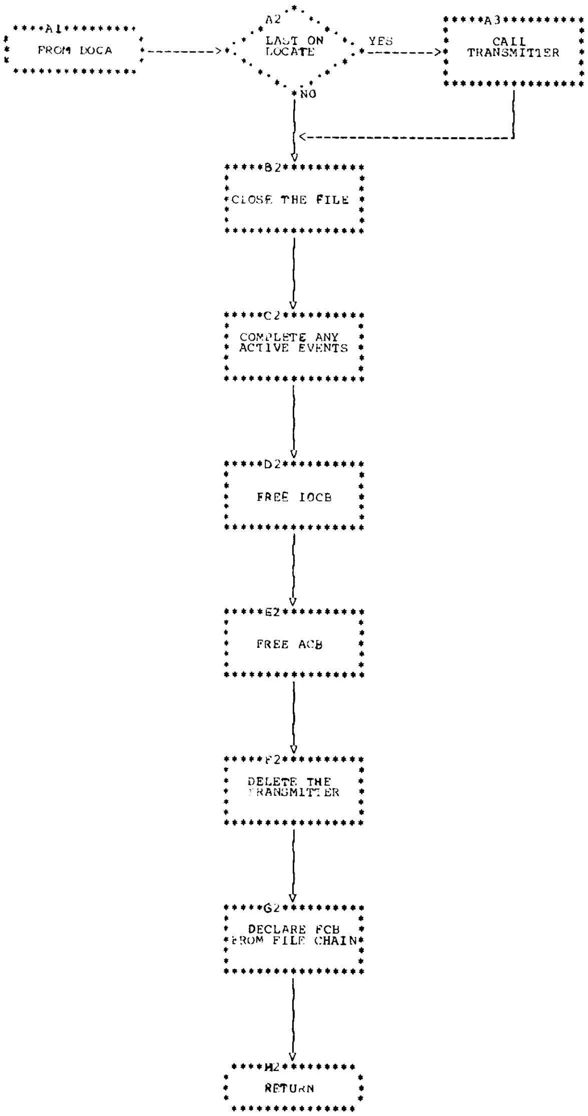


Chart DOCV. Close (VSAM)

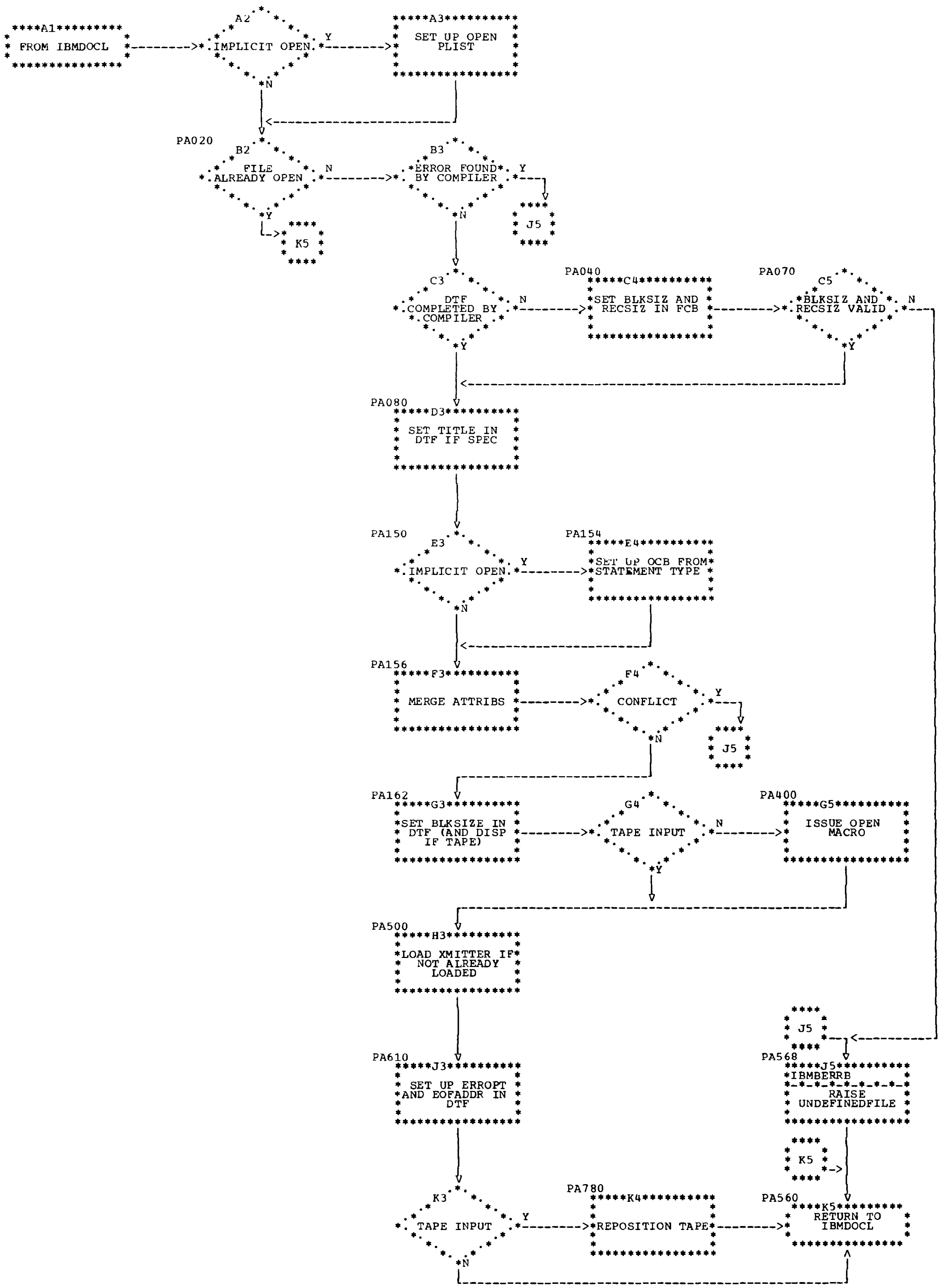


Chart DOPM. Open (Consecutive unbuffered files)

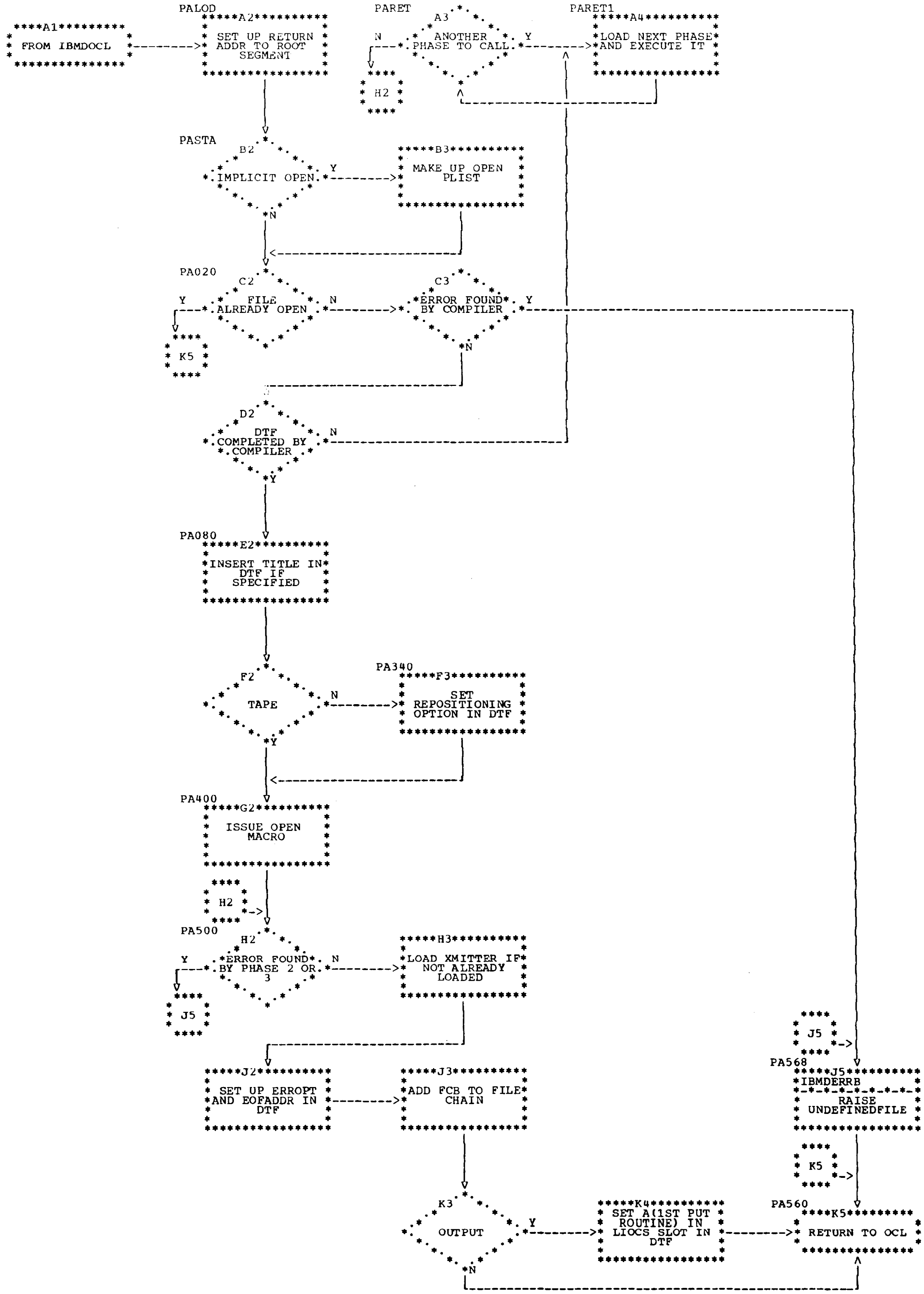


Chart DOPP. Open (Consecutive buffered files)

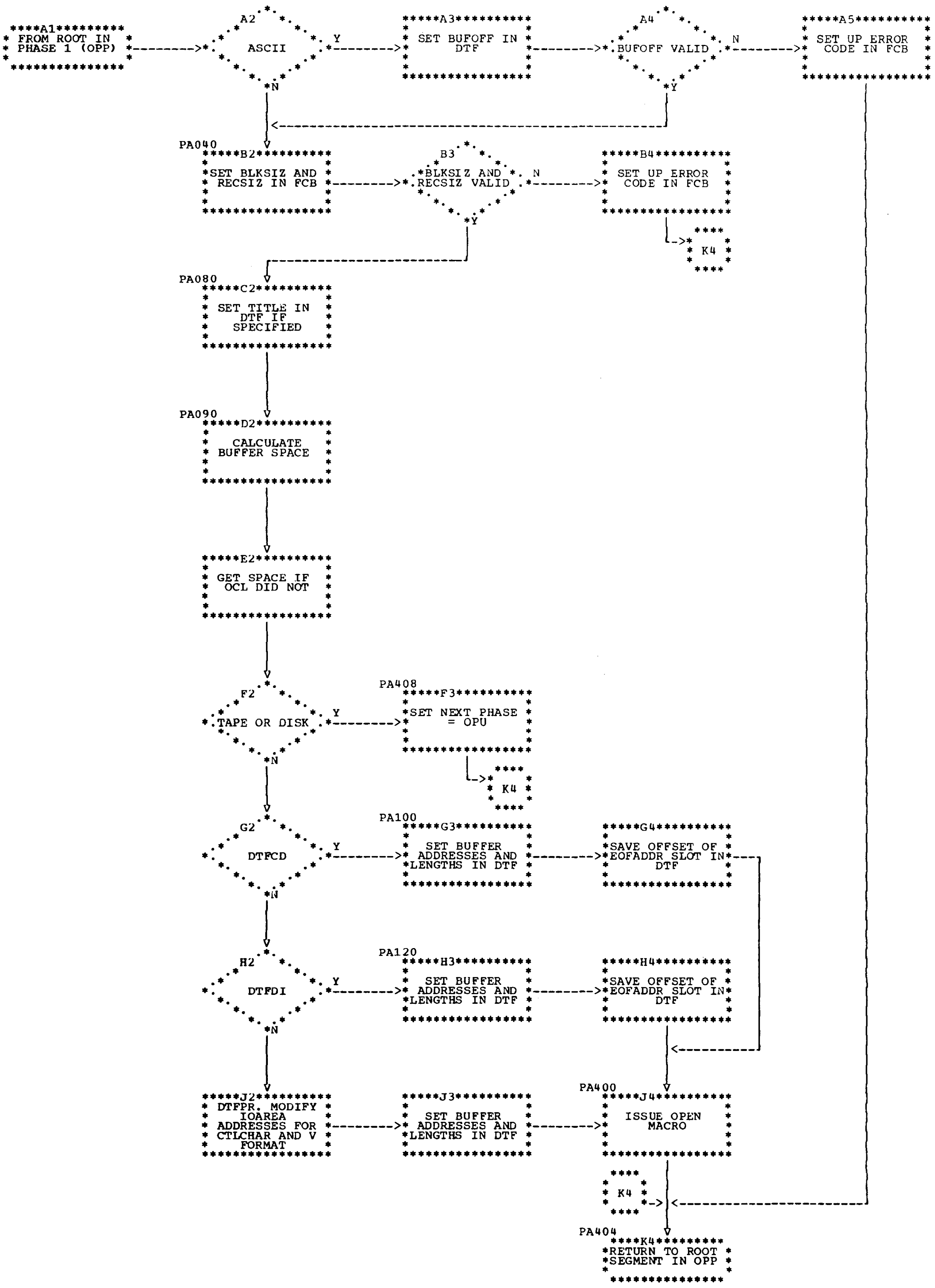

A1
FROM ROOT IN PHASE 1 (OPP)
A2
ASCII
Y
N
A3
SET BUFOFF IN DTF
A4
BUFOFF VALID
N
Y
A5
SET UP ERROR CODE IN FCB
PA040
B2
SET BLKSIZ AND RECSIZ IN FCB
B3
BLKSIZ AND RECSIZ VALID
N
Y
B4
SET UP ERROR CODE IN FCB
K4
PA080
C2
SET TITLE IN DTF IF SPECIFIED
PA090
D2
CALCULATE BUFFER SPACE
E2
GET SPACE IF OCL DID NOT
F2
TAPE OR DISK
Y
N
PA408
F3
SET NEXT PHASE = OPU
K4
G2
DTFCD
Y
N
PA100
G3
SET BUFFER ADDRESSES AND LENGTHS IN DTF
G4
SAVE OFFSET OF EOFADDR SLOT IN DTF
H2
DTFDI
Y
N
PA120
H3
SET BUFFER ADDRESSES AND LENGTHS IN DTF
H4
SAVE OFFSET OF EOFADDR SLOT IN DTF
J2
DTFPR. MODIFY IOAREA ADDRESSES FOR CTLCHAR AND V FORMAT
J3
SET BUFFER ADDRESSES AND LENGTHS IN DTF
PA400
J4
ISSUE OPEN MACRO
K4
PA404
K4
RETURN TO ROOT SEGMENT IN OPP

Chart DOPQ. Open (Consecutive buffered files) - level 2

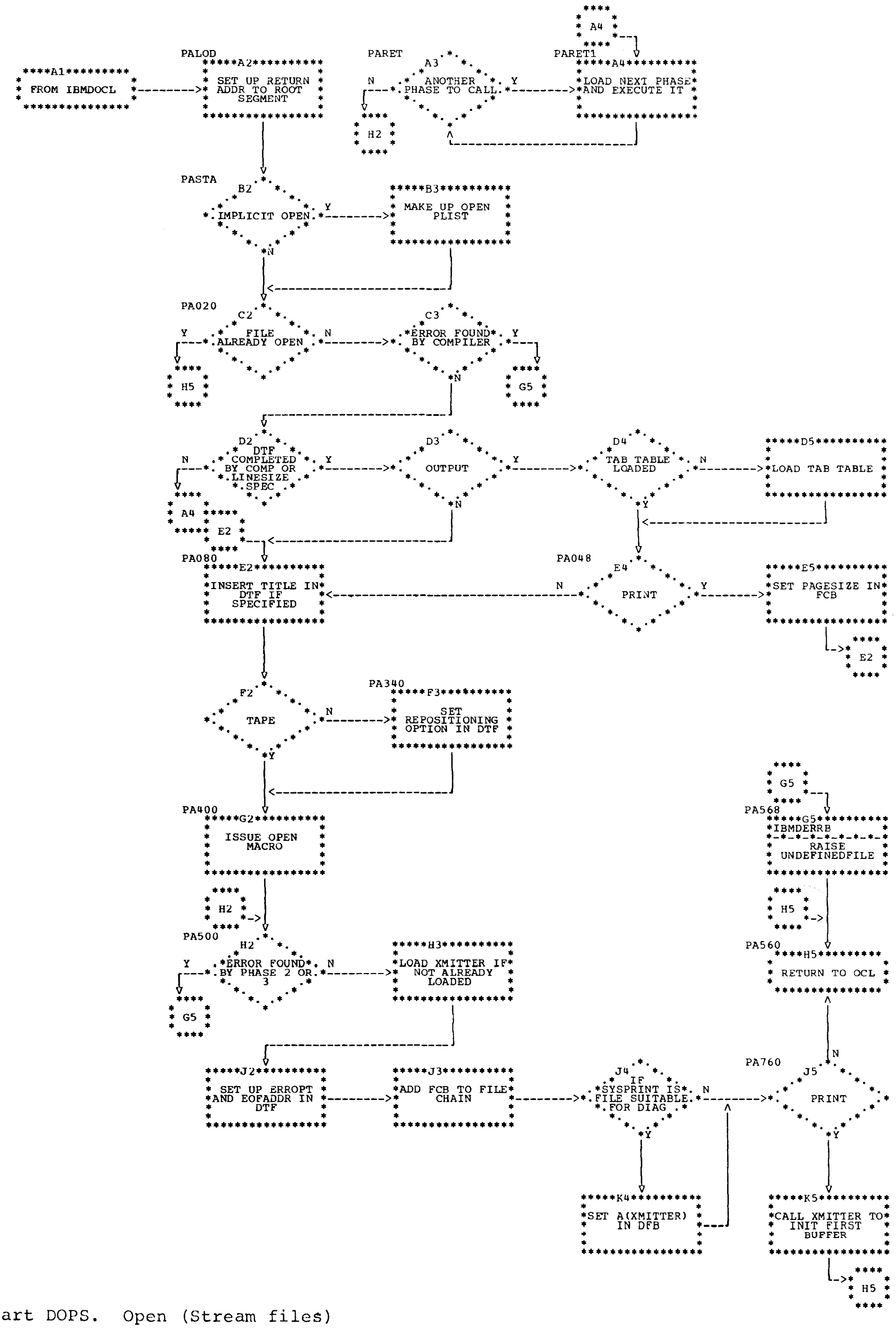


Chart DOPS. Open (Stream files)

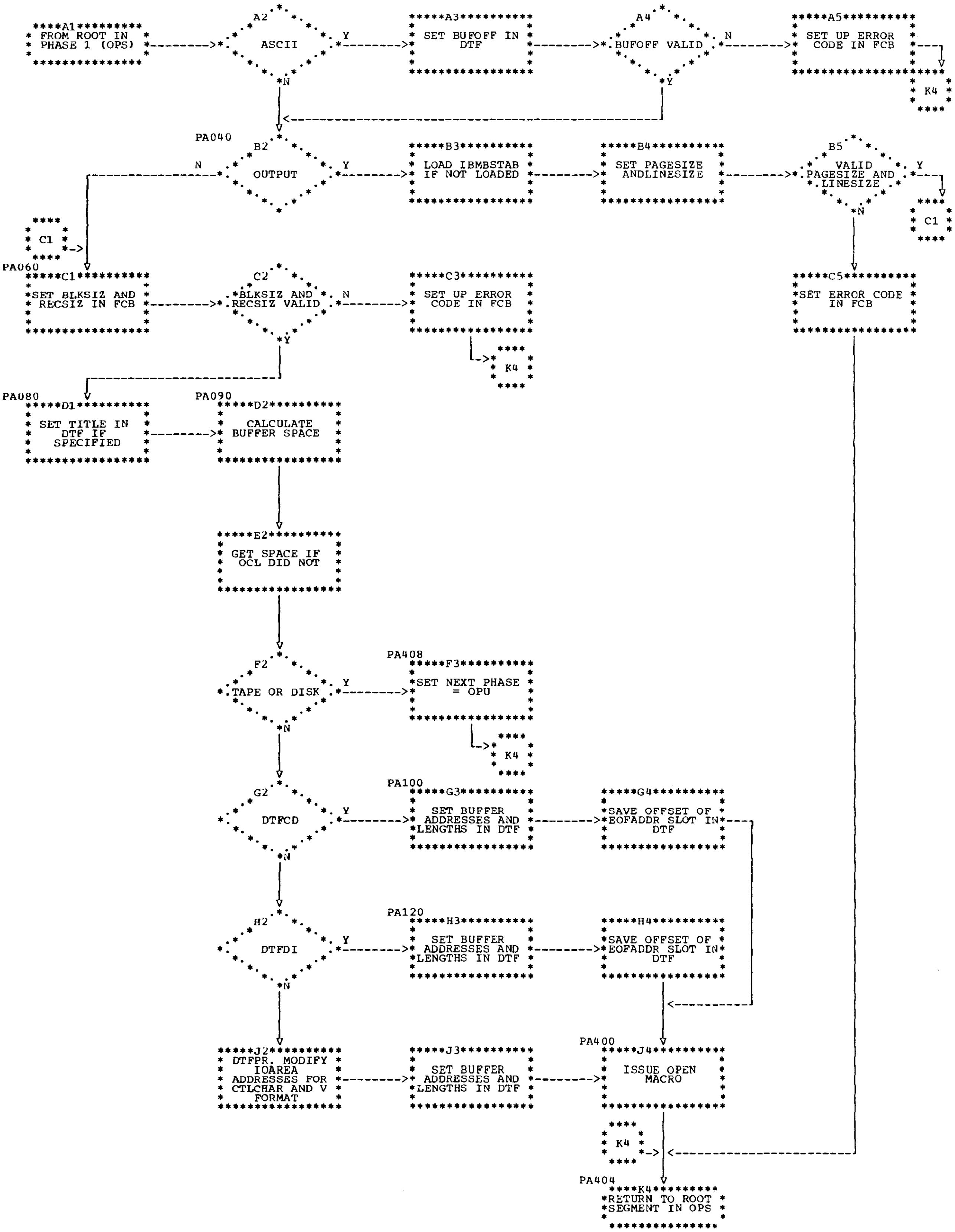

A1
FROM ROOT IN PHASE 1 (OPS)
A2
ASCII
Y
N
A3
SET BUFOFF IN DTF
A4
BUFOFF VALID
N
Y
A5
SET UP ERROR CODE IN FCB
K4
PA040
B2
OUTPUT
N
Y
B3
LOAD IBMBSTAB IF NOT LOADED
B4
SET PAGESIZE ANDLINESIZE
B5
VALID PAGESIZE AND LINESIZE
Y
N
C1
C1
PA060
C1
SET BLKSIZ AND RECSIZ IN FCB
C2
BLKSIZ AND RECSIZ VALID
N
Y
C3
SET UP ERROR CODE IN FCB
K4
C5
SET ERROR CODE IN FCB
PA080
D1
SET TITLE IN DTF IF SPECIFIED
PA090
D2
CALCULATE BUFFER SPACE
E2
GET SPACE IF OCL DID NOT
F2
TAPE OR DISK
Y
N
PA408
F3
SET NEXT PHASE = OPU
K4
G2
DTFCD
Y
N
PA100
G3
SET BUFFER ADDRESSES AND LENGTHS IN DTF
G4
SAVE OFFSET OF EOFADDR SLOT IN DTF
H2
DTFDI
Y
N
PA120
H3
SET BUFFER ADDRESSES AND LENGTHS IN DTF
H4
SAVE OFFSET OF EOFADDR SLOT IN DTF
J2
DTFPR. MODIFY IOAREA ADDRESSES FOR CTLCHAR AND V FORMAT
J3
SET BUFFER ADDRESSES AND LENGTHS IN DTF
PA400
J4
ISSUE OPEN MACRO
K4
PA404
K4
RETURN TO ROOT SEGMENT IN OPS

Chart DOPT. Open (Stream files) - level 2

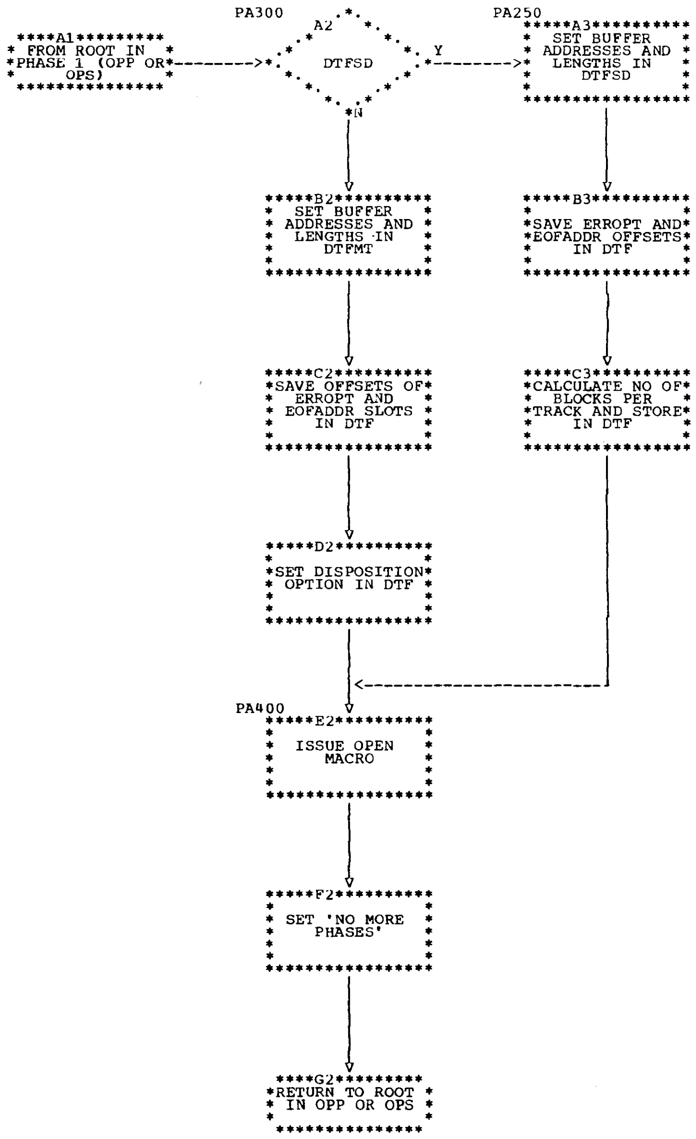
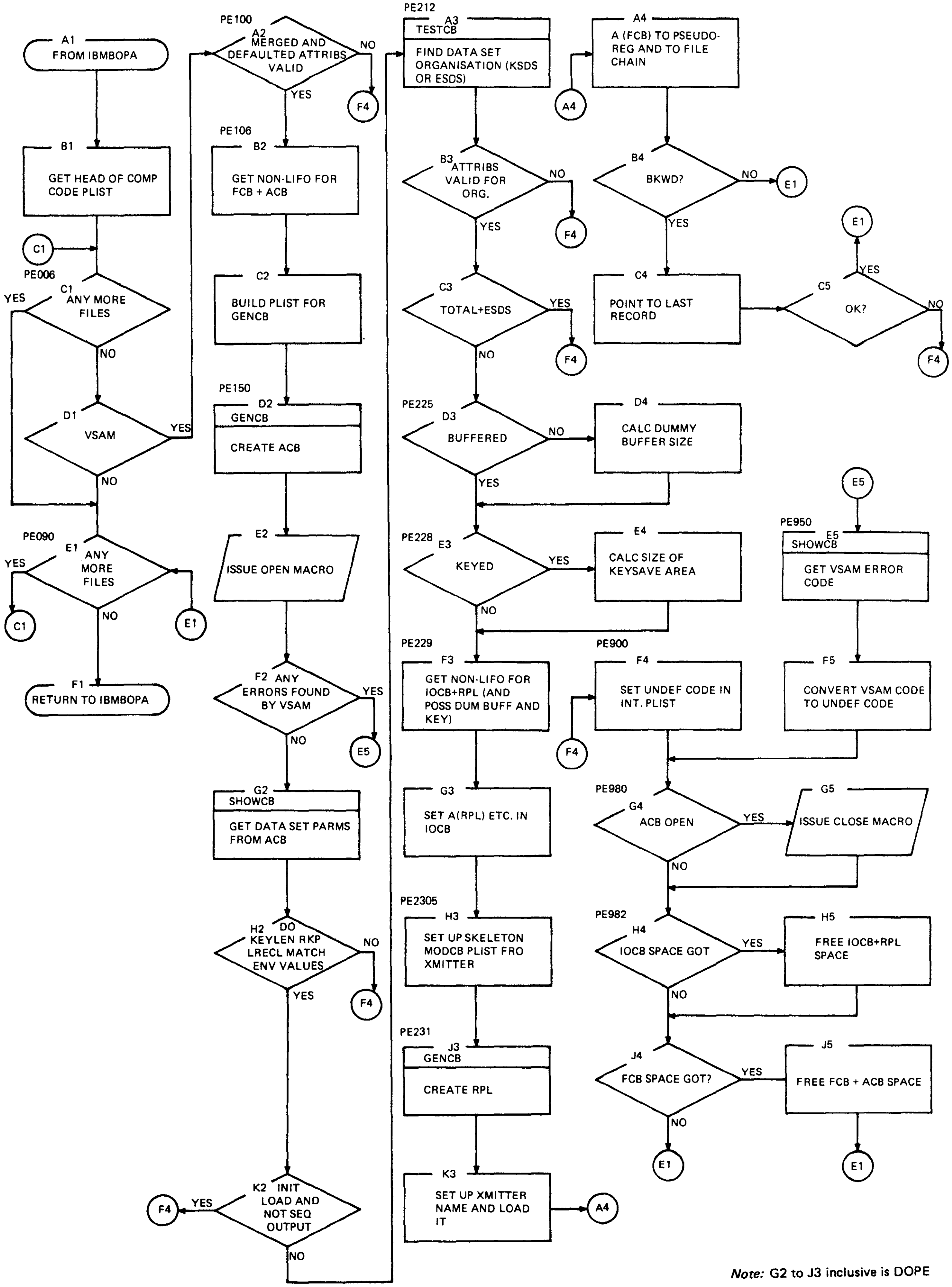

A1
FROM ROOT IN PHASE 1 (OPP OR OPS)
PA300
A2
DTFSD
Y
N
PA250
A3
SET BUFFER ADDRESSES AND LENGTHS IN DTFSD
B2
SET BUFFER ADDRESSES AND LENGTHS IN DTFMT
B3
SAVE ERROPT AND EOFADDR OFFSETS IN DTF
C2
SAVE OFFSETS OF ERROPT AND EOFADDR SLOTS IN DTF
C3
CALCULATE NO OF BLOCKS PER TRACK AND STORE IN DTF
D2
SET DISPOSITION OPTION IN DTF
PA400
E2
ISSUE OPEN MACRO
F2
SET 'NO MORE PHASES'
G2
RETURN TO ROOT IN OPP OR OPS

Chart DOPU. Open (Consecutive buffered and stream files) - level 3


A1 FROM IBMBOPA
B1 GET HEAD OF COMP CODE PLIST
C1
PE006
C1 ANY MORE FILES
YES
NO
D1 VSAM
YES
NO
PE090
E1 ANY MORE FILES
YES
NO
C1
E1
F1 RETURN TO IBMBOPA
PE100
A2 MERGED AND DEFAULTED ATTRIBS VALID
NO
YES
F4
PE106
B2 GET NON-LIFO FOR FCB + ACB
C2 BUILD PLIST FOR GENCB
PE150
D2 GENCB
CREATE ACB
E2 ISSUE OPEN MACRO
F2 ANY ERRORS FOUND BY VSAM
YES
NO
E5
G2 SHOWCB
GET DATA SET PARMS FROM ACB
H2 DO KEYLEN RKP LRECL MATCH ENV VALUES
NO
YES
F4
K2 INIT LOAD AND NOT SEQ OUTPUT
YES
F4
NO
PE212
A3 TESTCB
FIND DATA SET ORGANISATION (KSDS OR ESDS)
B3 ATTRIBS VALID FOR ORG.
NO
YES
F4
C3 TOTAL+ESDS
YES
NO
F4
PE225
D3 BUFFERED
NO
YES
PE228
E3 KEYED
YES
NO
PE229
F3 GET NON-LIFO FOR IOCB+RPL (AND POSS DUM BUFF AND KEY)
G3 SET A(RPL) ETC. IN IOCB
PE2305
H3 SET UP SKELETON MODCB PLIST FRO XMITTER
PE231
J3 GENCB
CREATE RPL
K3 SET UP XMITTER NAME AND LOAD IT
A4
A4 A (FCB) TO PSEUDO-REG AND TO FILE CHAIN
A4
B4 BKWD?
NO
E1
YES
C4 POINT TO LAST RECORD
D4 CALC DUMMY BUFFER SIZE
E4 CALC SIZE OF KEYSAVE AREA
PE900
F4 SET UNDEF CODE IN INT. PLIST
F4
PE980
G4 ACB OPEN
YES
NO
PE982
H4 IOCB SPACE GOT
YES
NO
J4 FCB SPACE GOT?
YES
NO
E1
E1
C5 OK?
YES
NO
F4
E5
PE950
E5 SHOWCB
GET VSAM ERROR CODE
F5 CONVERT VSAM CODE TO UNDEF CODE
G5 ISSUE CLOSE MACRO
H5 FREE IOCB+RPL SPACE
J5 FREE FCB + ACB SPACE
E1

Note: G2 to J3 inclusive is DOPE

|Chart DOPV/DOPE. Open (VSAM)

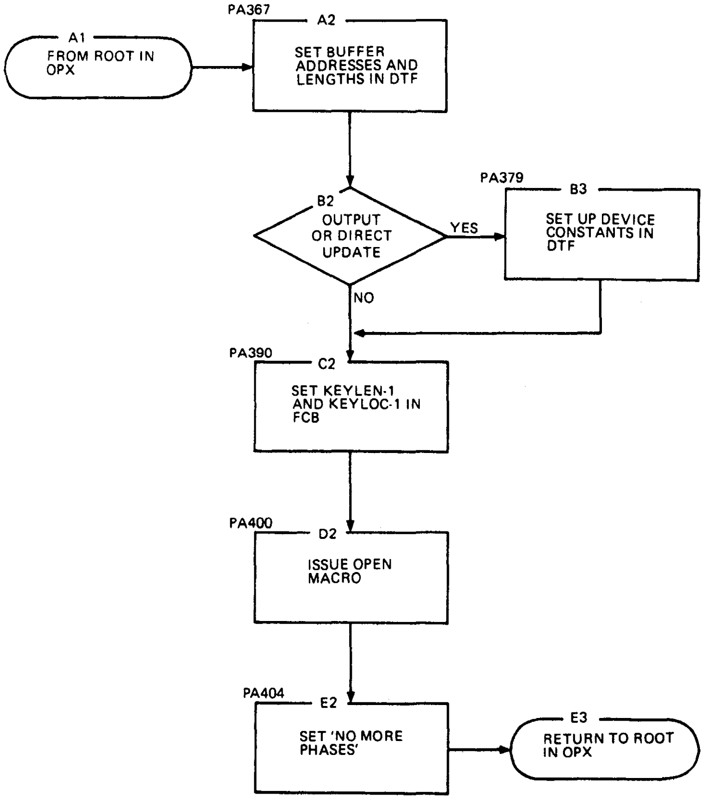


Chart DOPW. Open (Indexed files) - level 4

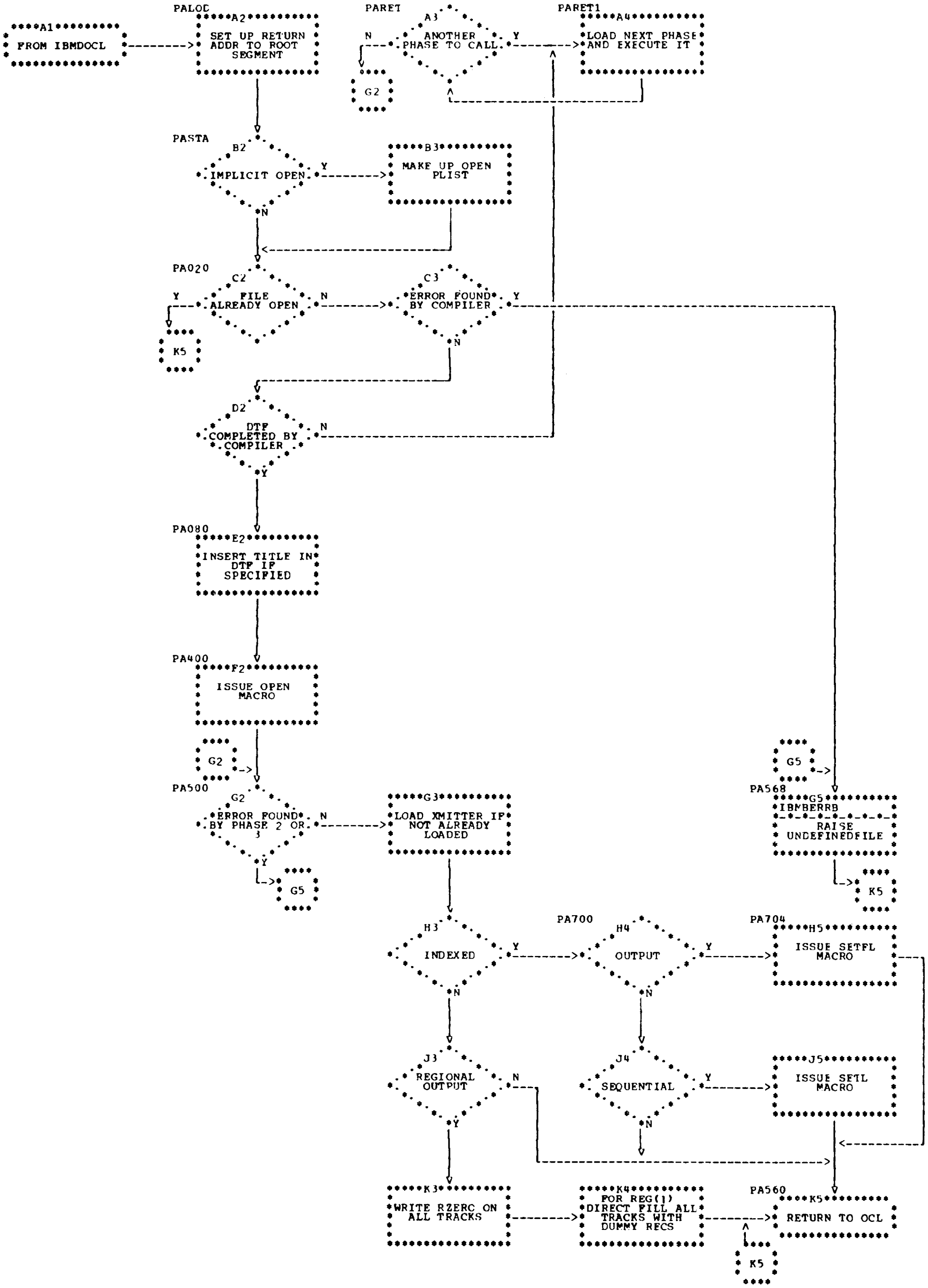


Chart DOPX. Open (Regional and Indexed files)

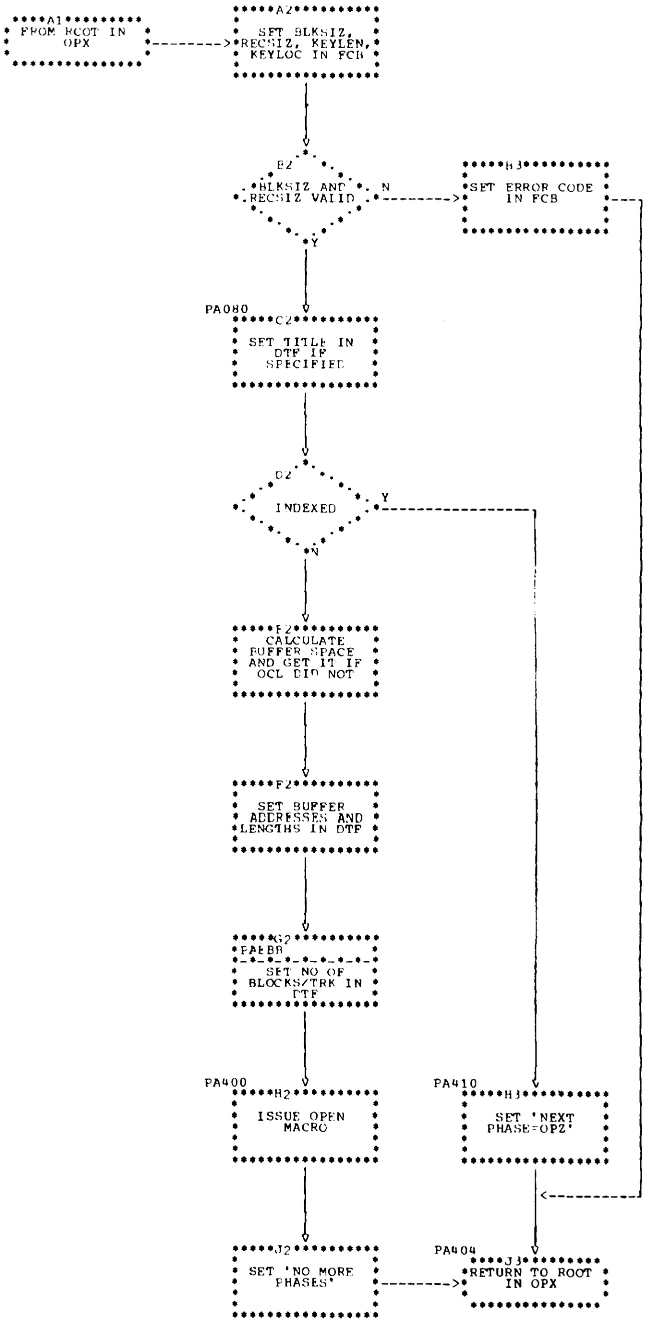


Chart DOPY. Open (Regional and Indexed) - level 2

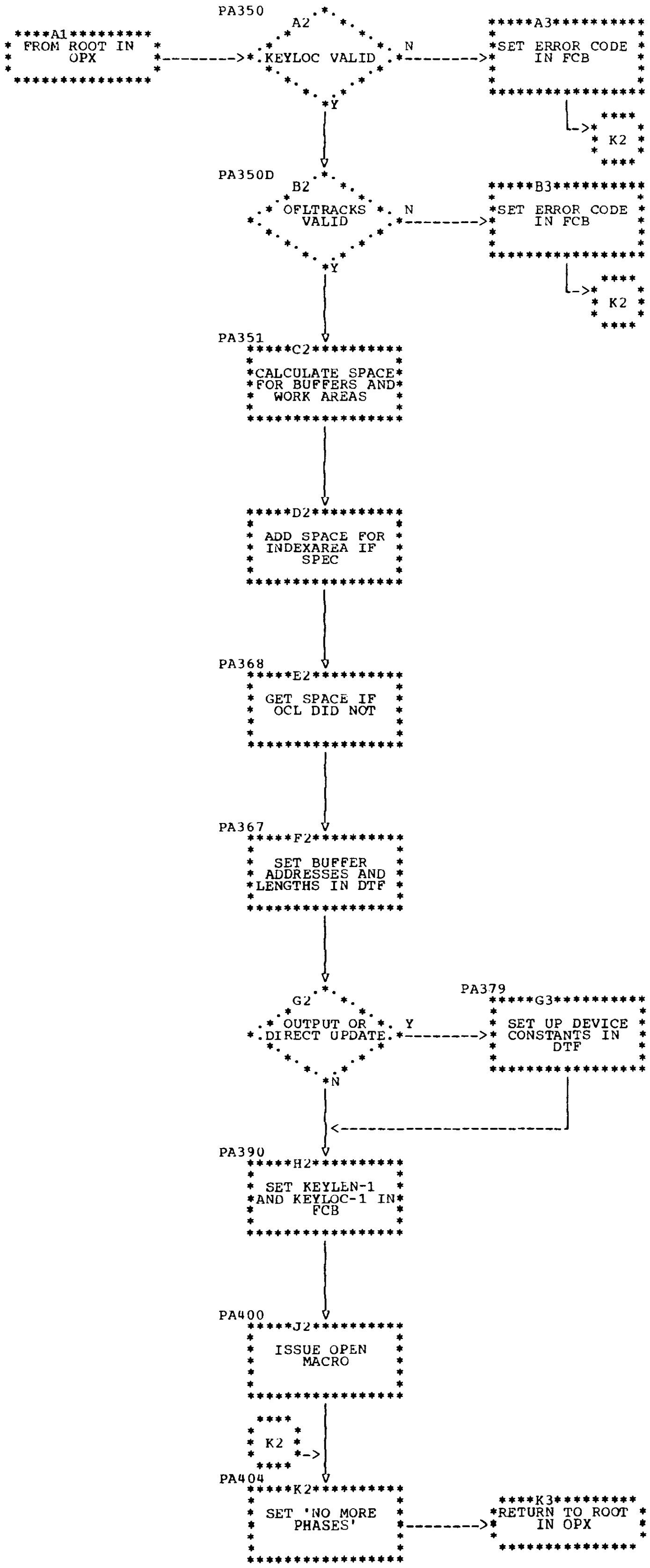


Chart DOPZ. Open (Regional Indexed files) - level 3

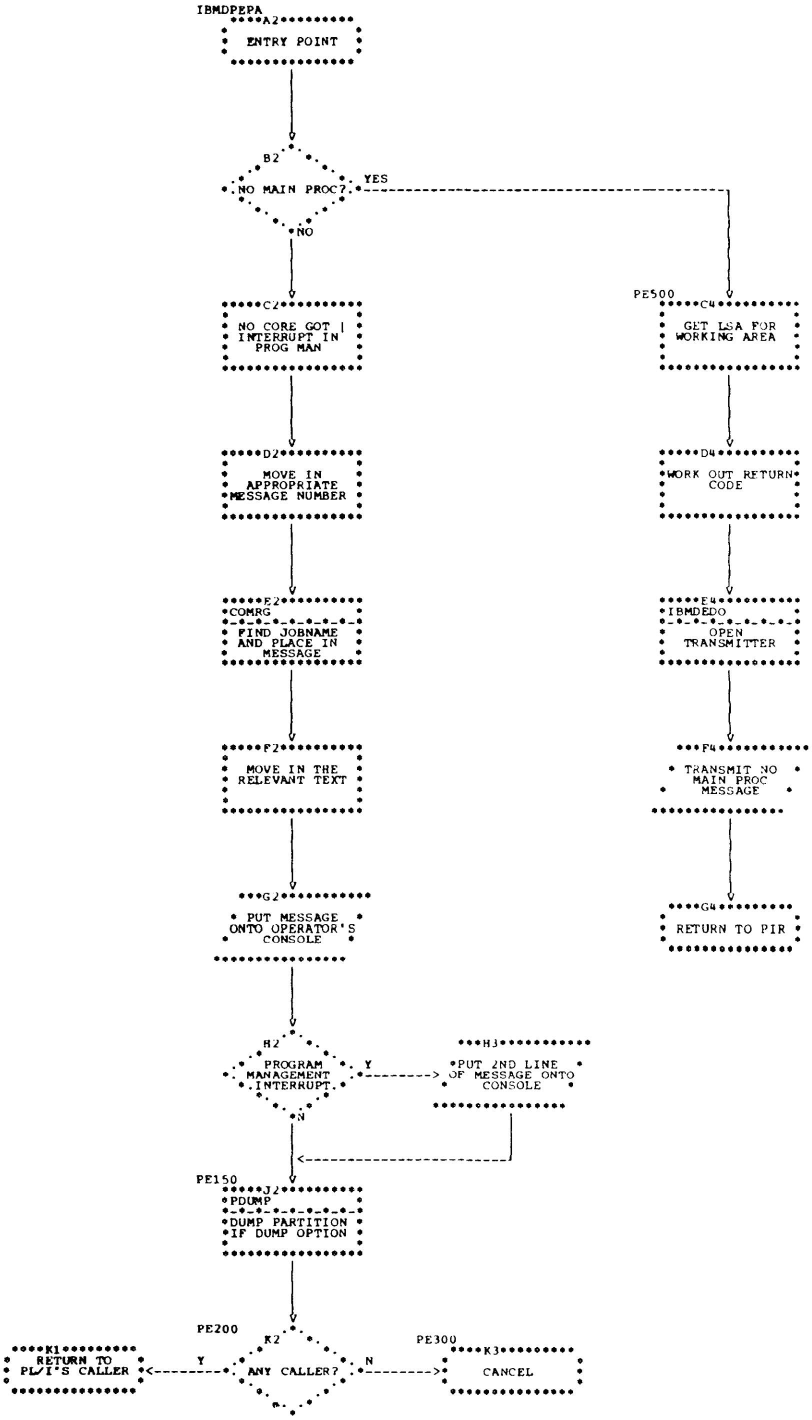
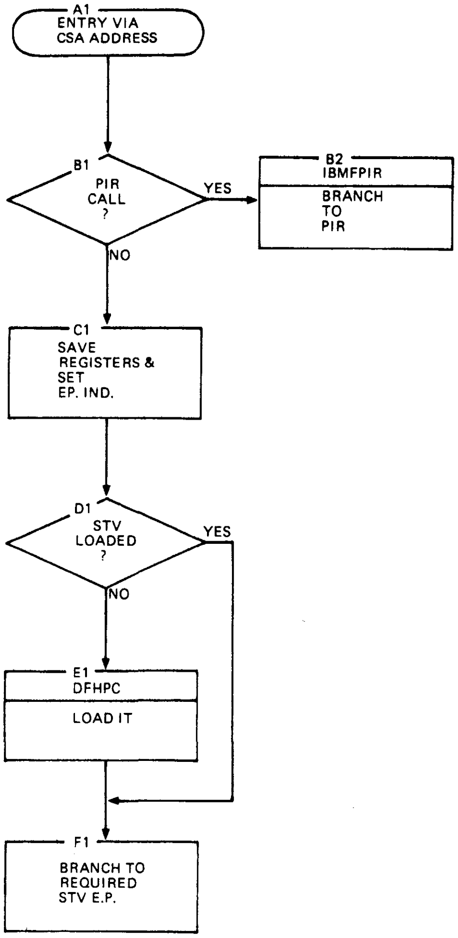
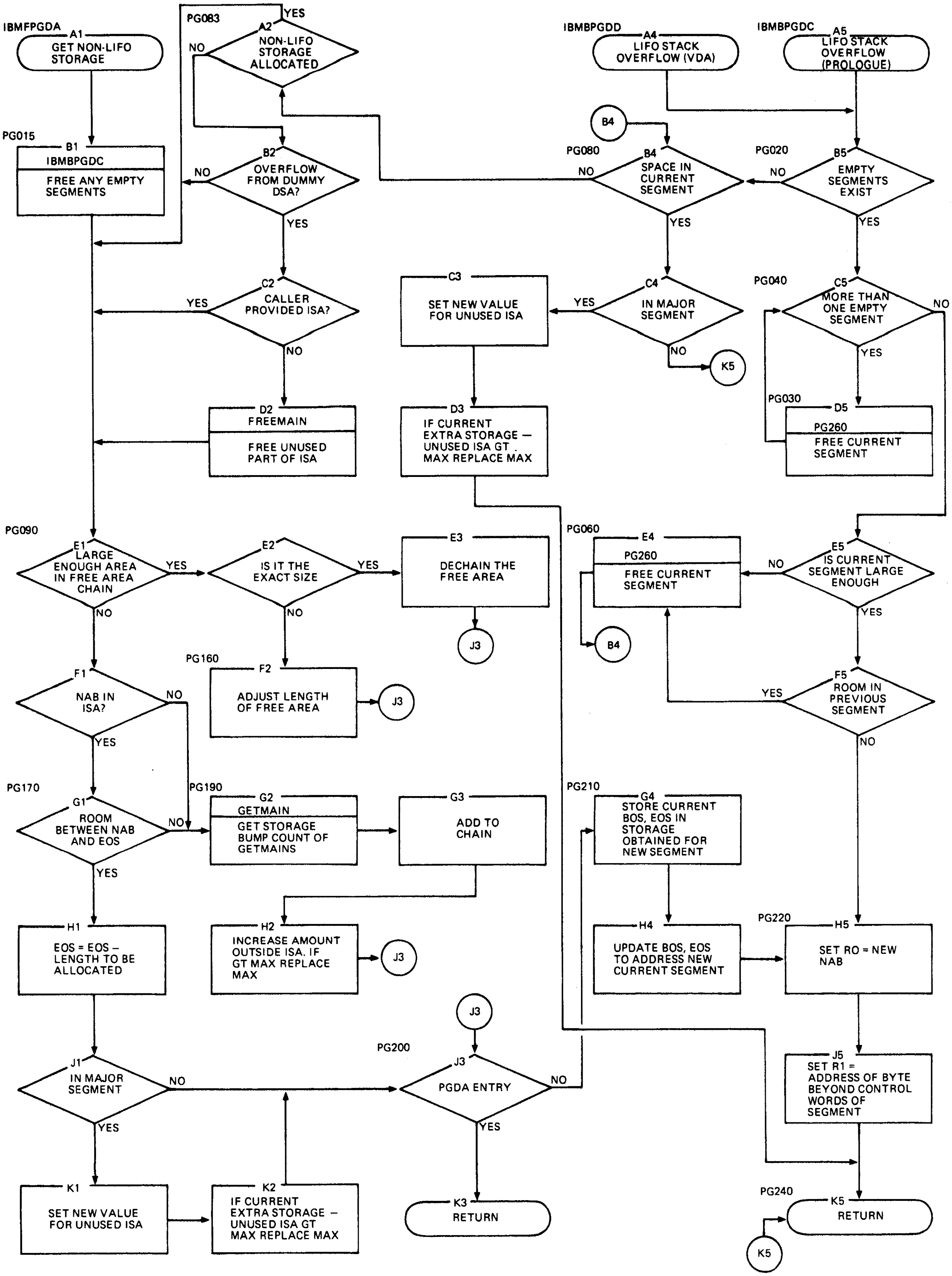

IBMDPEPA
A2
ENTRY POINT
B2
NO MAIN PROC?
YES
NO
C2
NO CORE GOT | INTERRUPT IN PROG MAN
PE500
C4
GET LSA FOR WORKING AREA
D2
MOVE IN APPROPRIATE MESSAGE NUMBER
D4
WORK OUT RETURN CODE
E2
COMRG
FIND JOBNAME AND PLACE IN MESSAGE
E4
IBMDEDO
OPEN TRANSMITTER
F2
MOVE IN THE RELEVANT TEXT
F4
TRANSMIT NO MAIN PROC MESSAGE
G2
PUT MESSAGE ONTO OPERATOR'S CONSOLE
G4
RETURN TO PIR
H2
PROGRAM MANAGEMENT INTERRUPT.
Y
N
H3
PUT 2ND LINE OF MESSAGE ONTO CONSOLE
PE150
J2
PDUMP
DUMP PARTITION IF DUMP OPTION
PE200
K2
ANY CALLER?
Y
N
K1
RETURN TO PL/I'S CALLER
PE300
K3
CANCEL

Chart DPEP. Housekeeping diagnostic message module



|Chart FPCC. Routing module - CICS



|Chart FPGD. Storage management (non-multitasking) - CICS (part 1 of 2)

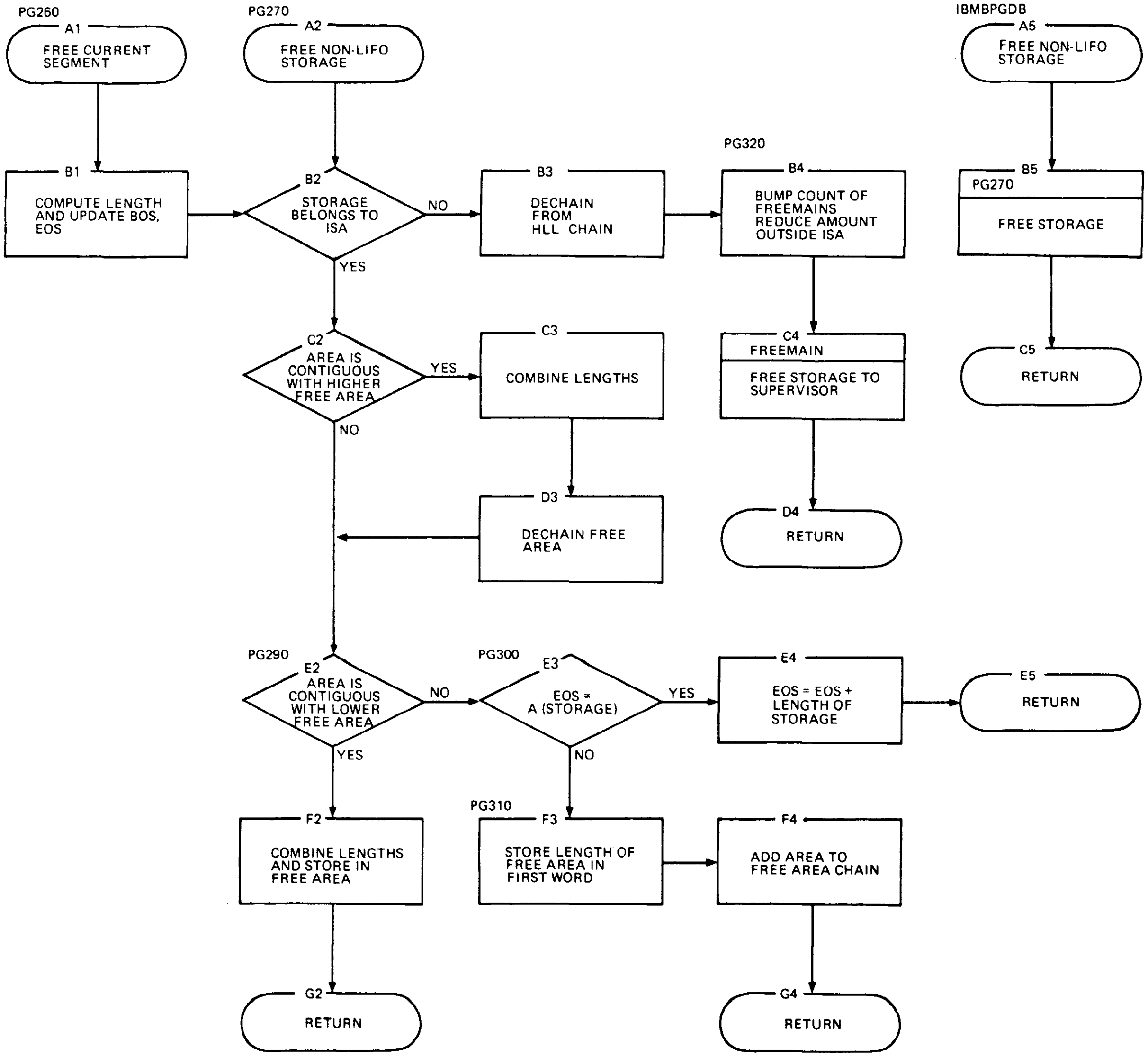
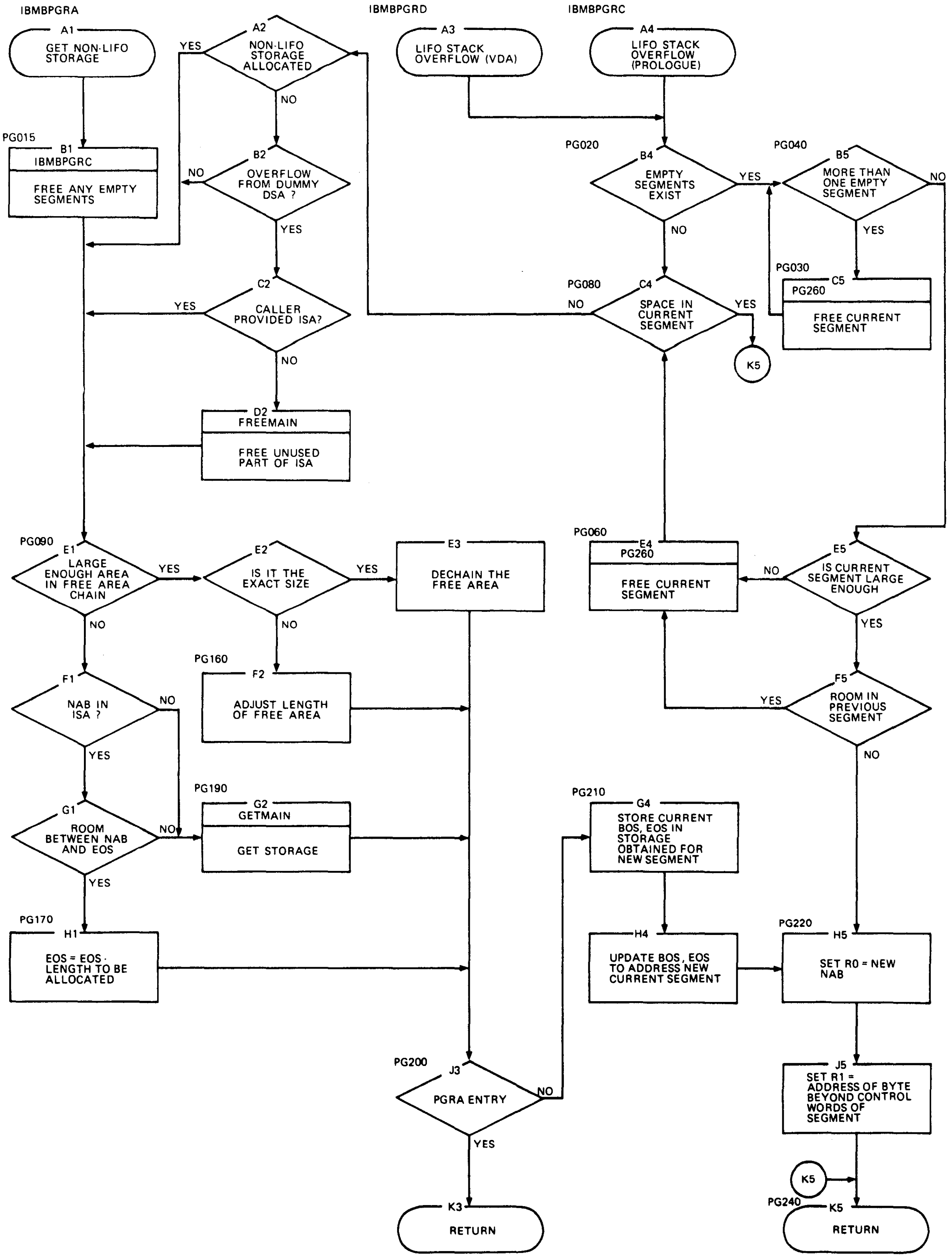
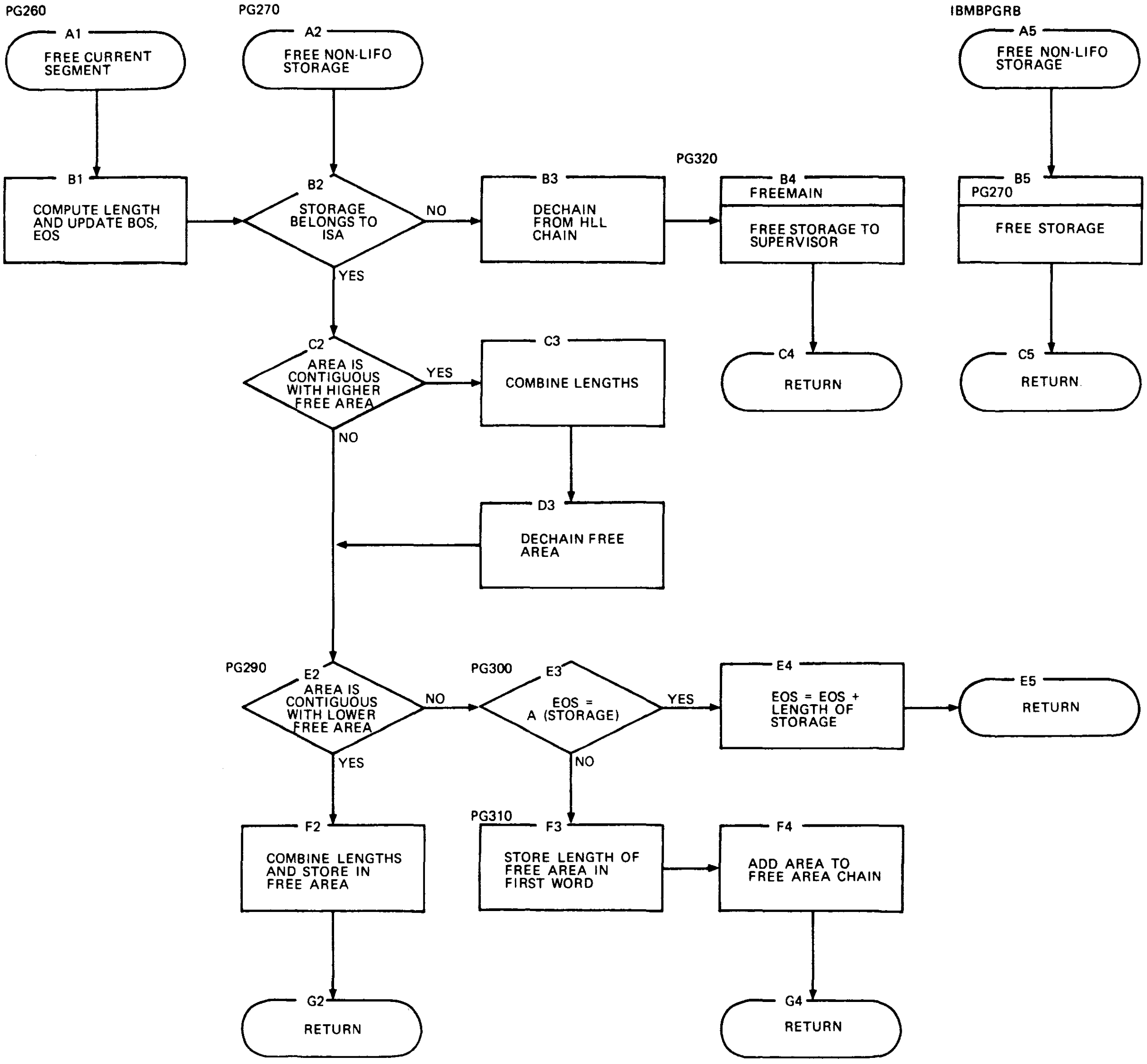

PG260
A1
FREE CURRENT SEGMENT
B1
COMPUTE LENGTH AND UPDATE BOS, EOS
PG270
A2
FREE NON-LIFO STORAGE
B2
STORAGE BELONGS TO ISA
NO
YES
C2
AREA IS CONTIGUOUS WITH HIGHER FREE AREA
YES
NO
B3
DECHAIN FROM HLL CHAIN
PG320
B4
BUMP COUNT OF FREEMAINS REDUCE AMOUNT OUTSIDE ISA
C3
COMBINE LENGTHS
C4
FREEMAIN
FREE STORAGE TO SUPERVISOR
D3
DECHAIN FREE AREA
D4
RETURN
IBMBPGDB
A5
FREE NON-LIFO STORAGE
B5
PG270
FREE STORAGE
C5
RETURN
PG290
E2
AREA IS CONTIGUOUS WITH LOWER FREE AREA
NO
YES
PG300
E3
EOS = A (STORAGE)
YES
NO
E4
EOS = EOS + LENGTH OF STORAGE
E5
RETURN
F2
COMBINE LENGTHS AND STORE IN FREE AREA
PG310
F3
STORE LENGTH OF FREE AREA IN FIRST WORD
F4
ADD AREA TO FREE AREA CHAIN
G2
RETURN
G4
RETURN

Chart FPGD. Storage management (non-multitasking) - CICS (part 2 of 2)



|Chart FPGR. Storage management (non-multitasking) - CICS (part 1 of 2)



|Chart FPGR. Storage management (non-multitasking) - CICS (part 2 of 2)

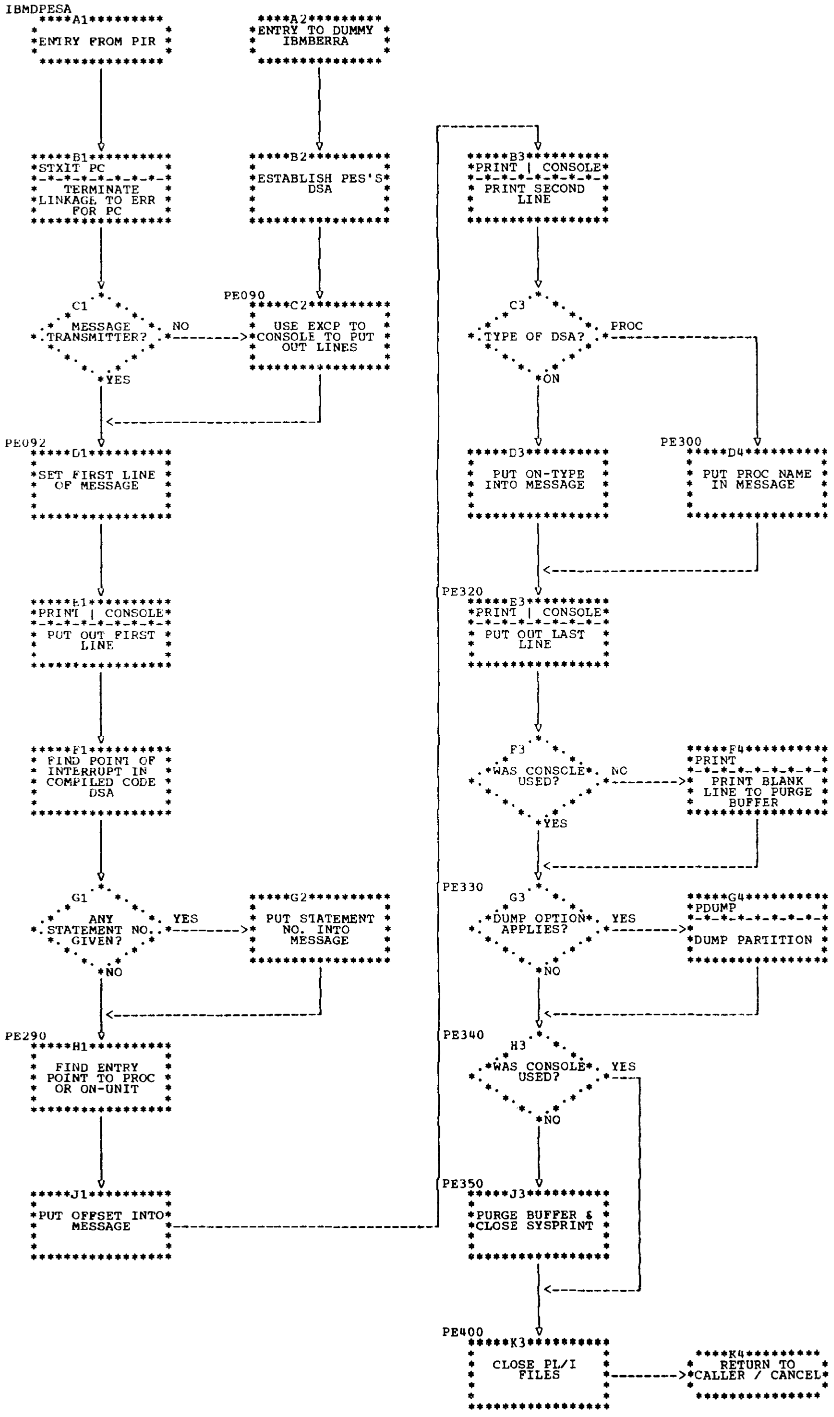


Chart DPES. Storage management diagnostic message module

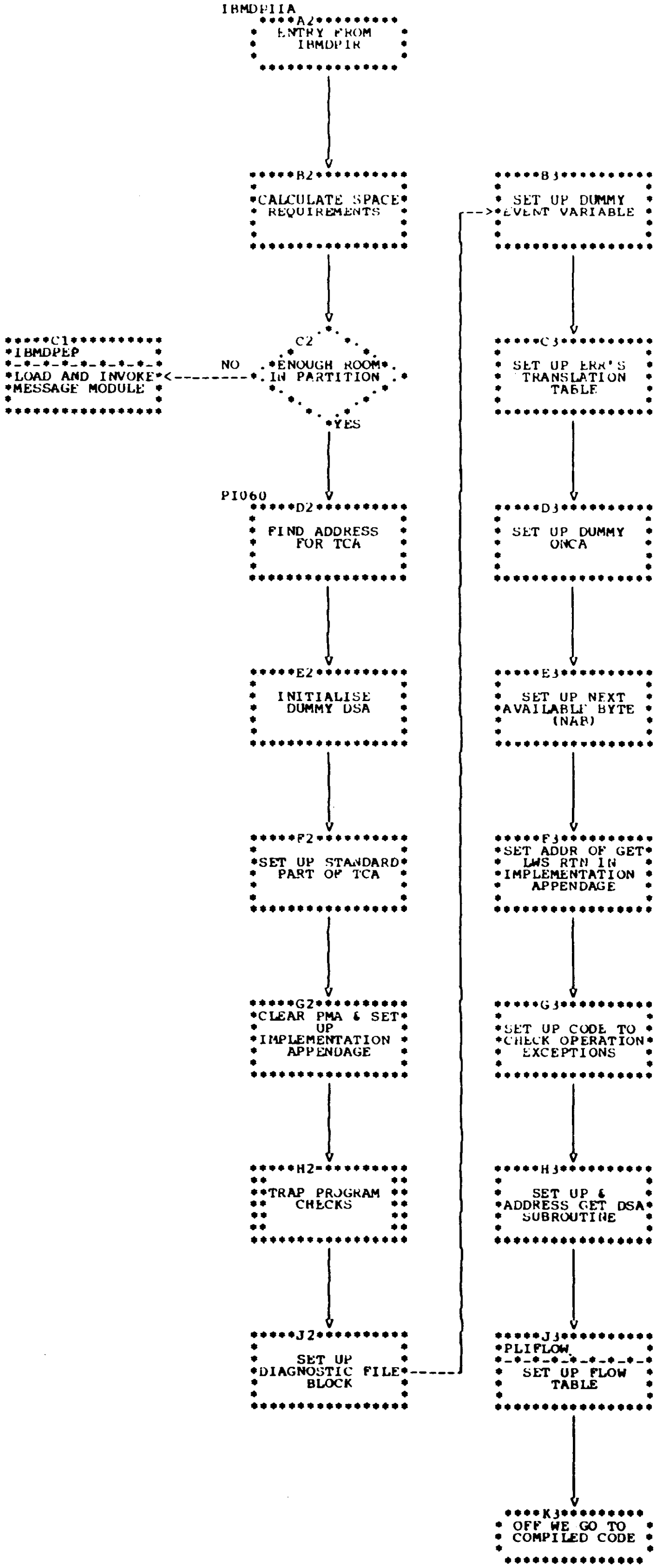
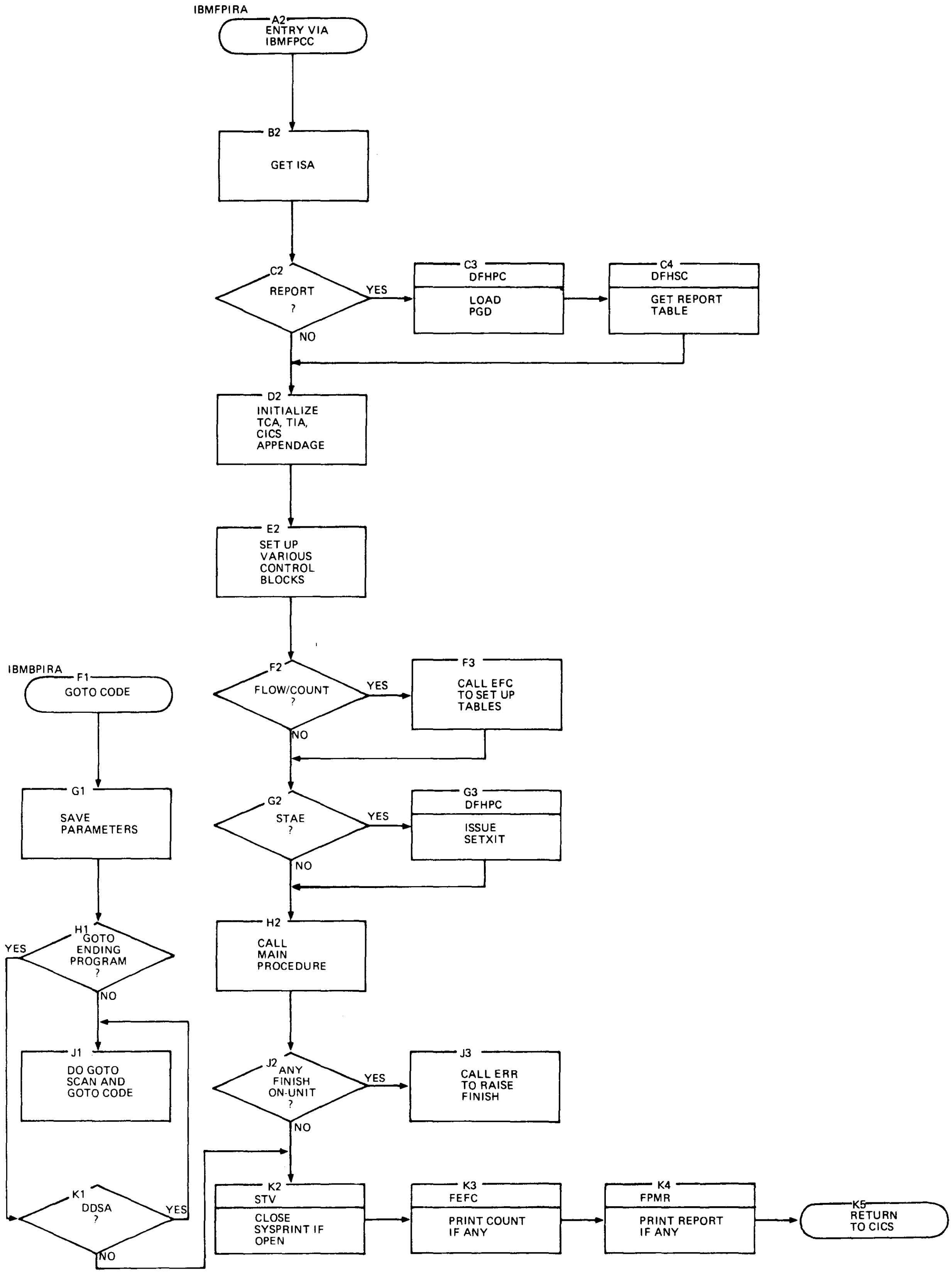

IBMDPIIA
A2
ENTRY FROM IBMDPIR
B2
CALCULATE SPACE REQUIREMENTS
C1
IBMDPEP
LOAD AND INVOKE MESSAGE MODULE
C2
ENOUGH ROOM IN PARTITION
NO
YES
PI060
D2
FIND ADDRESS FOR TCA
E2
INITIALISE DUMMY DSA
F2
SET UP STANDARD PART OF TCA
G2
CLEAR PMA & SET UP IMPLEMENTATION APPENDAGE
H2
TRAP PROGRAM CHECKS
J2
SET UP DIAGNOSTIC FILE BLOCK
B3
SET UP DUMMY EVENT VARIABLE
C3
SET UP ERR'S TRANSLATION TABLE
D3
SET UP DUMMY ONCA
E3
SET UP NEXT AVAILABLE BYTE (NAB)
F3
SET ADDR OF GET LWS RTN IN IMPLEMENTATION APPENDAGE
G3
SET UP CODE TO CHECK OPERATION EXCEPTIONS
H3
SET UP & ADDRESS GET DSA SUBROUTINE
J3
PLIFLOW
SET UP FLOW TABLE
K3
OFF WE GO TO COMPILED CODE

Chart DPII. Program ISA initialization module



|Chart HPIR. Initialization/termination module - CICS

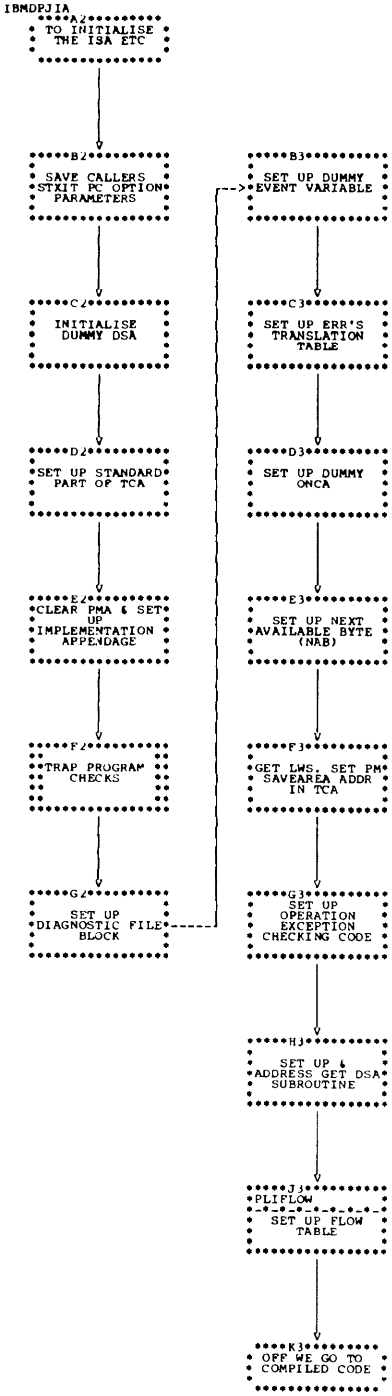

IBMDPJIA
A2
TO INITIALISE THE ISA ETC
B2
SAVE CALLERS STXIT PC OPTION PARAMETERS
C2
INITIALISE DUMMY DSA
D2
SET UP STANDARD PART OF TCA
E2
CLEAR PMA & SET UP IMPLEMENTATION APPENDAGE
F2
TRAP PROGRAM CHECKS
G2
SET UP DIAGNOSTIC FILE BLOCK
B3
SET UP DUMMY EVENT VARIABLE
C3
SET UP ERR'S TRANSLATION TABLE
D3
SET UP DUMMY ONCA
E3
SET UP NEXT AVAILABLE BYTE (NAB)
F3
GET LWS. SET PM SAVEAREA ADDR IN TCA
G3
SET UP OPERATION EXCEPTION CHECKING CODE
H3
SET UP & ADDRESS GET DSA SUBROUTINE
J3
PLIFLOW
SET UP FLOW TABLE
K3
OFF WE GO TO COMPILED CODE

Chart DPJI. Program ISA initialization from caller

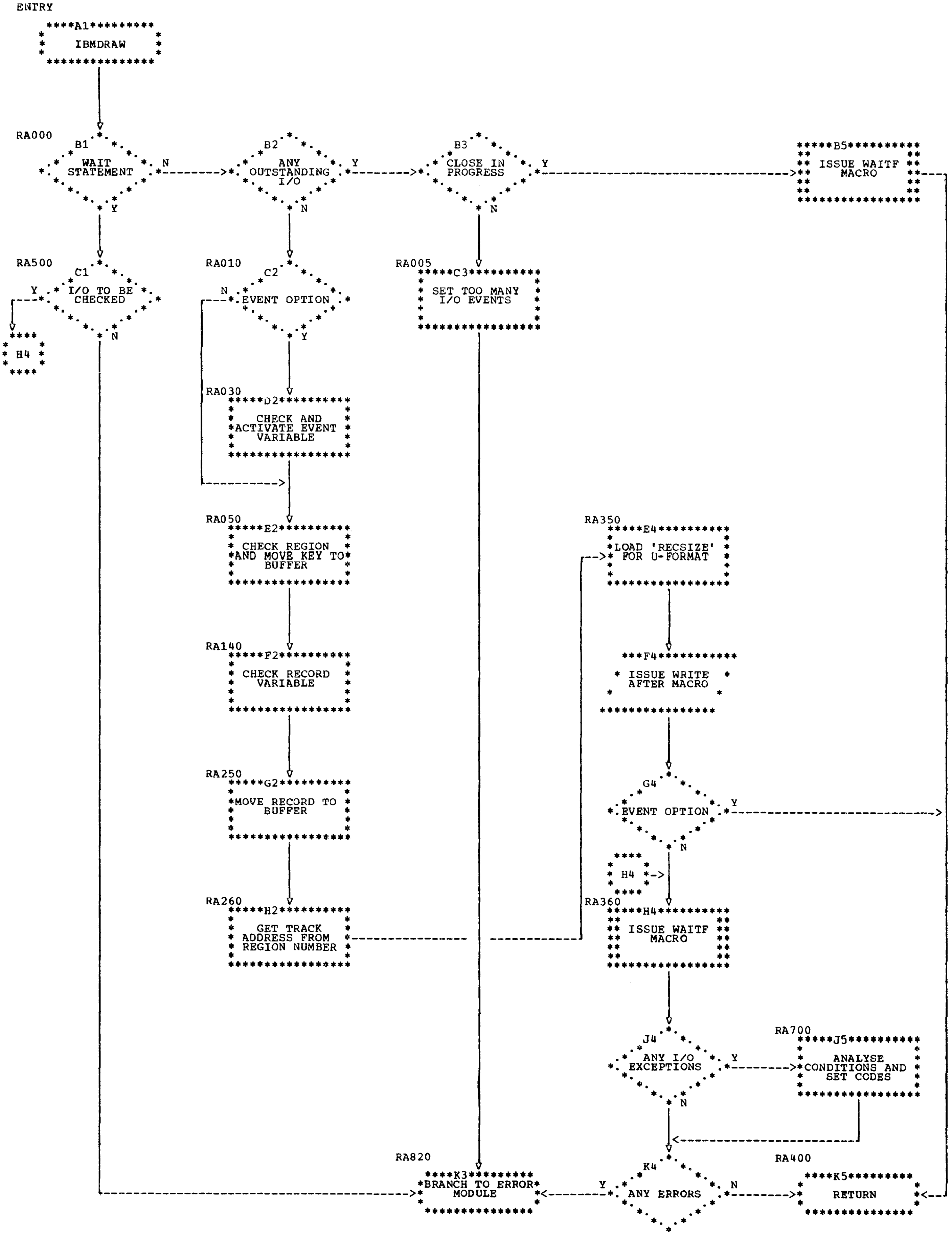


Chart DRAW. Regional(3) sequential unbuffered output transmitter

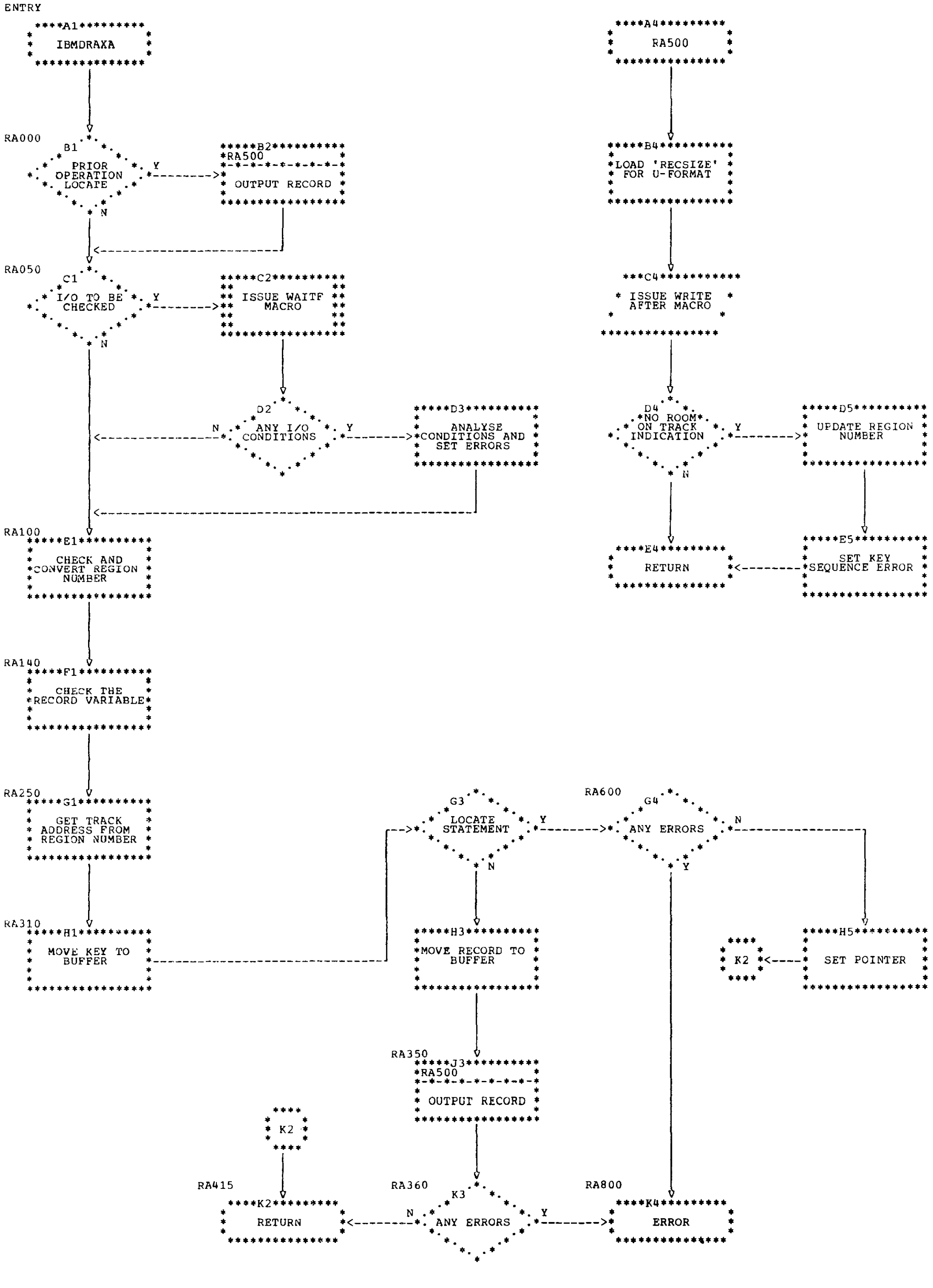


Chart DRAX. Regional(3) sequential buffered output transmitter

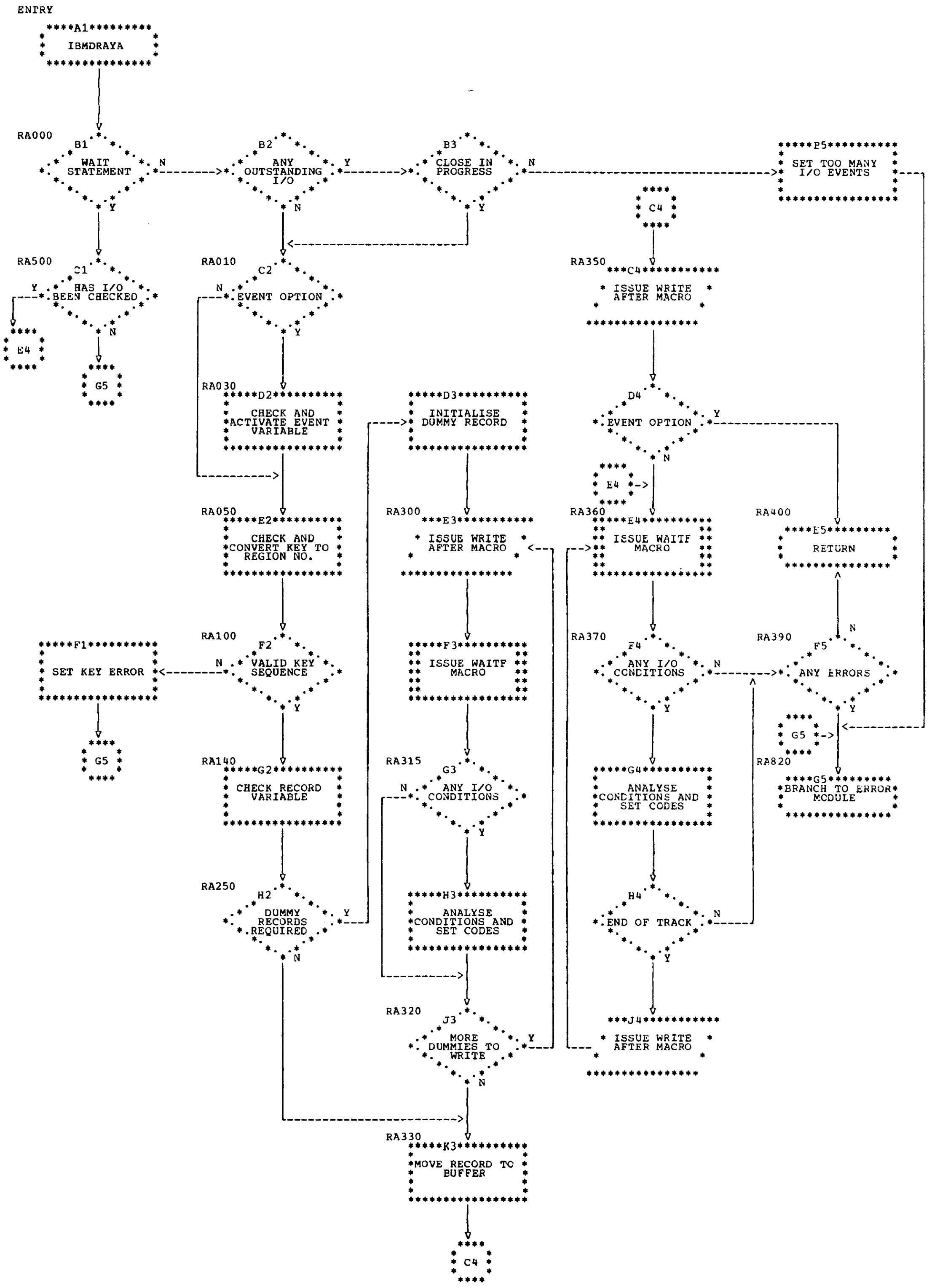


Chart DRAY. Regional(1) sequential unbuffered output transmitter

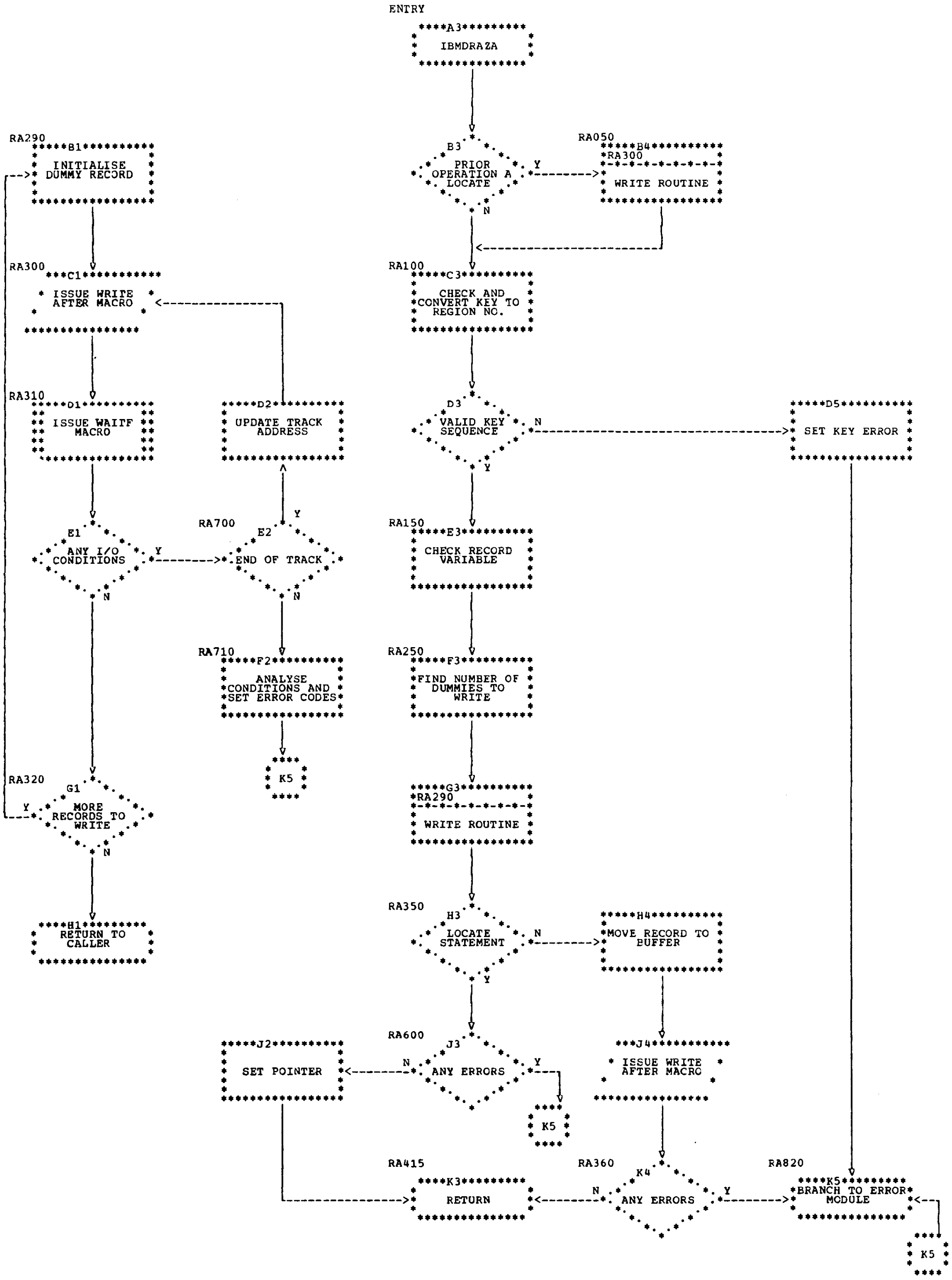


Chart DRAZ. Regional(1) sequential buffered output transmitter

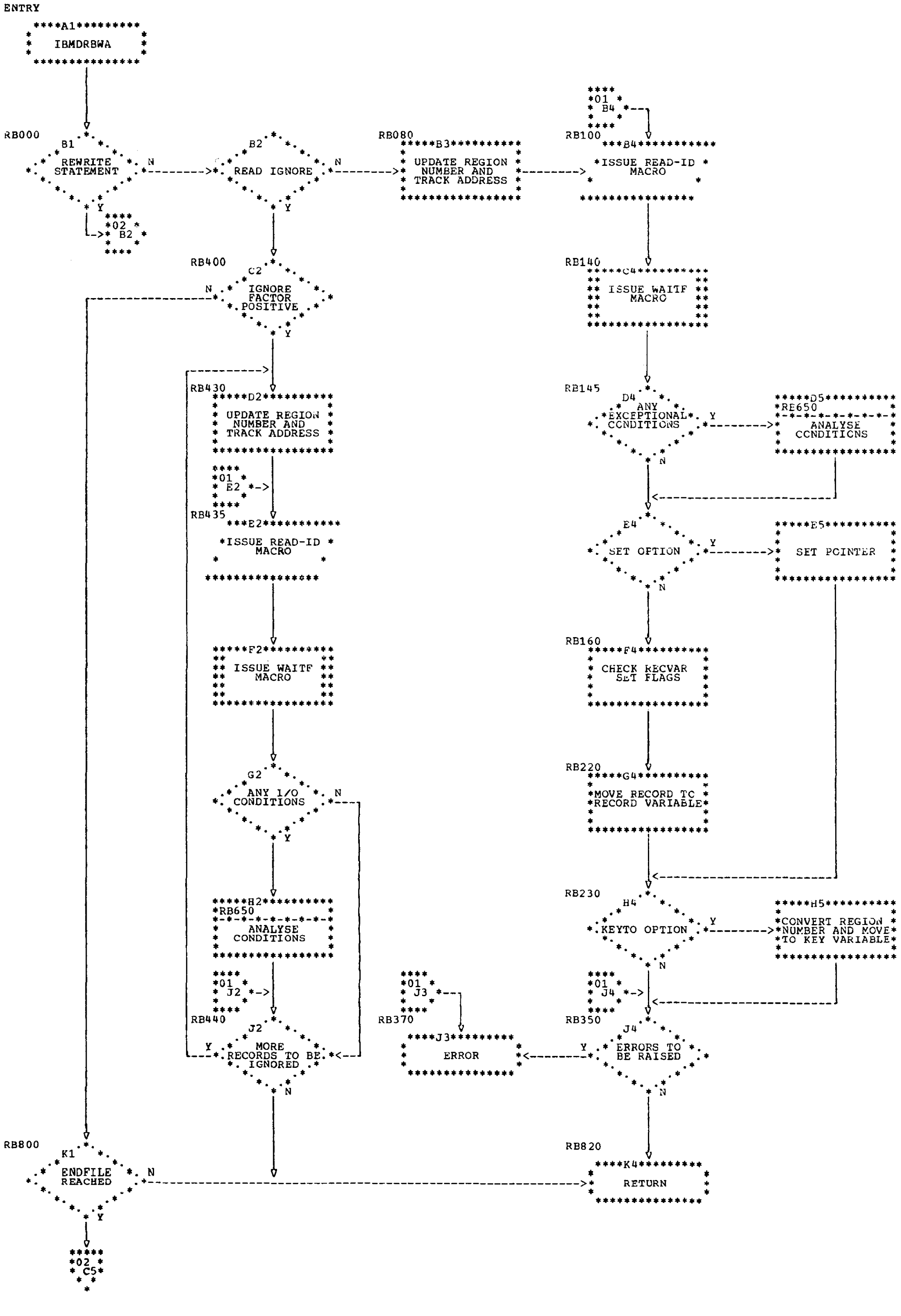


Chart DRBW. Regional(1) sequential buffered input/update transmitter (part 1 of 2)

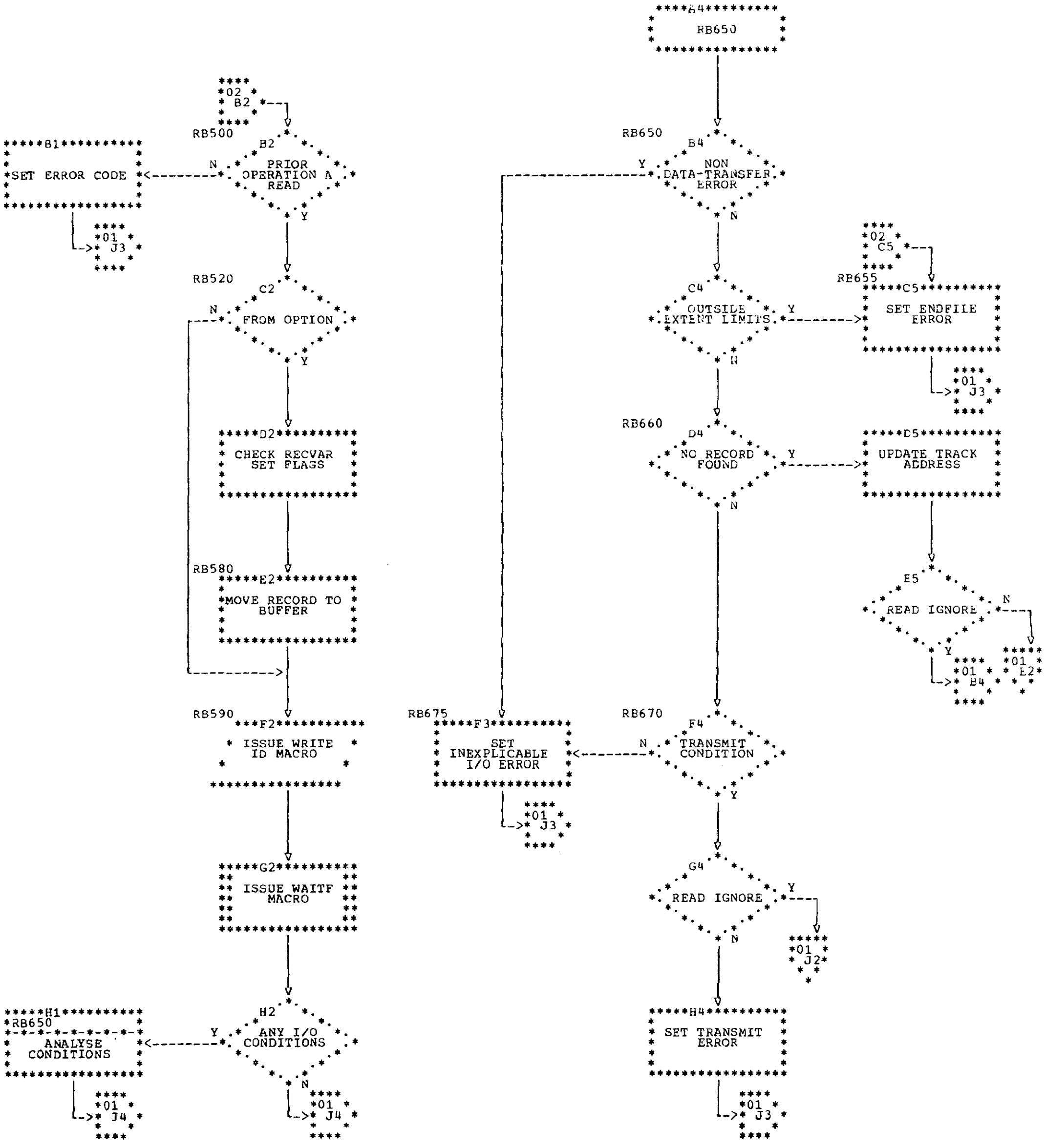


Chart DRBW. Regional(1) sequential buffered input/update transmitter (part 2 of 2)

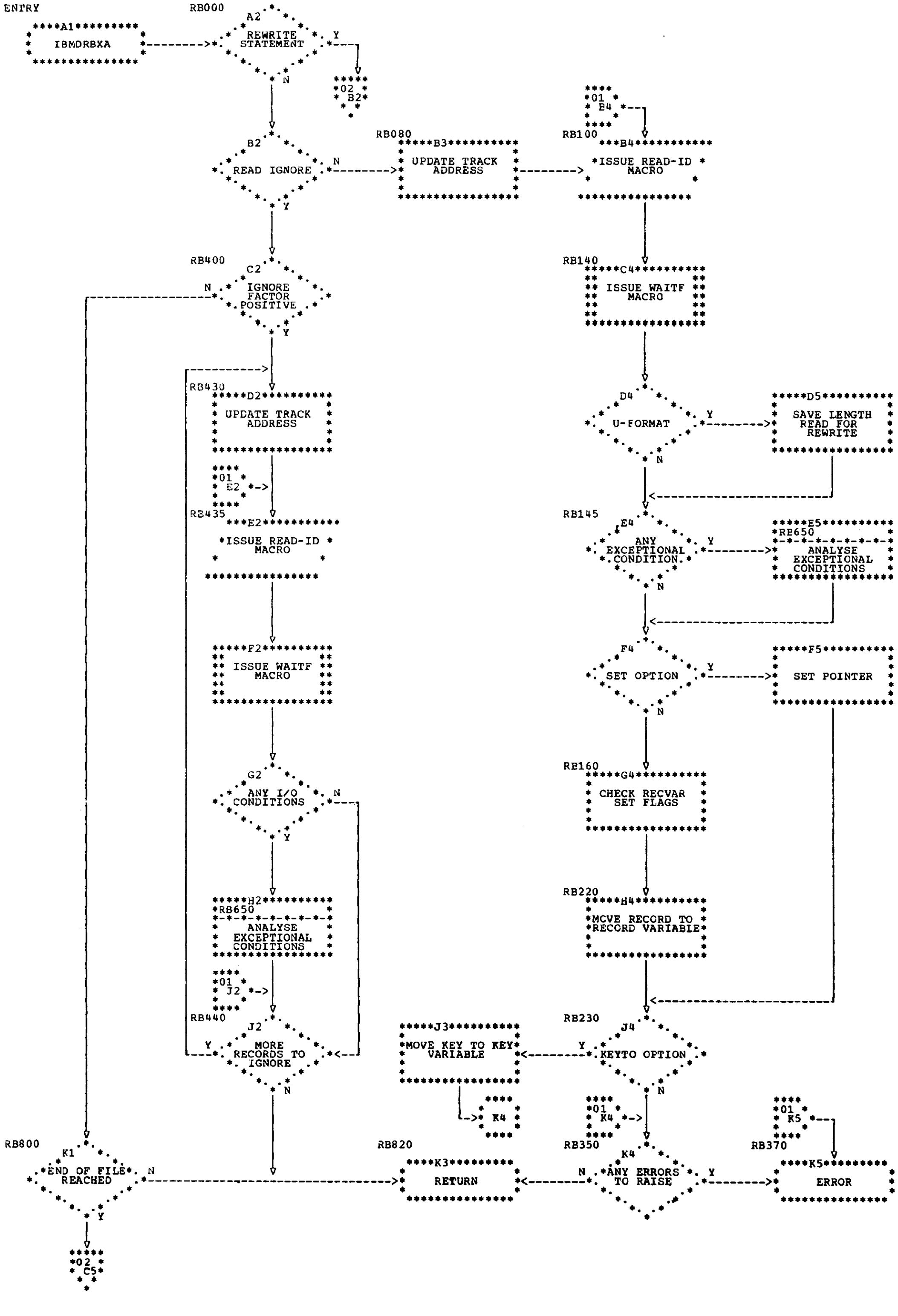


Chart DRBX. Regional(3) sequential buffcred input/update transmitter (part 1 of 2)

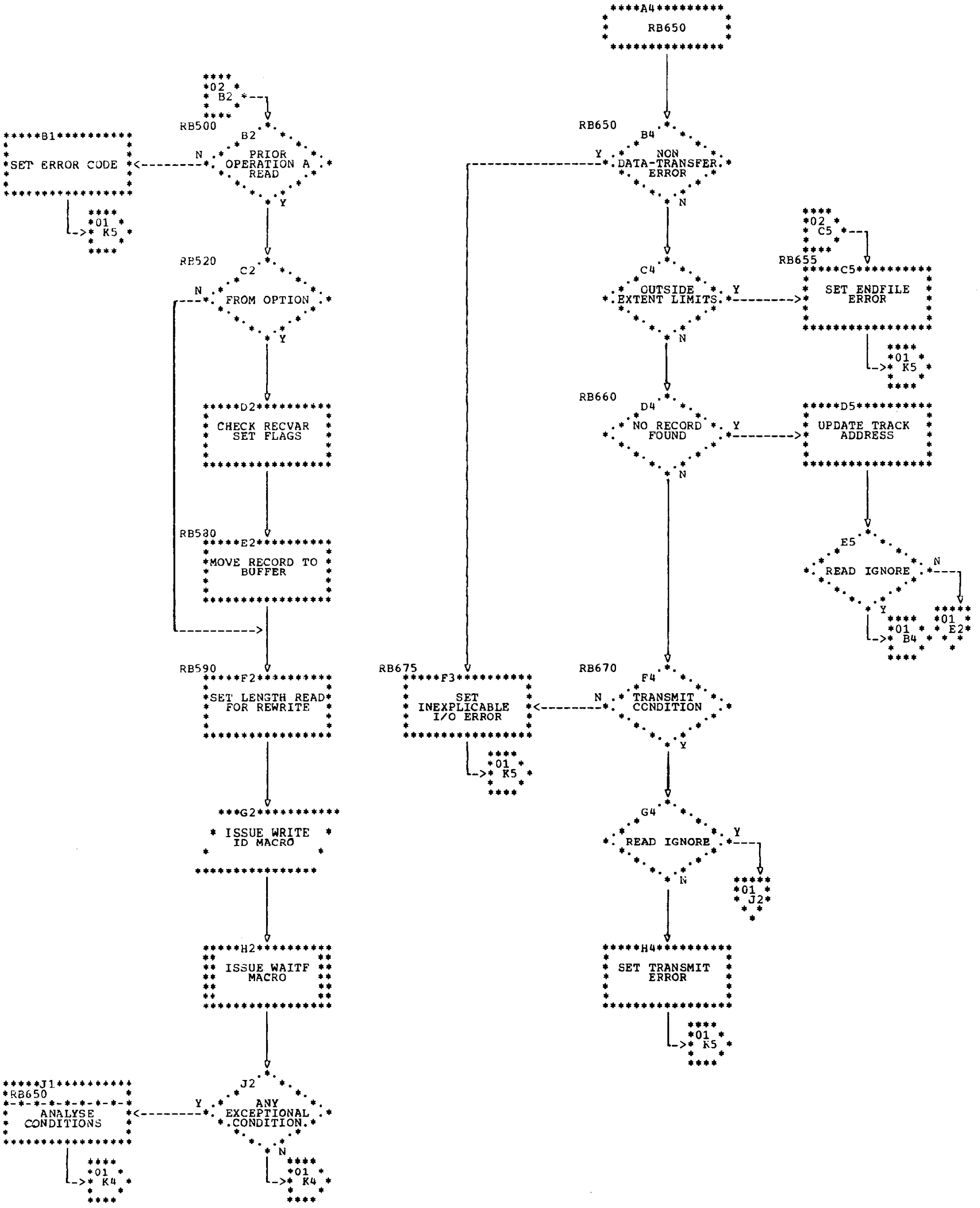


Chart DRBX. Regional(3) sequential buffered input/update transmitter (part 2 of 2)

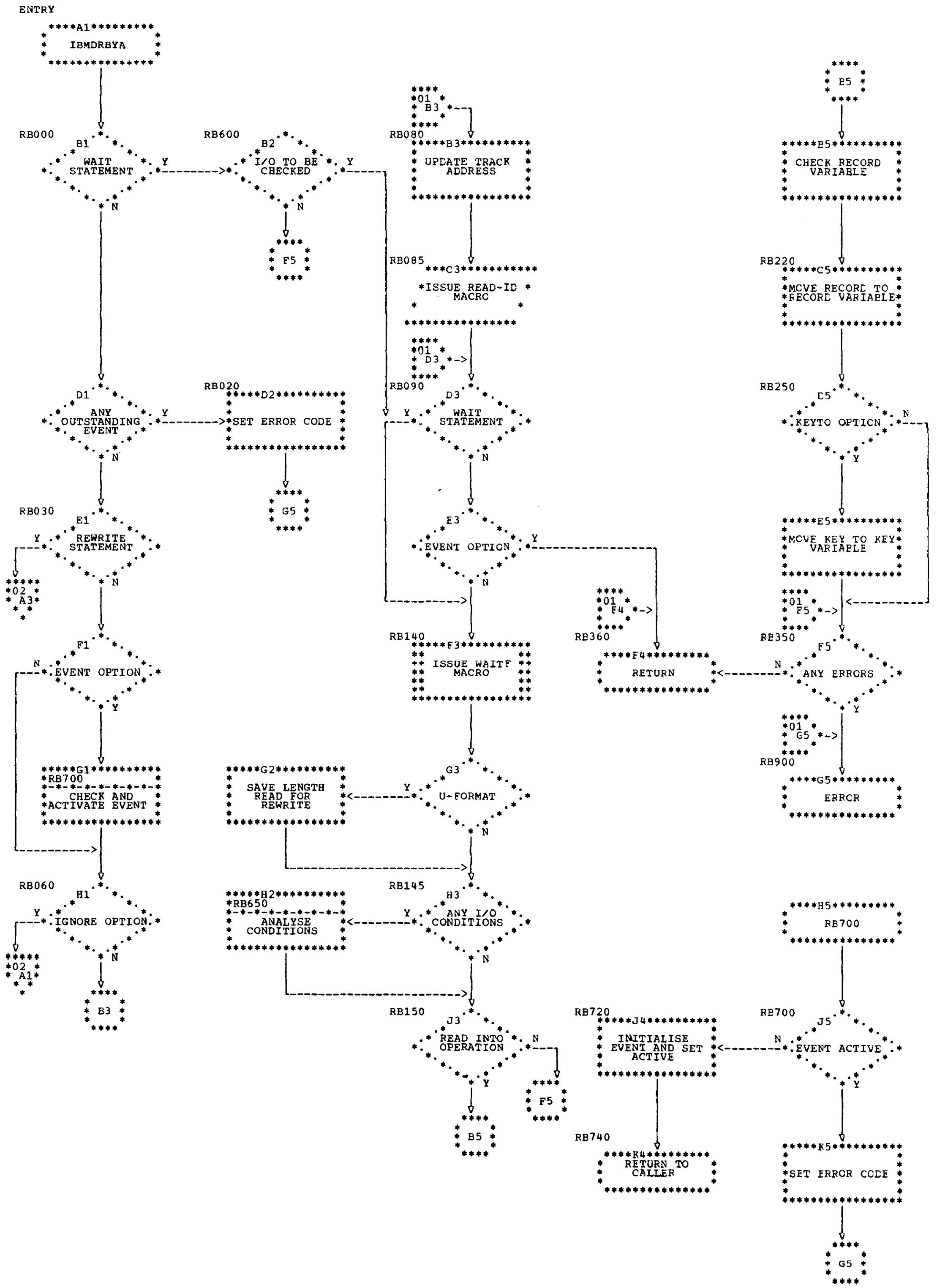


Chart DRBY. Regional(3) sequential unbuffered input/update transmitter (part 1 of 2)

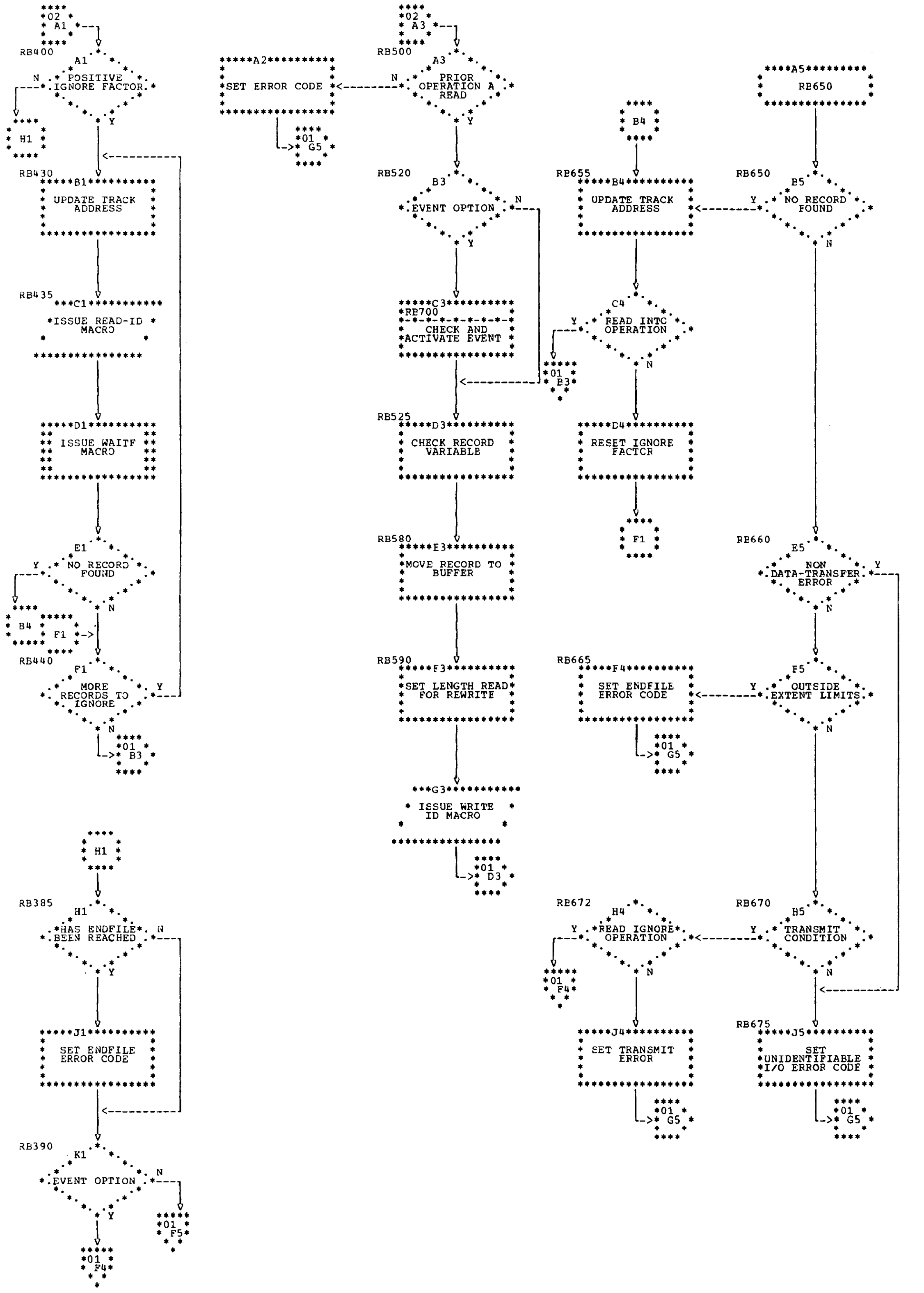

02 A1
RB400 A1 POSITIVE IGNORE FACTOR
N
Y
H1
RB430 B1 UPDATE TRACK ADDRESS
RB435 C1 ISSUE READ-ID MACRO
D1 ISSUE WAITF MACRO
E1 NO RECORD FOUND
Y
N
B4
F1
RB440 F1 MORE RECORDS TO IGNORE
Y
N
01 B3
H1
RB385 H1 HAS ENDFILE BEEN REACHED
N
Y
J1 SET ENDFILE ERROR CODE
RB390 K1 EVENT OPTION
N
Y
01 F5
01 F4
A2 SET ERROR CODE
01 G5
02 A3
RB500 A3 PRIOR OPERATION A READ
N
Y
RB520 B3 EVENT OPTION
N
Y
C3 RB700 CHECK AND ACTIVATE EVENT
RB525 D3 CHECK RECORD VARIABLE
RB580 E3 MOVE RECORD TO BUFFER
RB590 F3 SET LENGTH READ FOR REWRITE
G3 ISSUE WRITE ID MACRO
01 D3
B4
RB655 B4 UPDATE TRACK ADDRESS
C4 READ INTO OPERATION
Y
N
01 B3
D4 RESET IGNORE FACTOR
F1
RB665 F4 SET ENDFILE ERROR CODE
01 G5
RB672 H4 READ IGNORE OPERATION
Y
N
01 F4
J4 SET TRANSMIT ERROR
01 G5
A5 RB650
RB650 B5 NO RECORD FOUND
Y
N
RB660 E5 NON DATA-TRANSFER ERROR
Y
N
F5 OUTSIDE EXTENT LIMITS
Y
N
RB670 H5 TRANSMIT CONDITION
Y
N
RB675 J5 SET UNIDENTIFIABLE I/O ERROR CODE
01 G5

Chart DRBY. Regional(3) sequential unbuffered input/update transmitter (part 2 of 2)

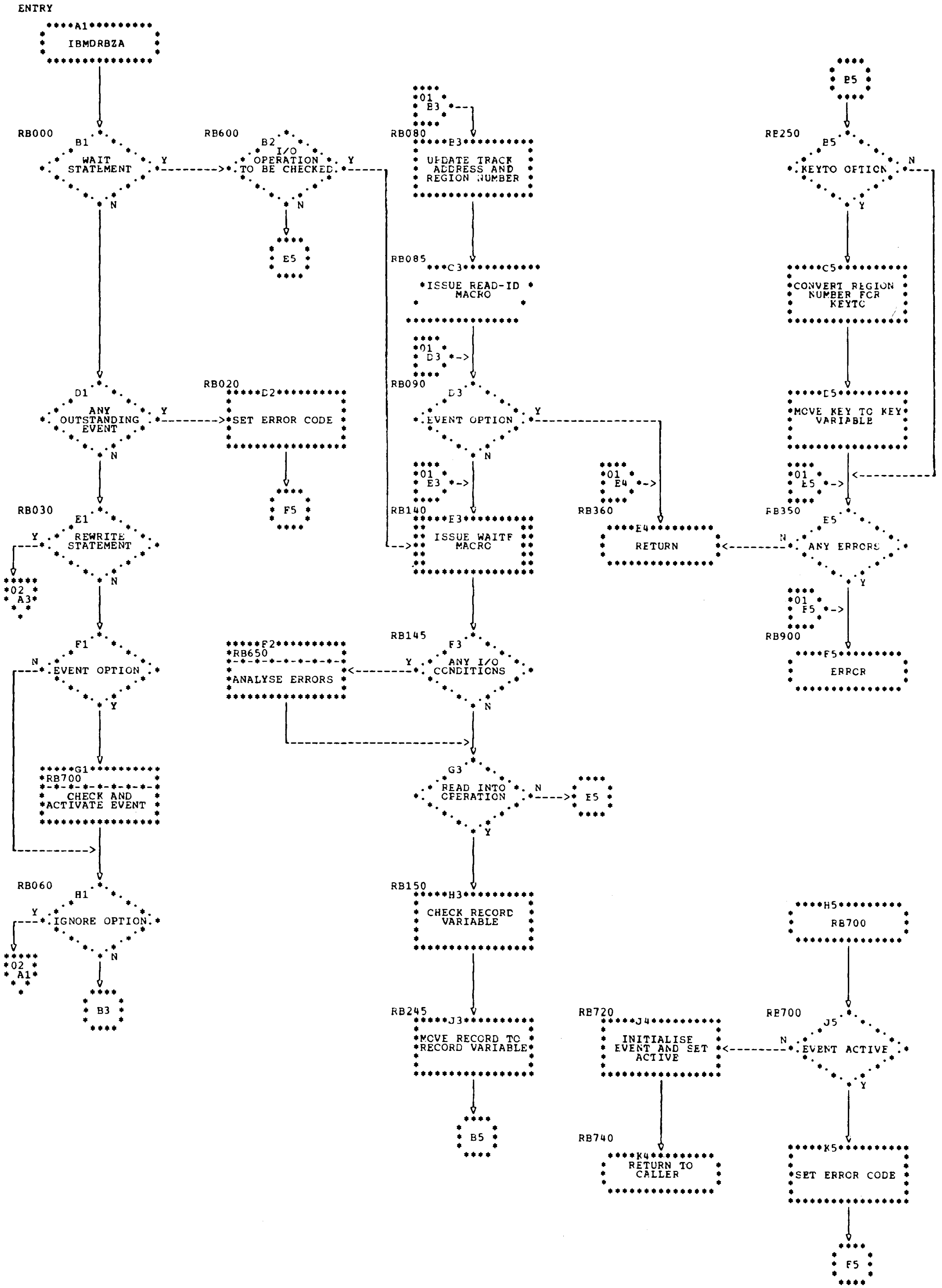


Chart DRBZ. Regional(1) sequential unbuffered input/update transmitter (part 1 of 2)

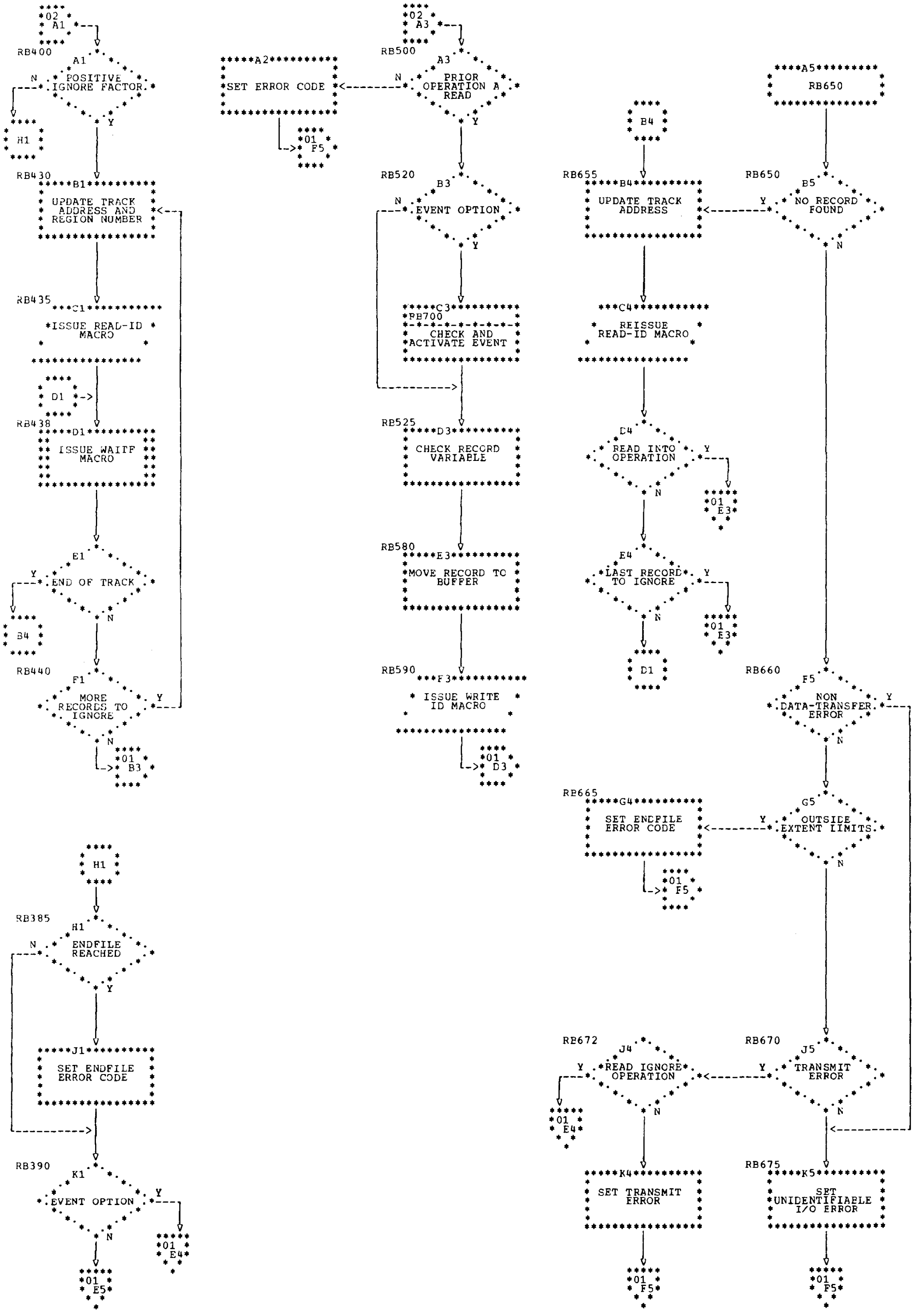


Chart DPBZ. Regional(1) sequential unbuffered input/update transmitter (part 2 of 2)

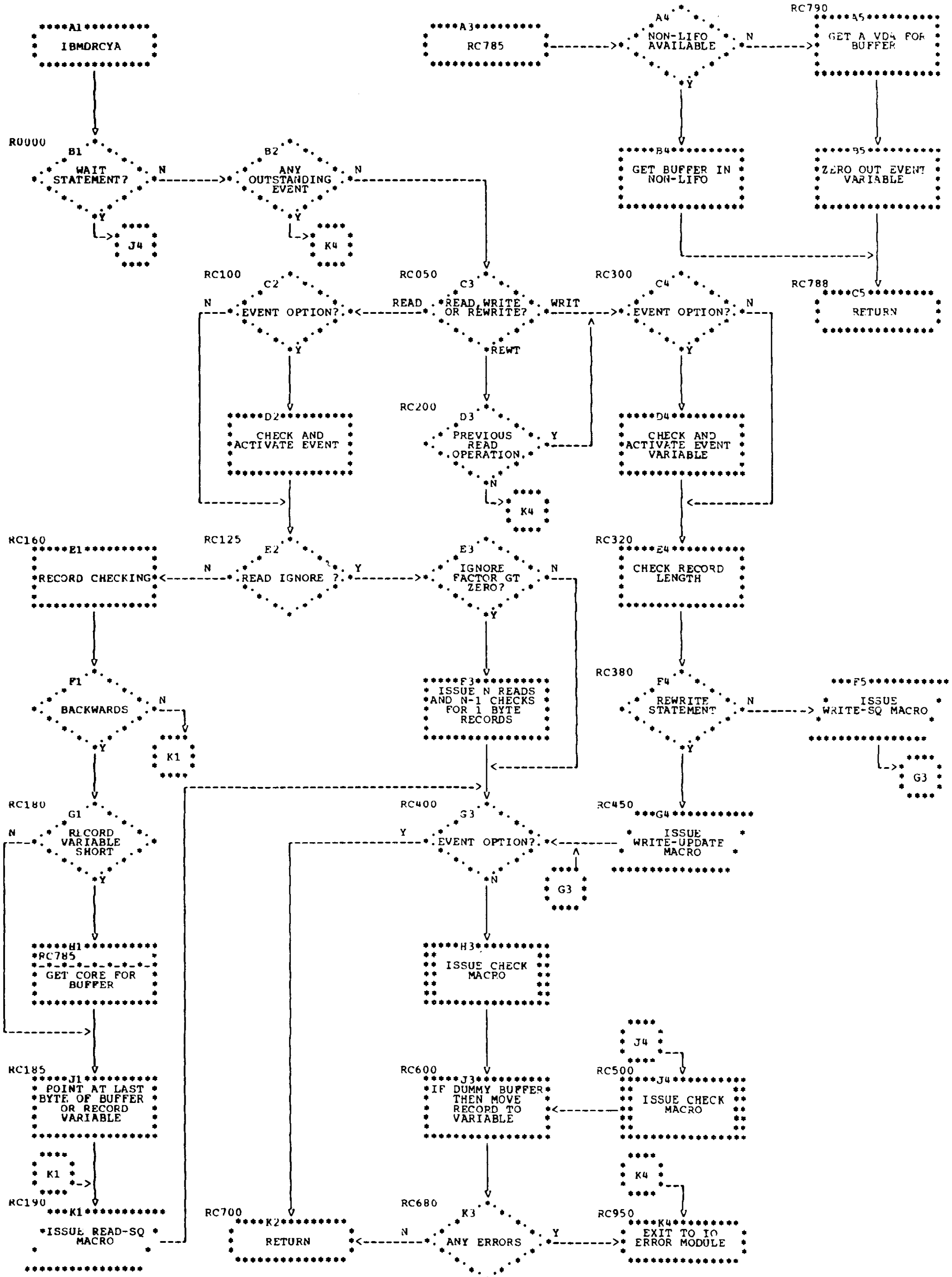


Chart DRCY. Consecutive sequential unbuffered U-format transmitter

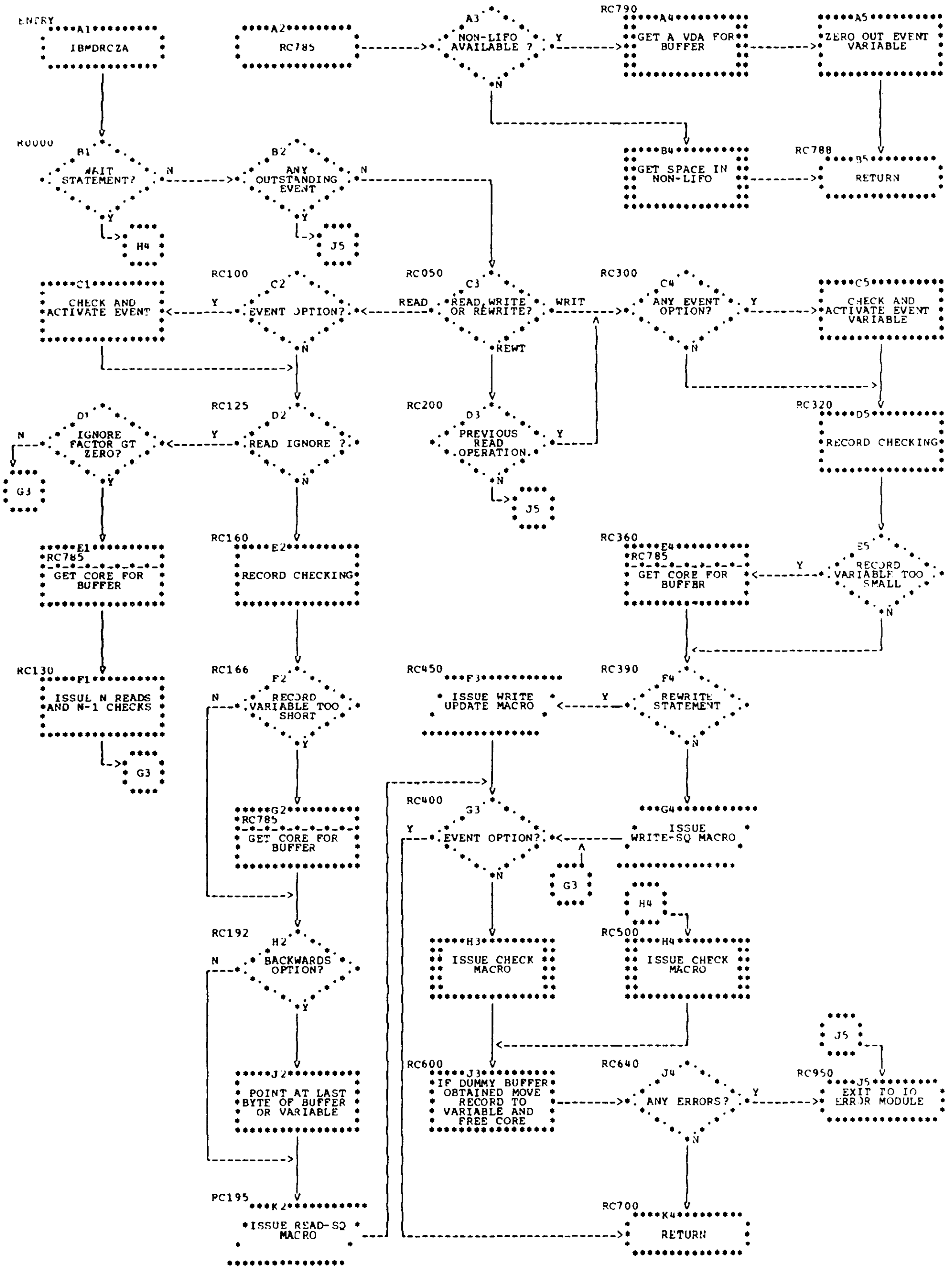


Chart DRCZ. Consecutive sequential unbuffered F-format transmitter

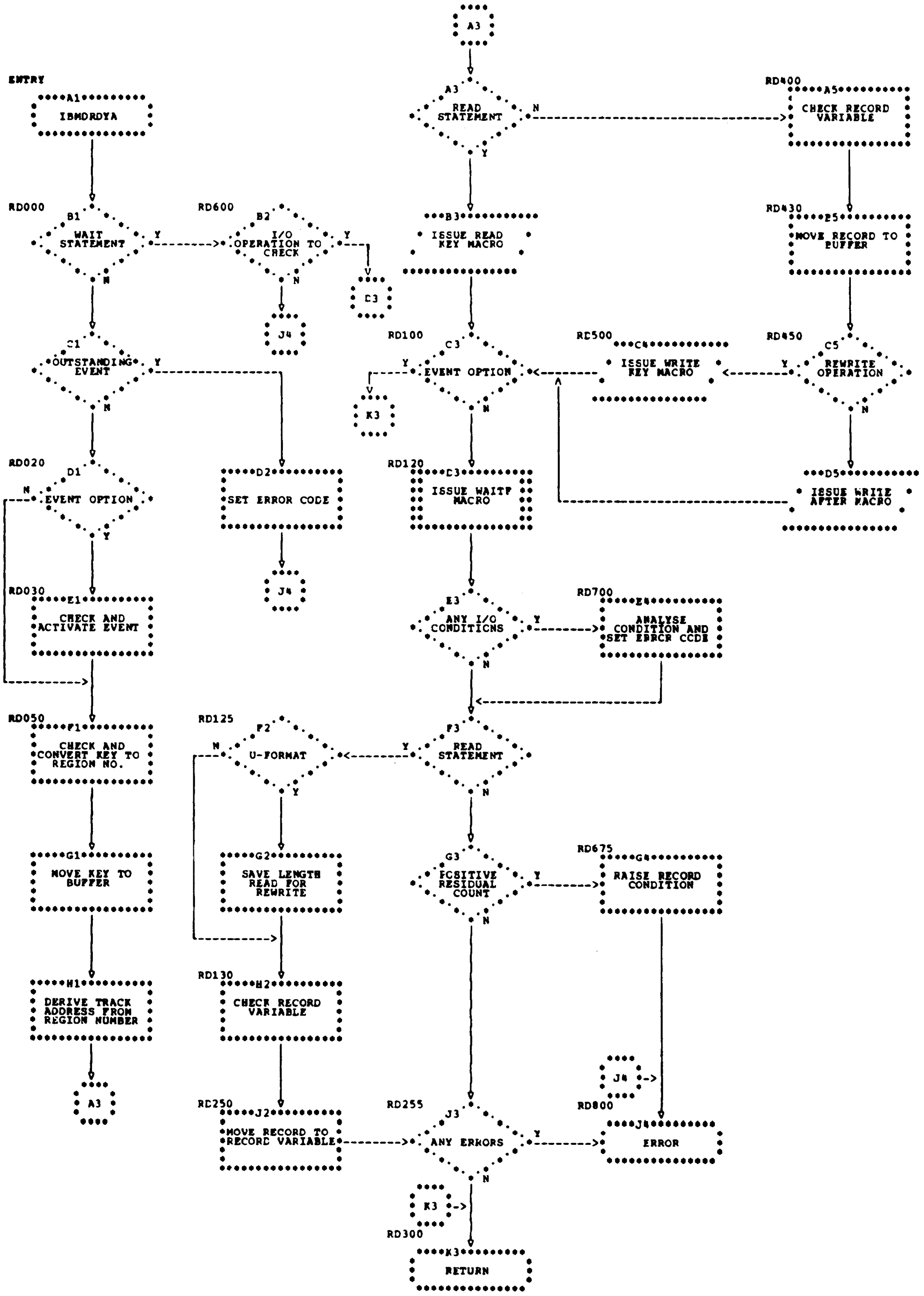


Chart DRDY. Regional(3) direct transmitter

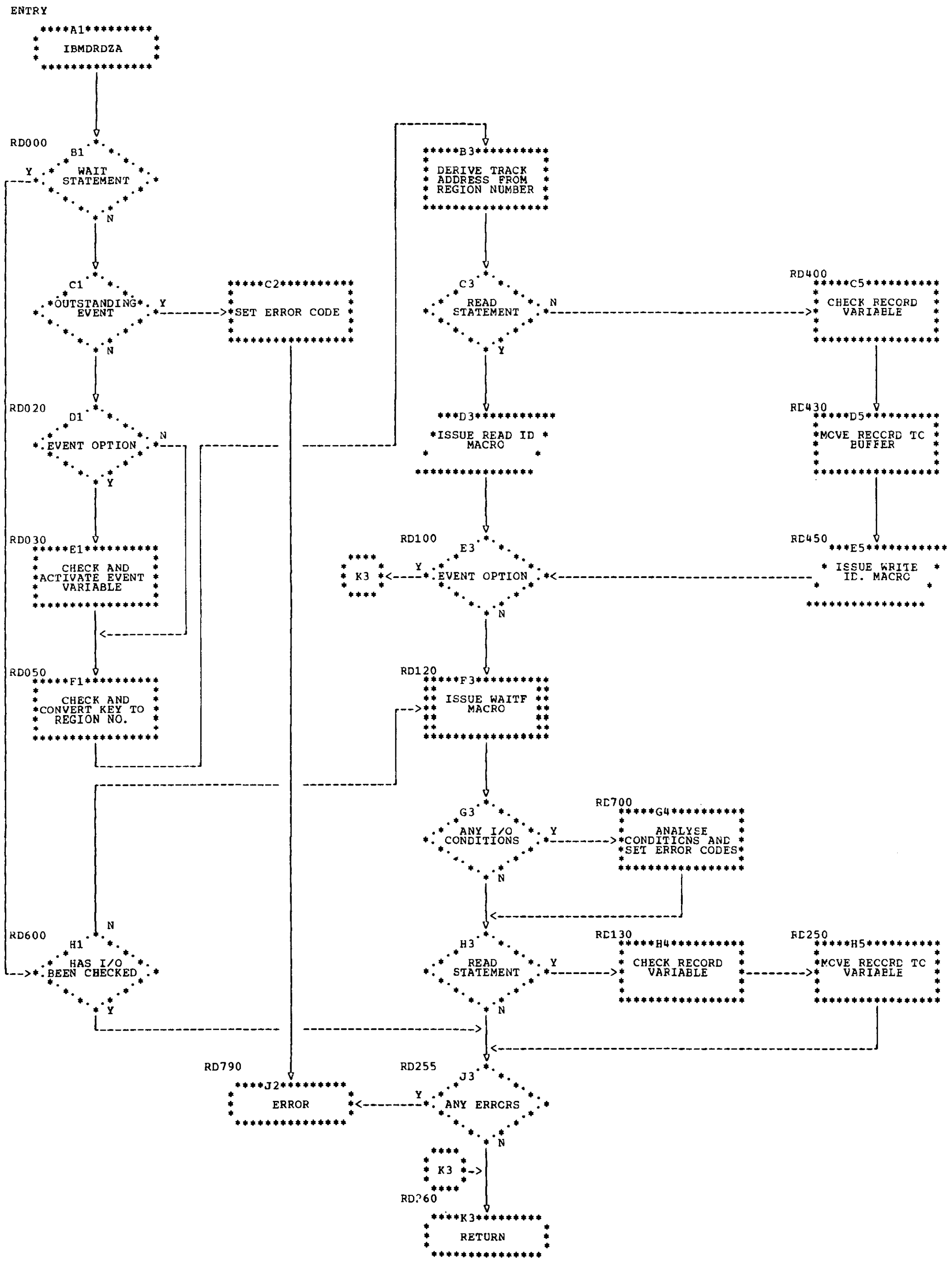


Chart DRDZ. Regional(1) direct transmitter

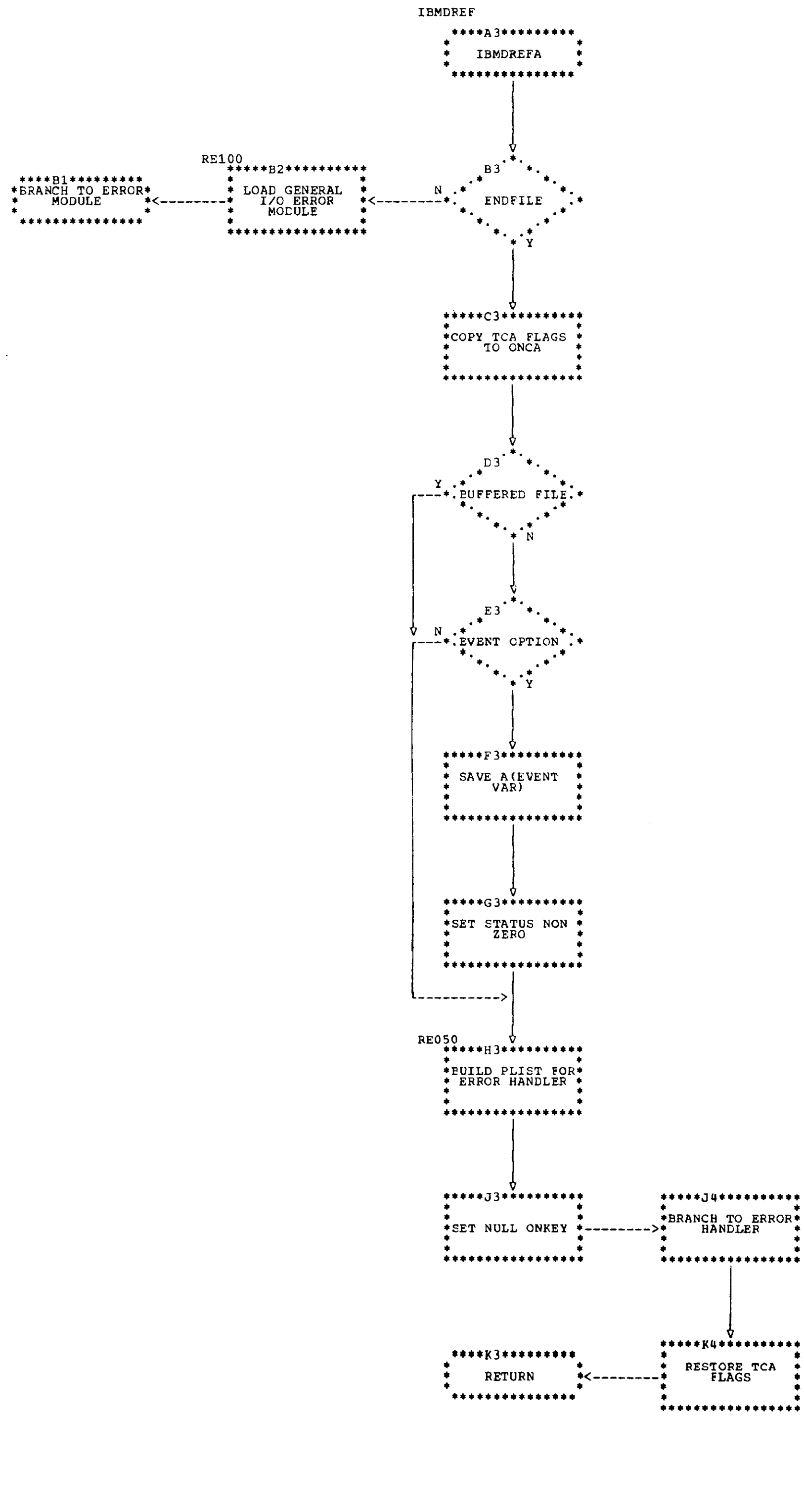
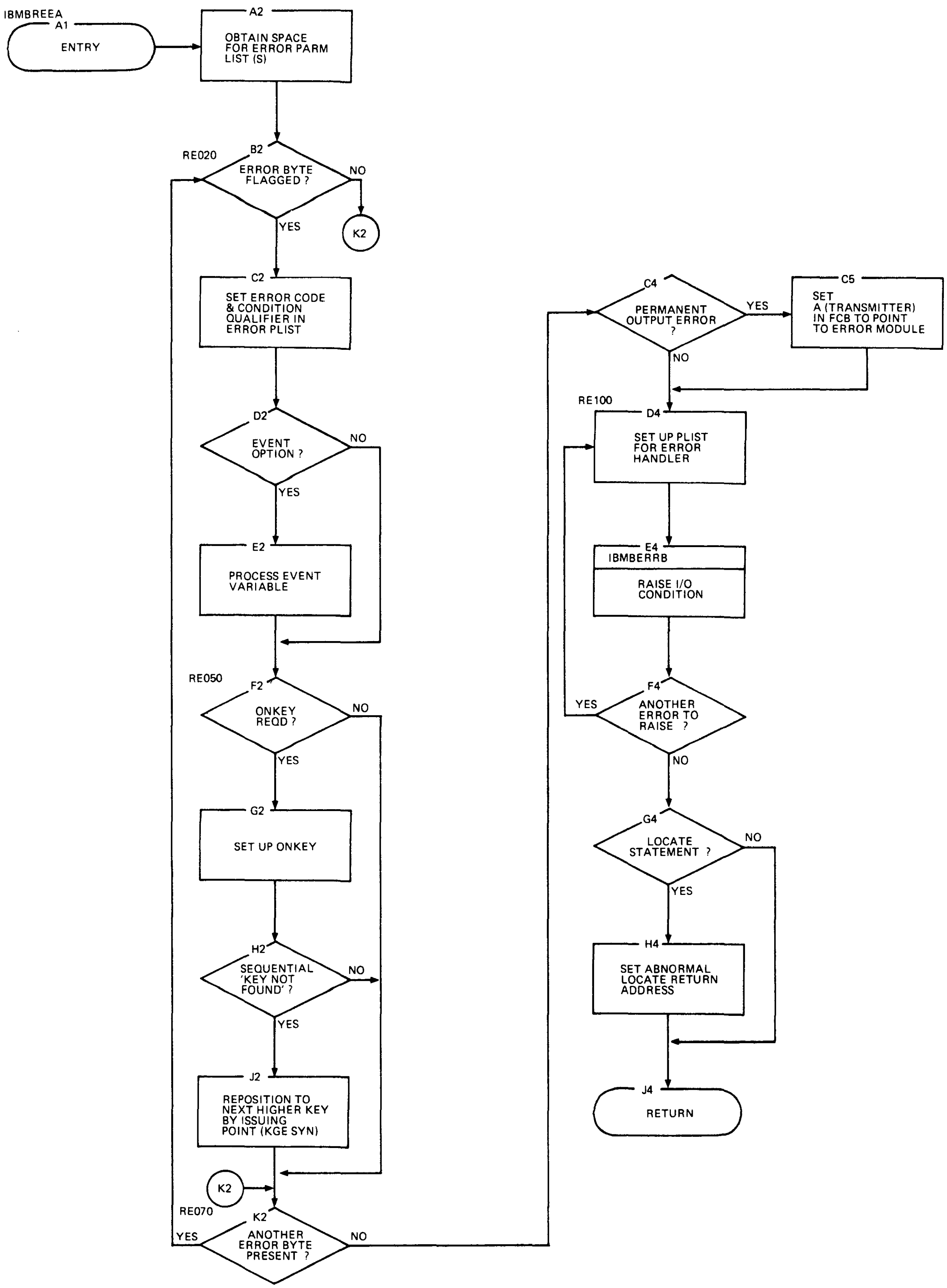


Chart DREF. ENDFILE module


IBMBREEA
A1
ENTRY
A2
OBTAIN SPACE FOR ERROR PARM LIST (S)
RE020
B2
ERROR BYTE FLAGGED ?
NO
YES
K2
C2
SET ERROR CODE & CONDITION QUALIFIER IN ERROR PLIST
D2
EVENT OPTION ?
NO
YES
E2
PROCESS EVENT VARIABLE
RE050
F2
ONKEY REQD ?
NO
YES
G2
SET UP ONKEY
H2
SEQUENTIAL 'KEY NOT FOUND' ?
NO
YES
J2
REPOSITION TO NEXT HIGHER KEY BY ISSUING POINT (KGE SYN)
K2
RE070
K2
ANOTHER ERROR BYTE PRESENT ?
YES
NO
C4
PERMANENT OUTPUT ERROR ?
YES
NO
C5
SET A (TRANSMITTER) IN FCB TO POINT TO ERROR MODULE
RE100
D4
SET UP PLIST FOR ERROR HANDLER
E4
IBMBERRB
RAISE I/O CONDITION
F4
ANOTHER ERROR TO RAISE ?
YES
NO
G4
LOCATE STATEMENT ?
NO
YES
H4
SET ABNORMAL LOCATE RETURN ADDRESS
J4
RETURN

|Chart DREV. Error module for VSAM files

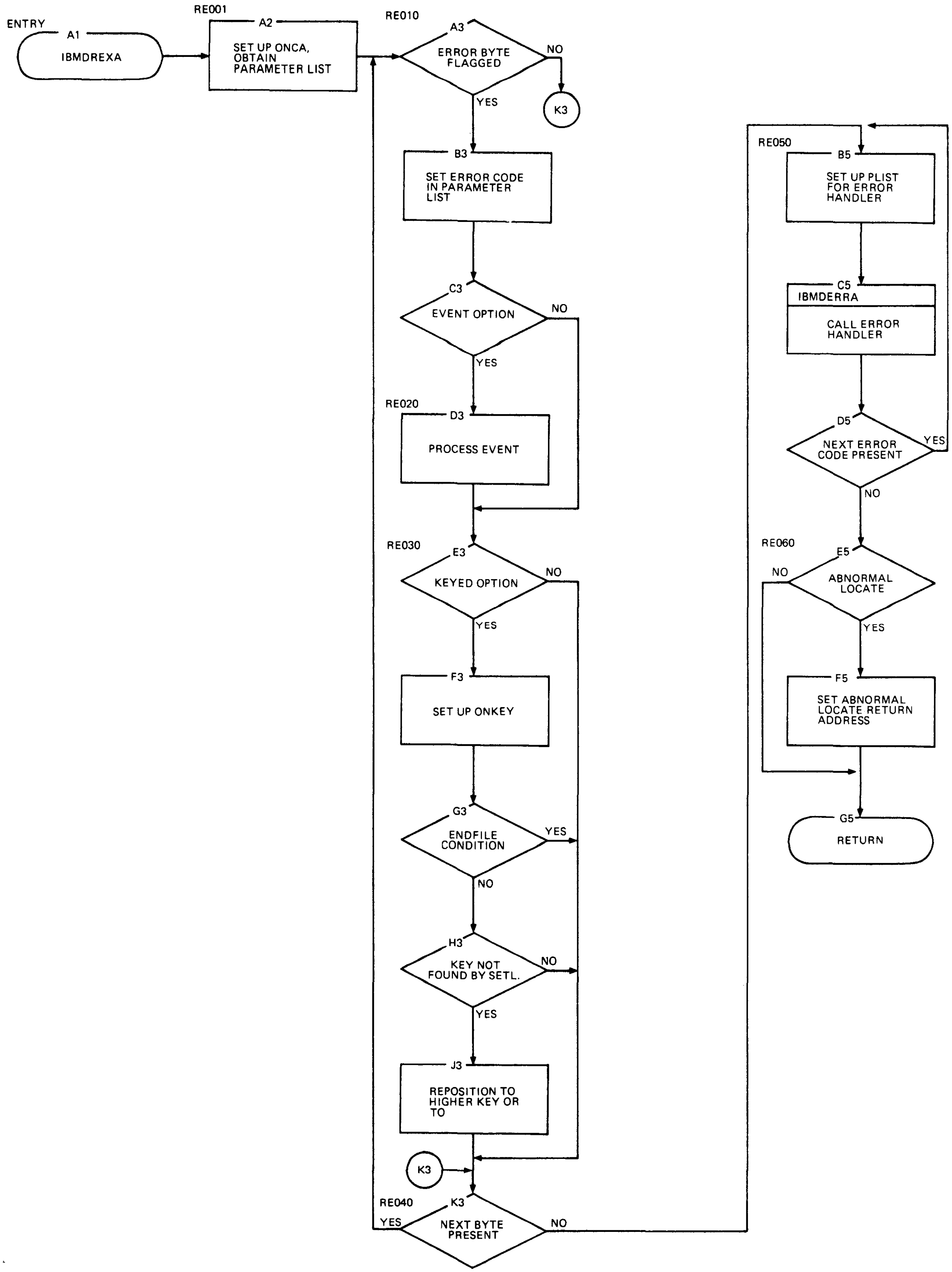


Chart DREX. Error handler for indexed sequential files

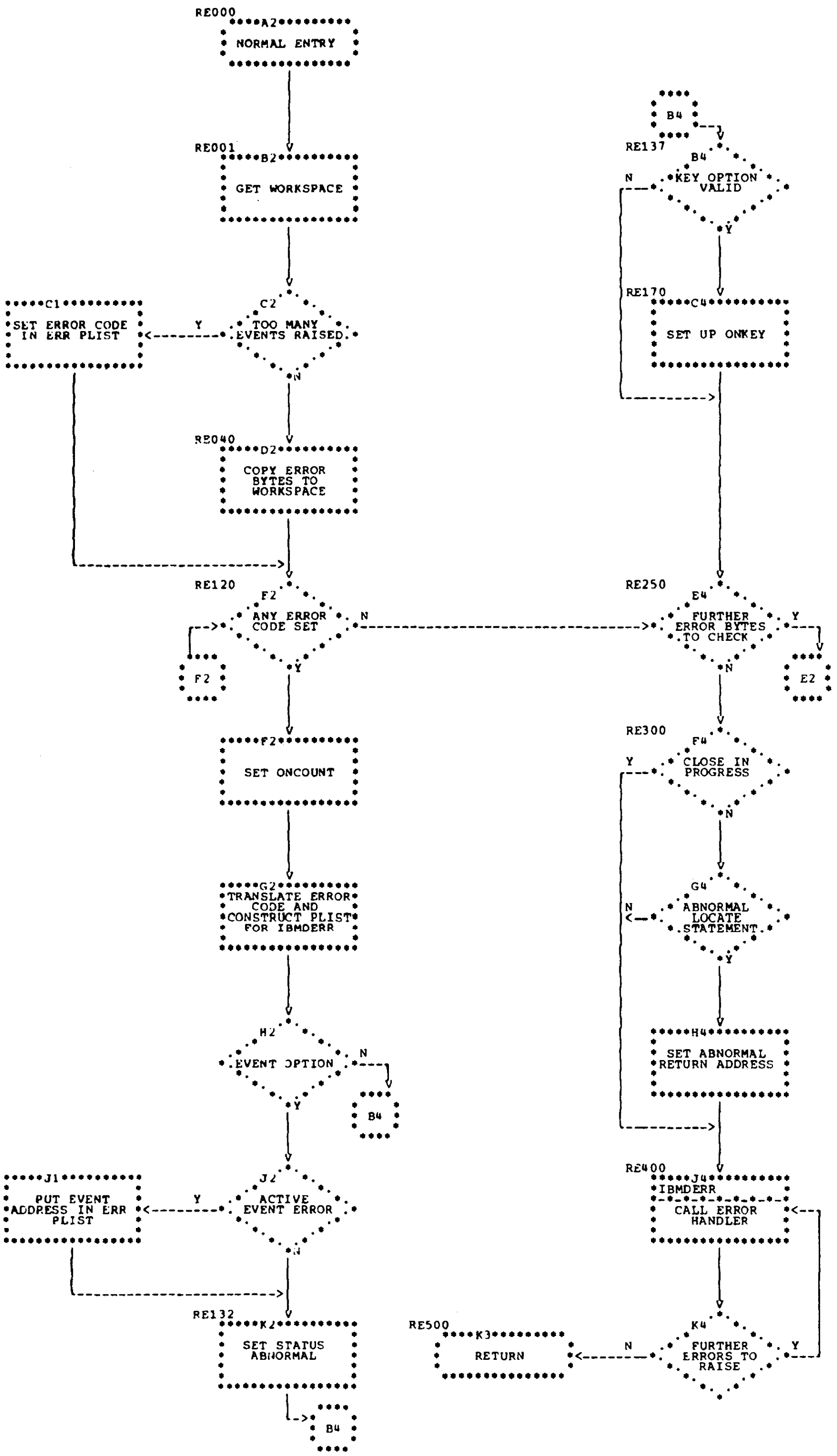


Chart DREY. Error handler for regional and unbuffered consecutive files

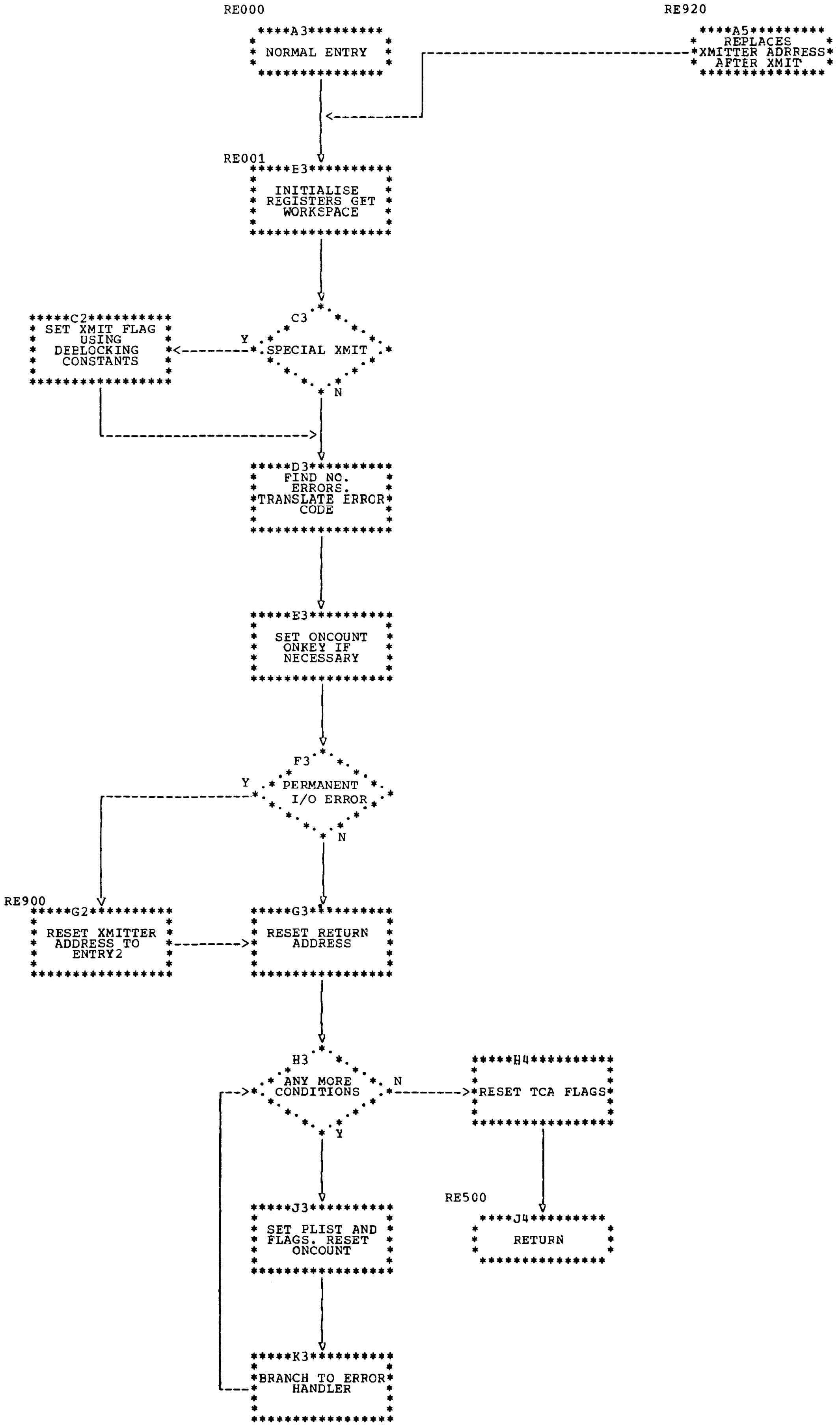


Chart DRLZ. Error handler for buffered consecutive files

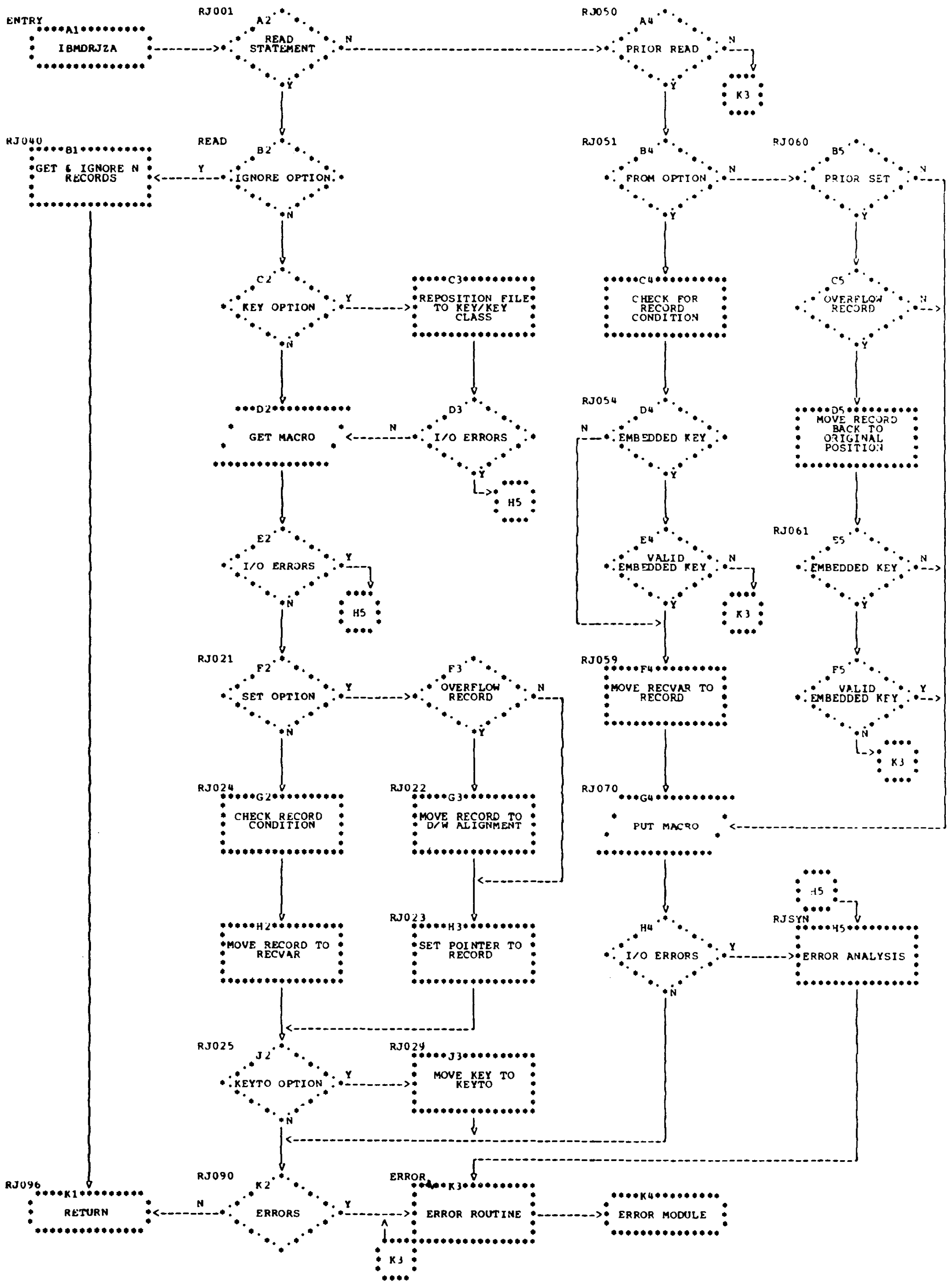


Chart DRJZ. Indexed sequential input/update transmitter

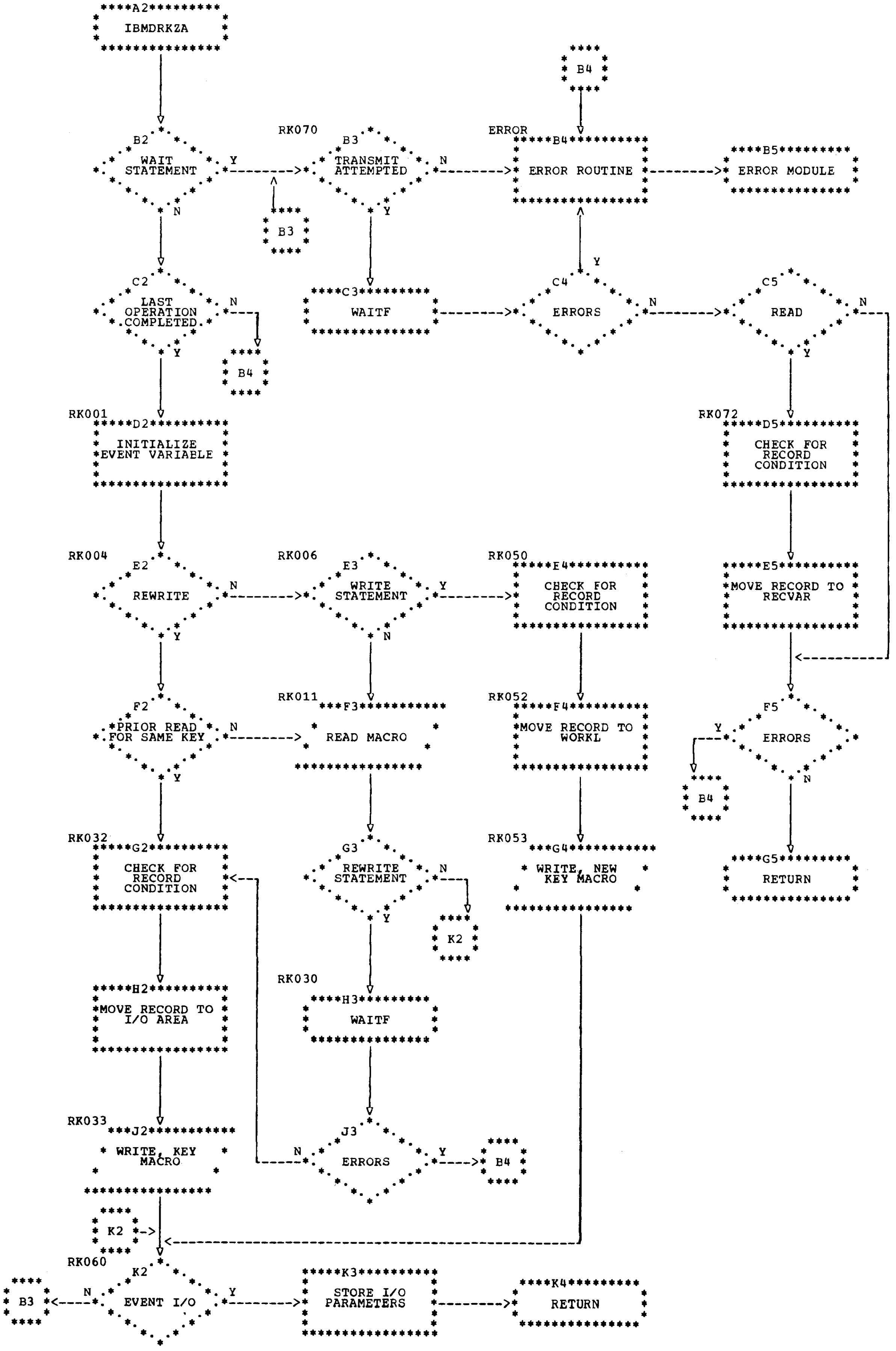


Chart DRKZ. Indexed direct input/update transmitter

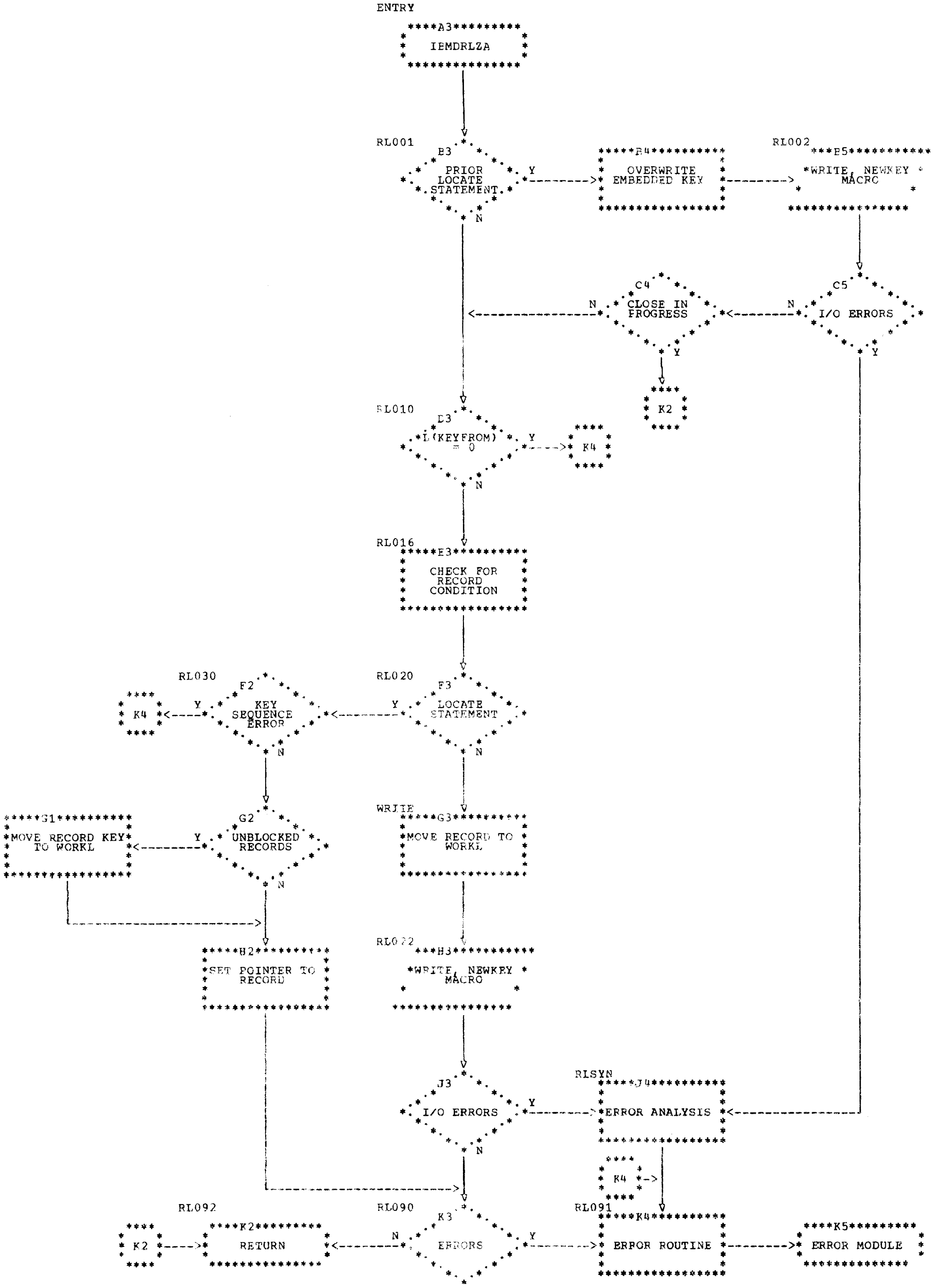


Chart DHLZ. Indexed sequential output transmitter

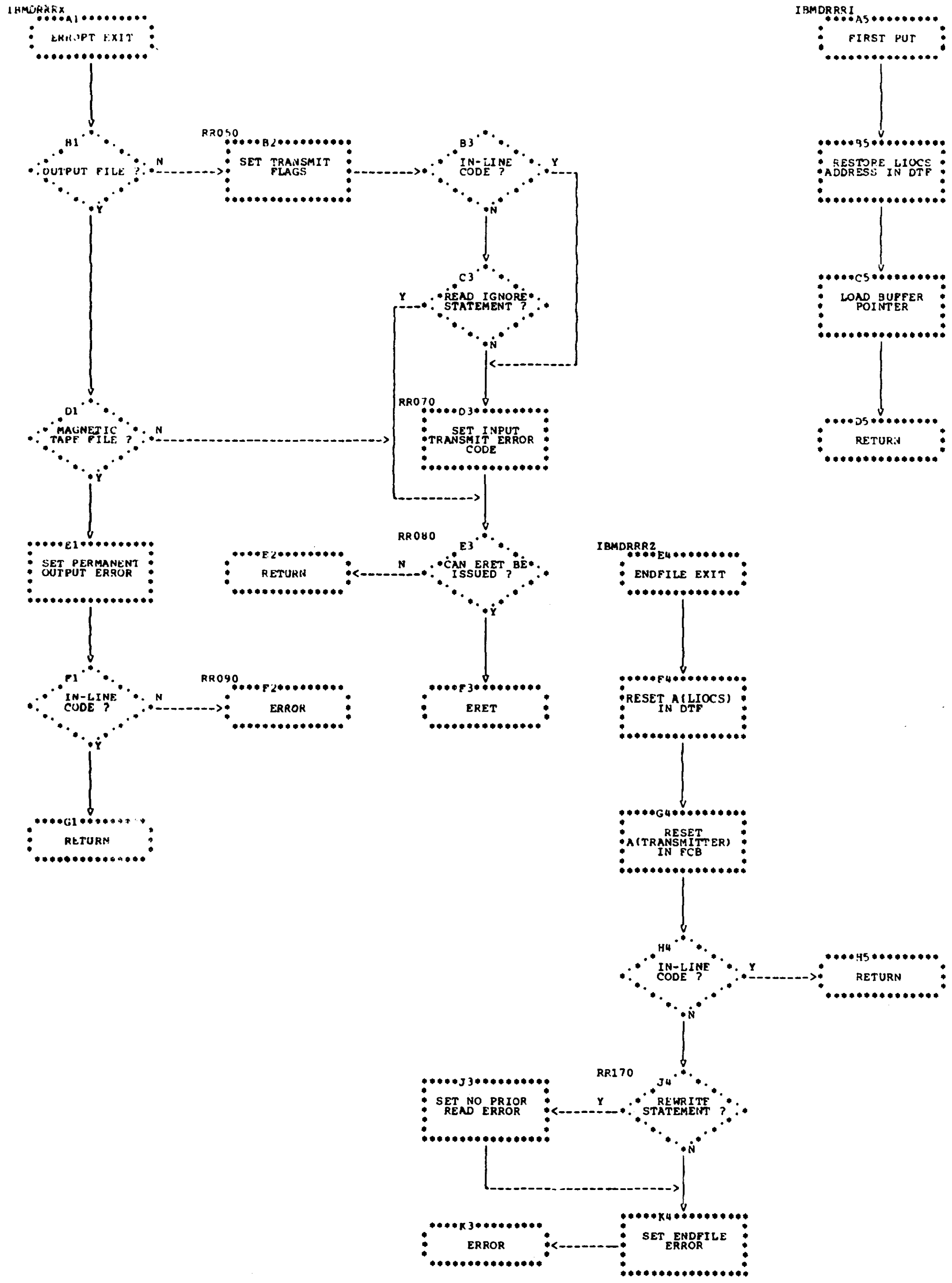


Chart DRRR. Exit module

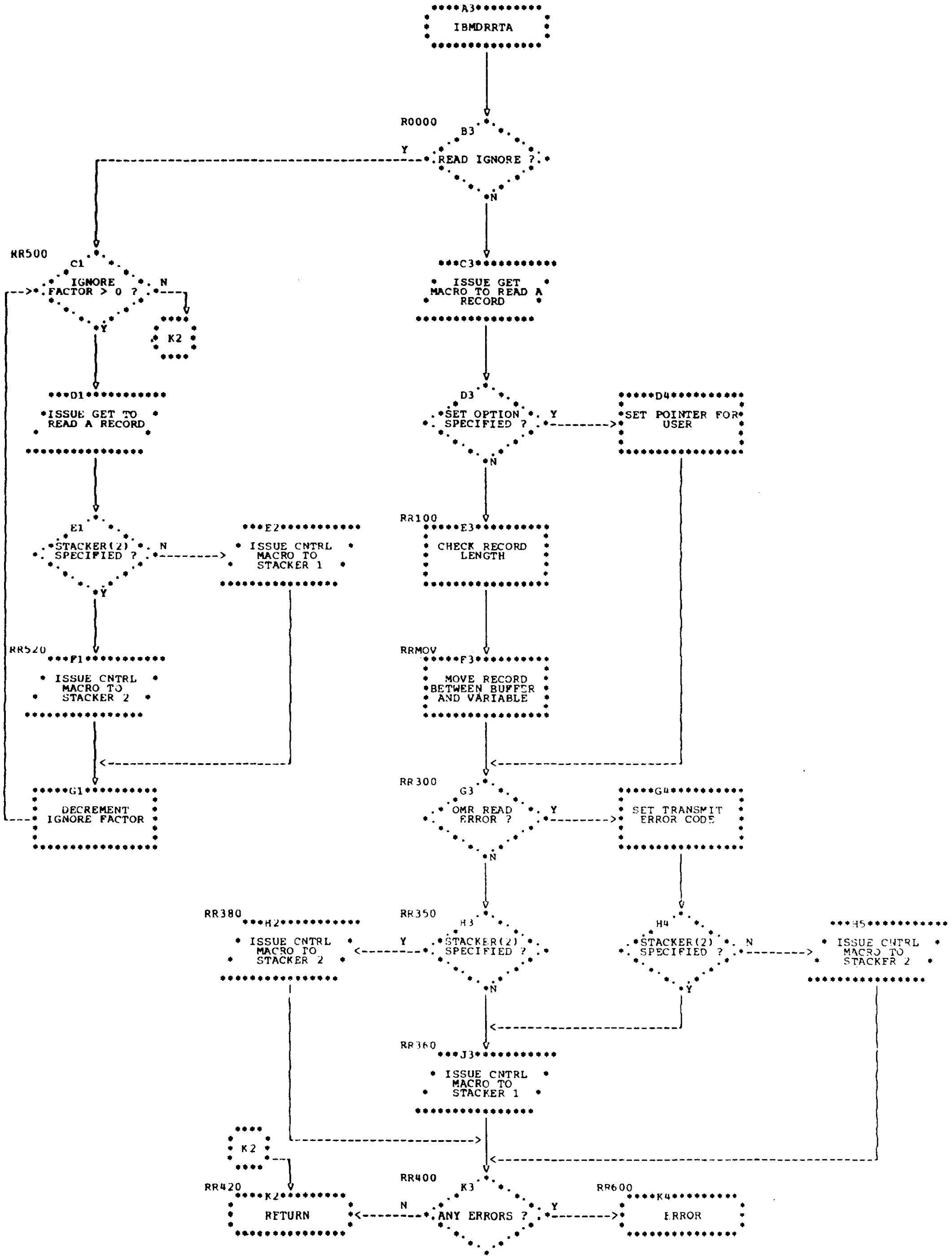


Chart DRRT. Consecutive sequential buffered F-format OMR transmitter

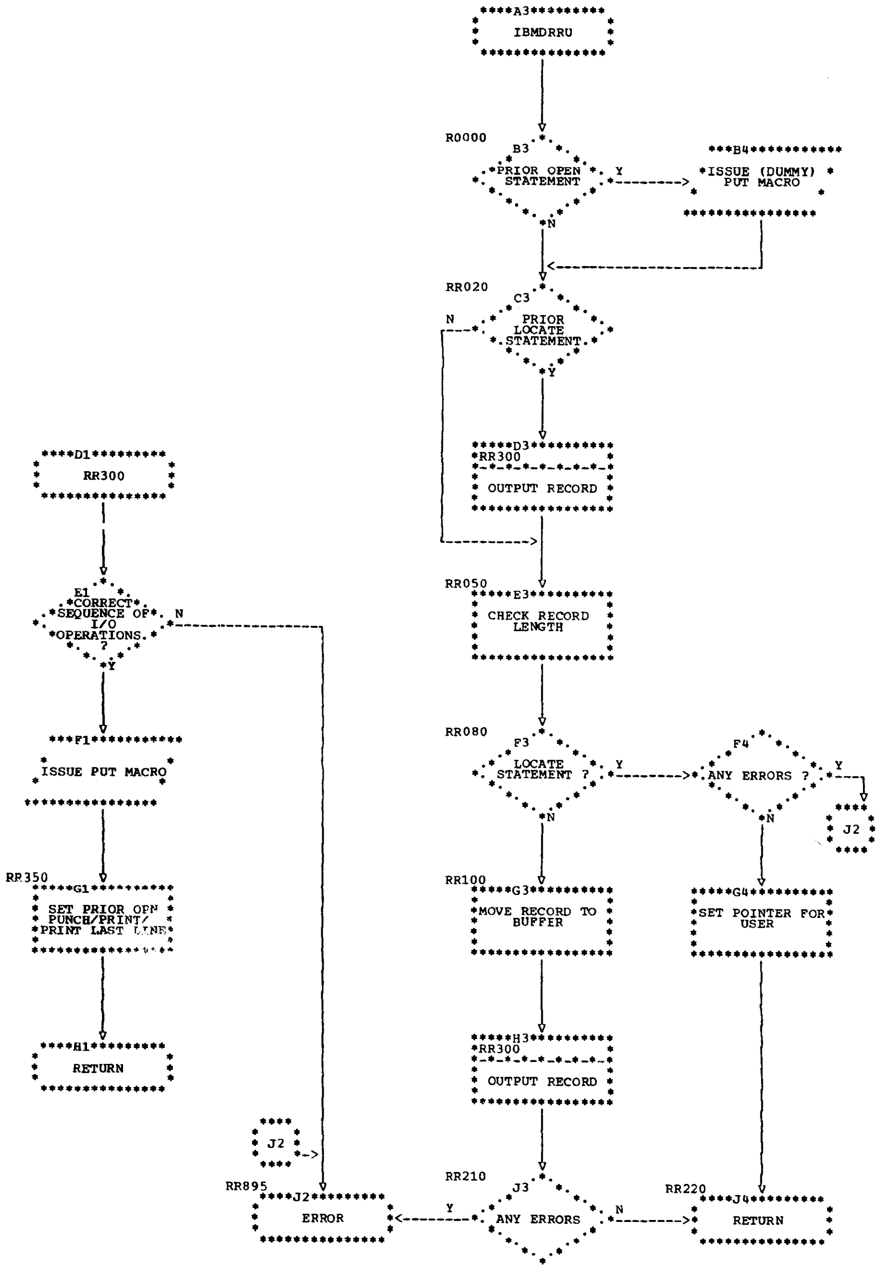


Chart DRRU. Consecutive sequential buffered associate U-format transmitter

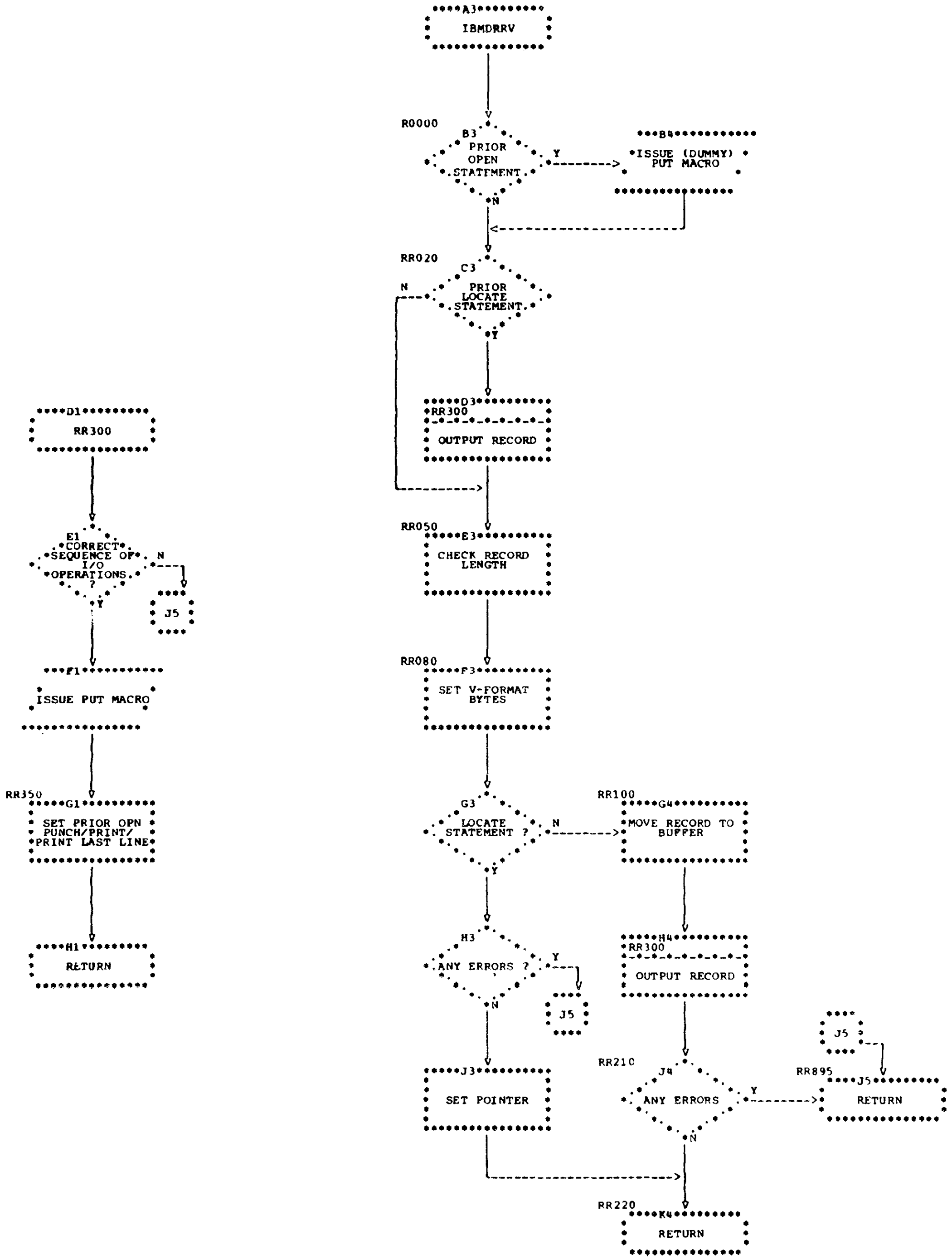


Chart DRRV. Consecutive sequential buffered associate V-format transmitter

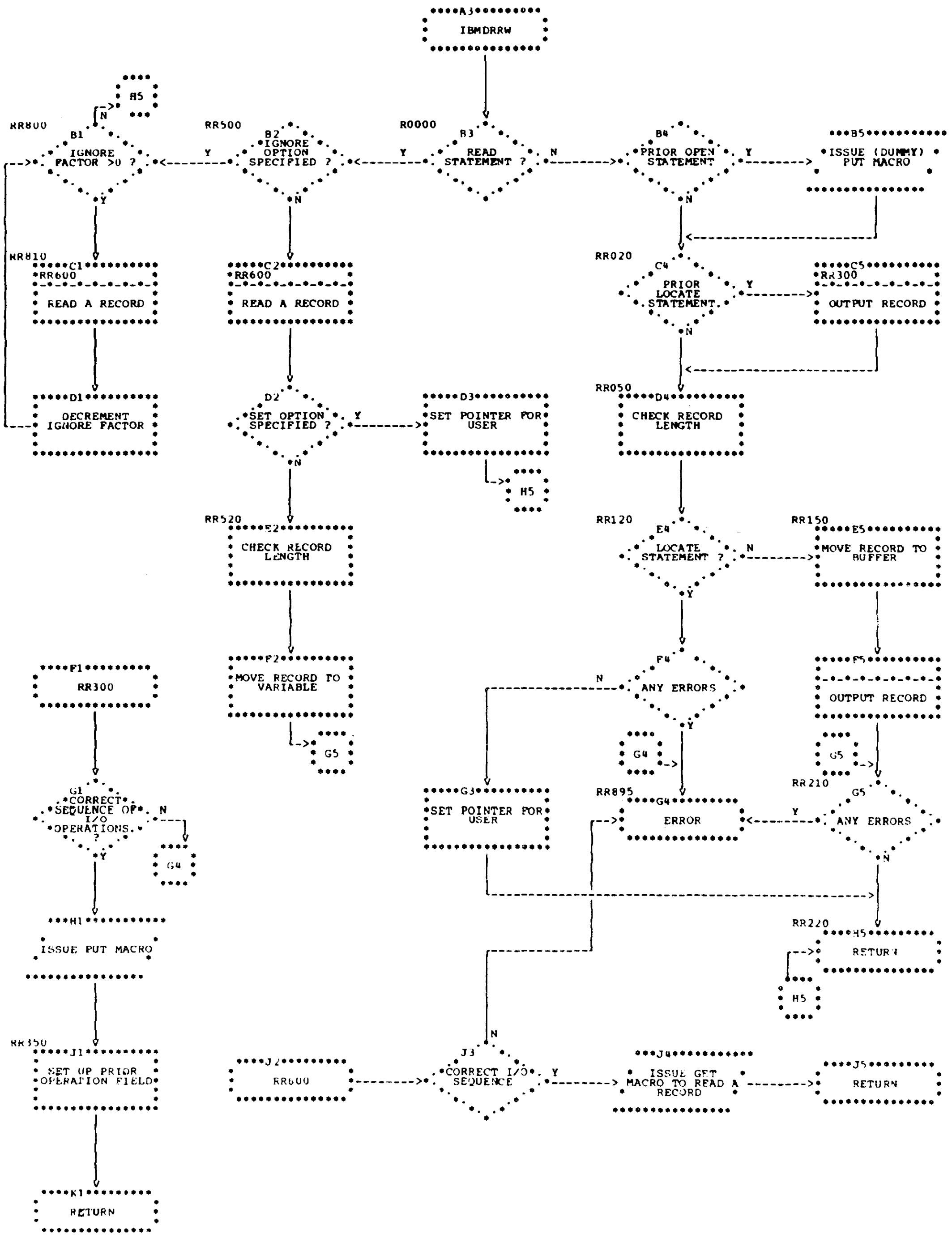


Chart DRRW. Consecutive sequential buffered associate F-format transmitter

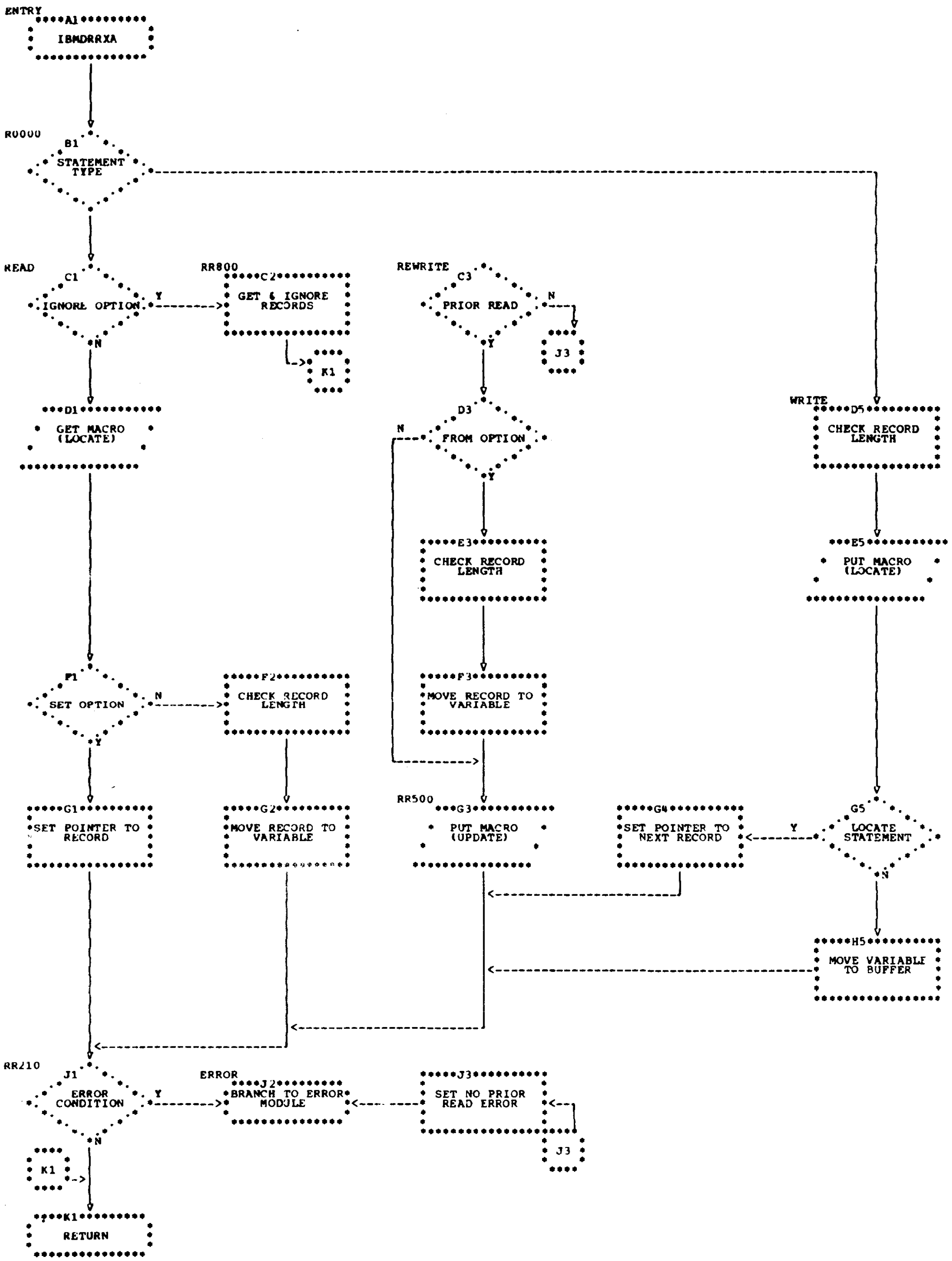


Chart DRRX. Consecutive sequential buffered U-format transmitter

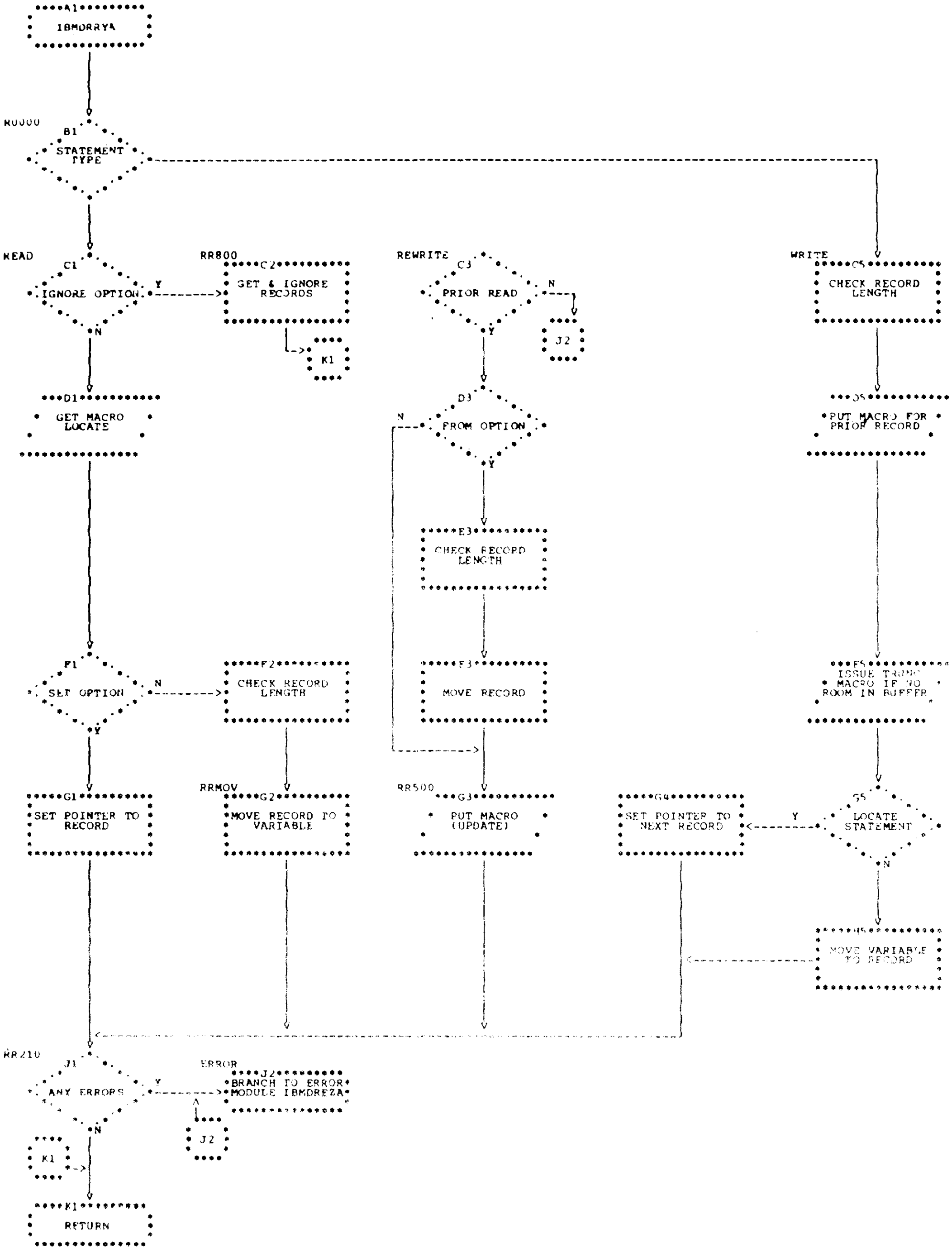


Chart DRRY. Consecutive sequential buffered V-format transmitter

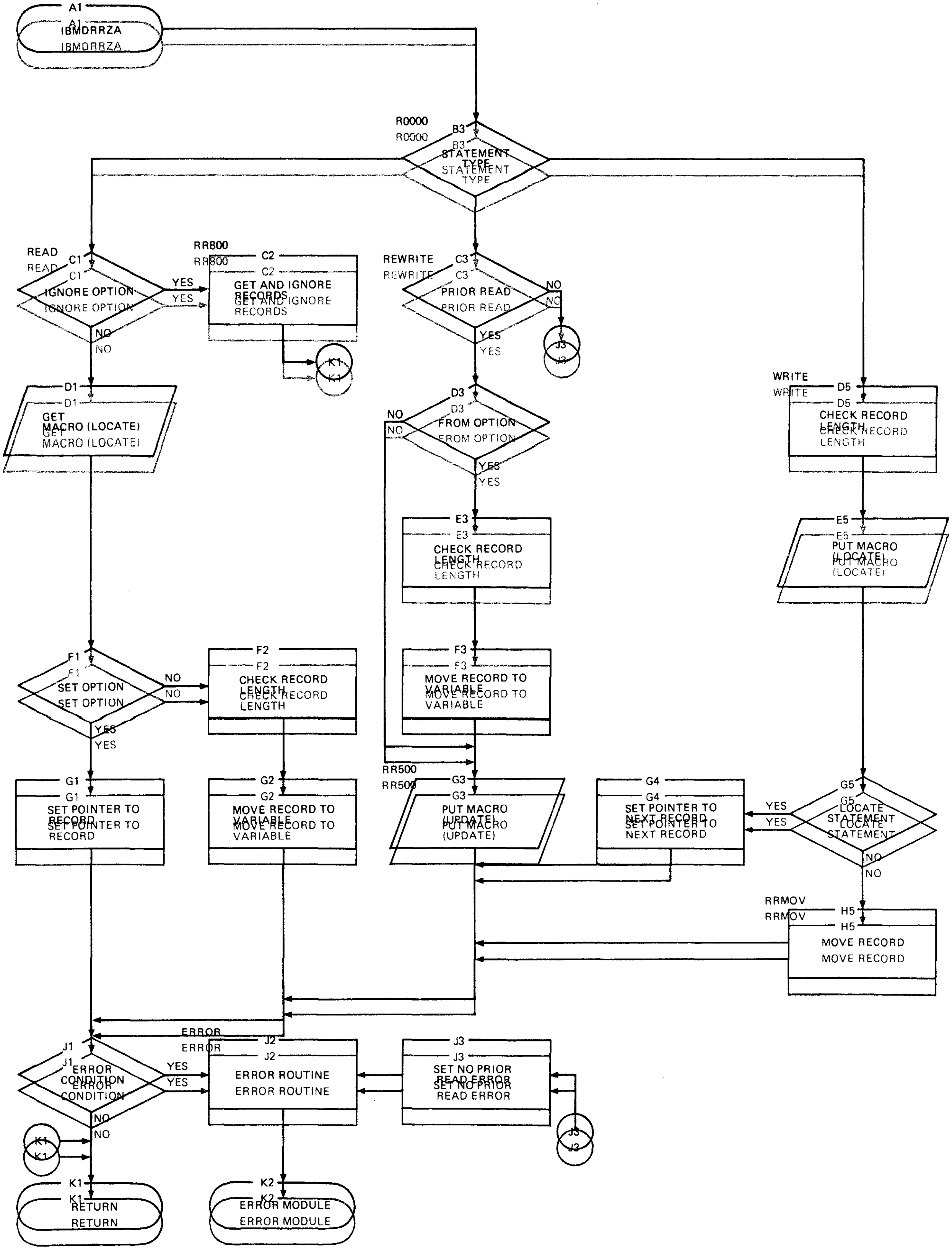
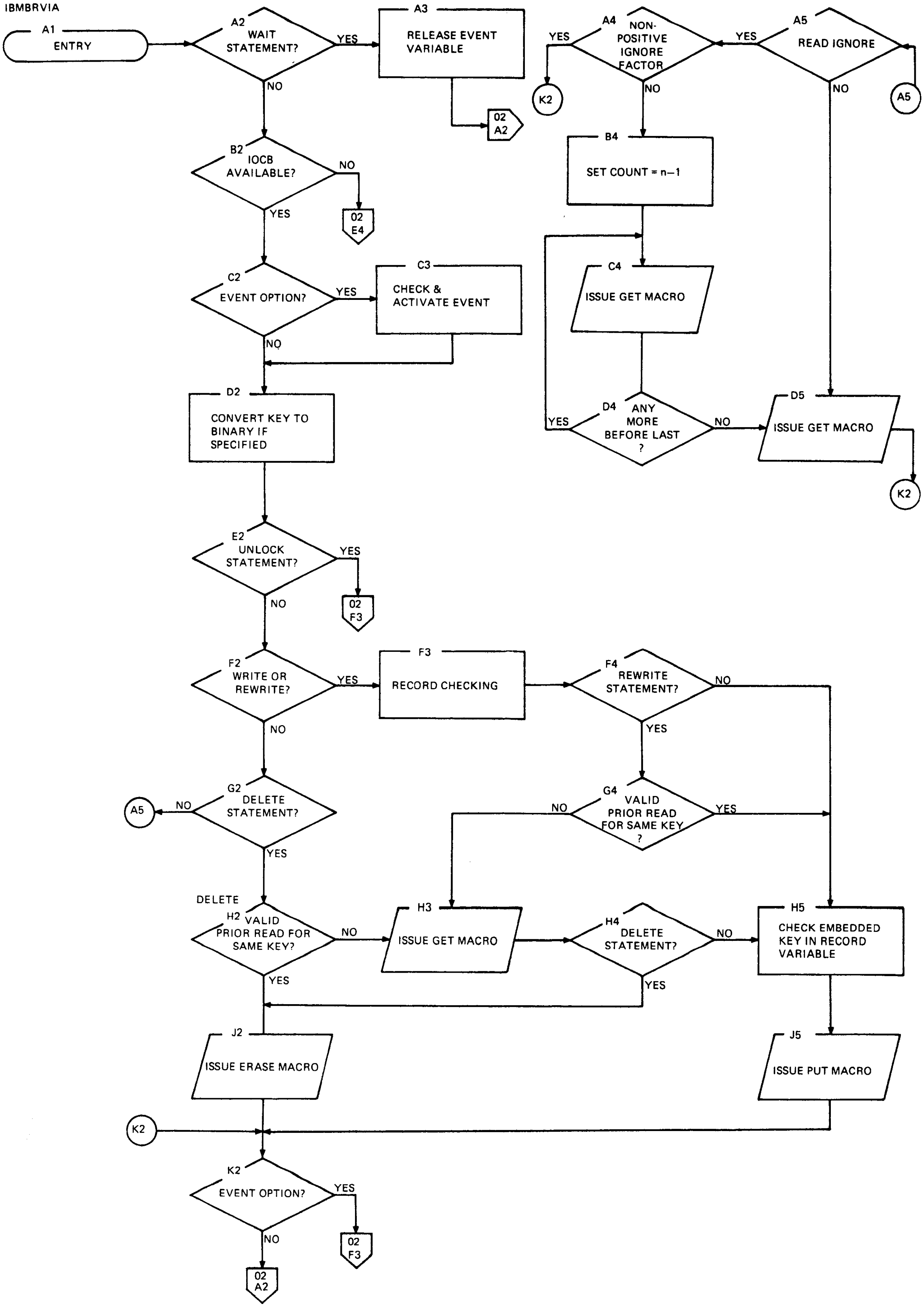
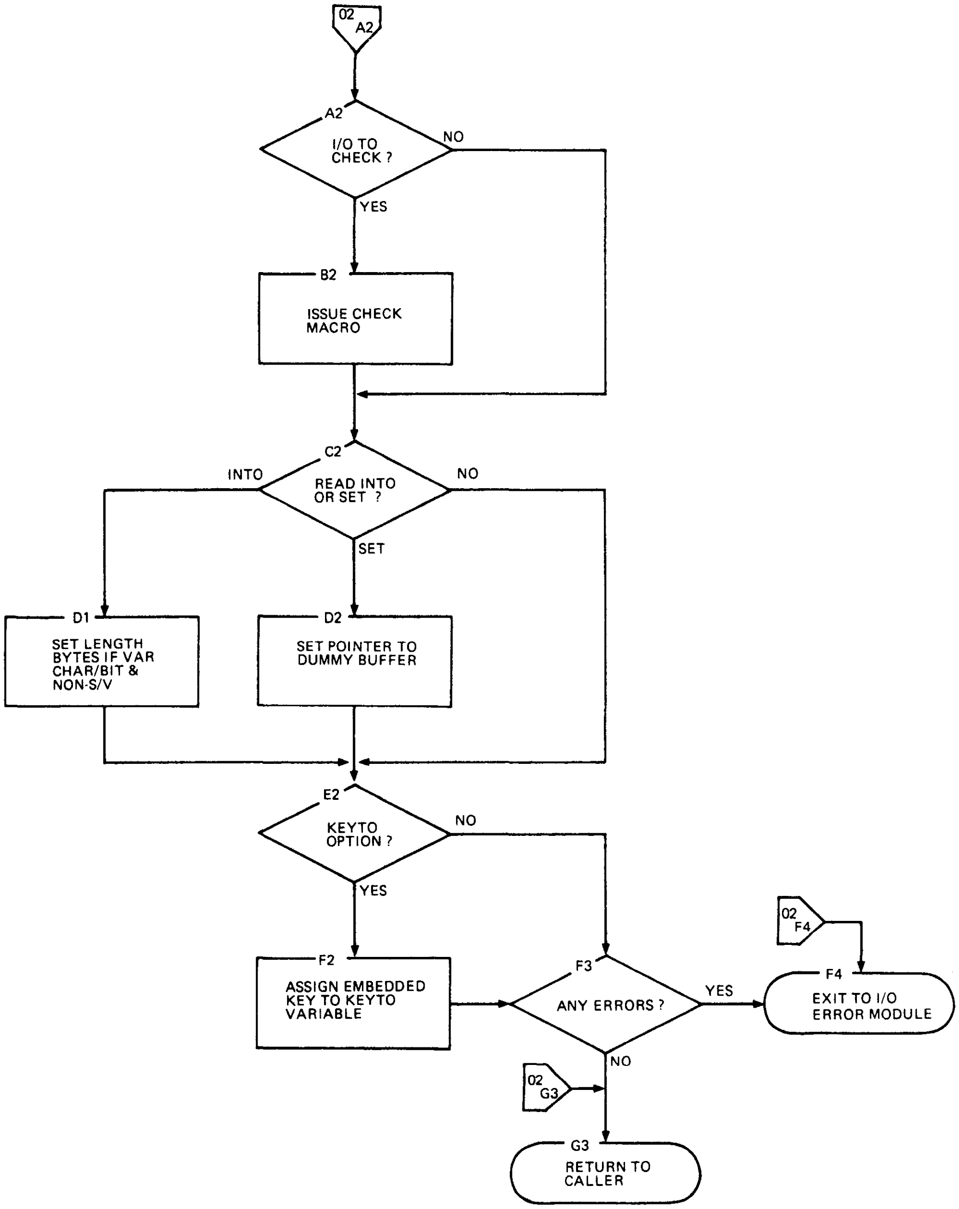


Chart DRRZ. Consecutive sequential buffered F-format transmitter

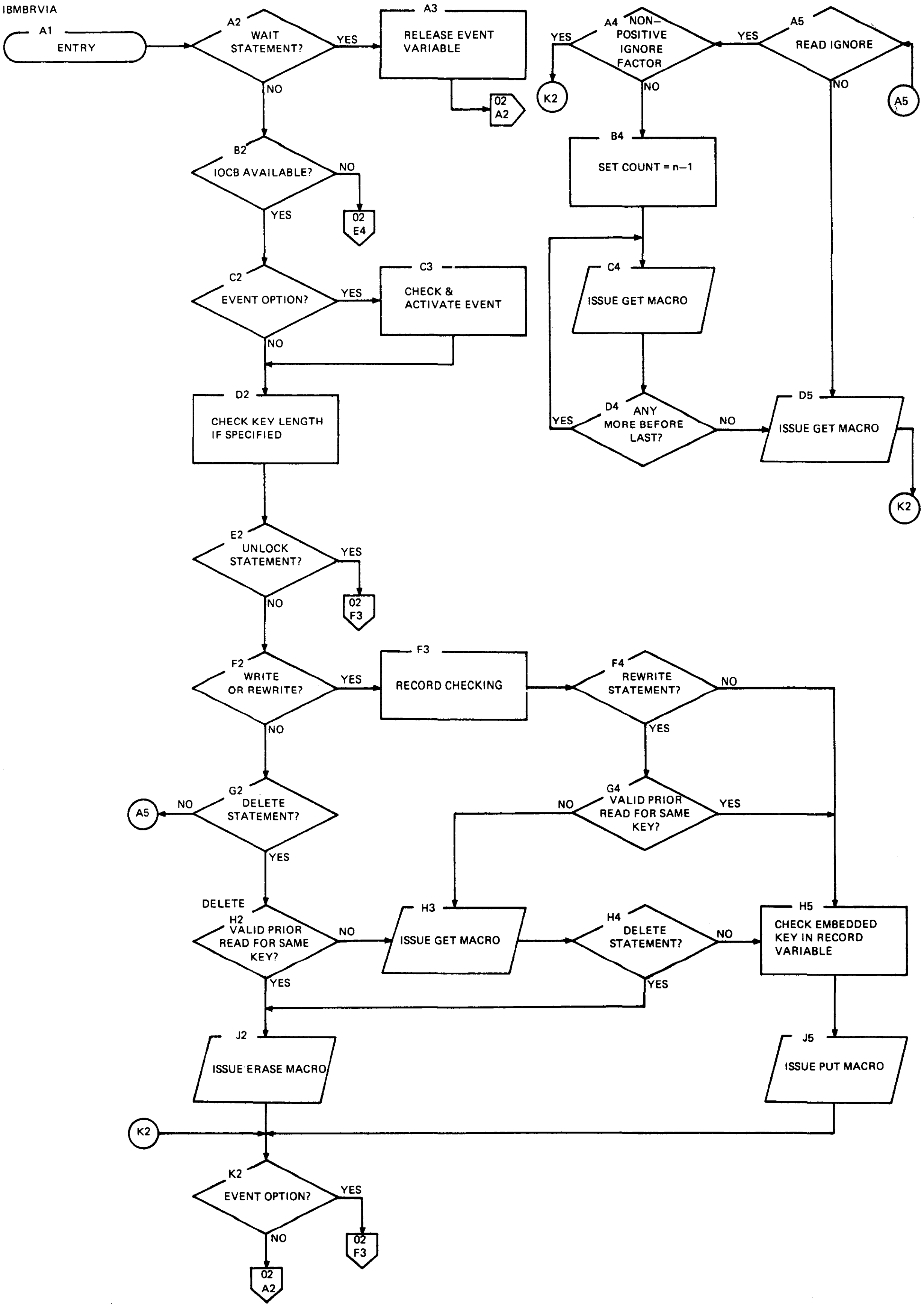
Chart DRRZ. Consecutive sequential buffered F-format transmitter



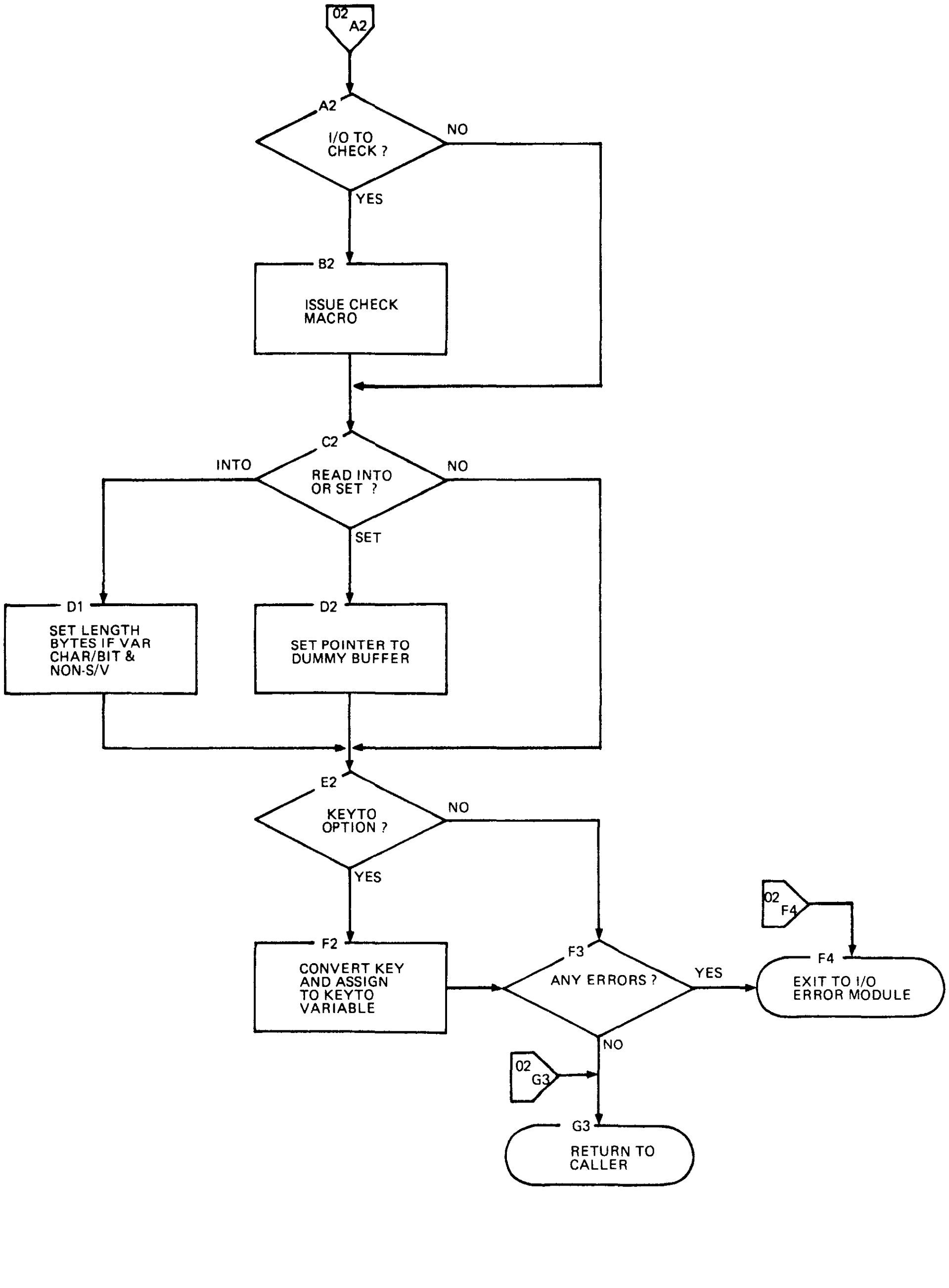
|Chart DRVR. RRDS transmitter (part 1 of 2)



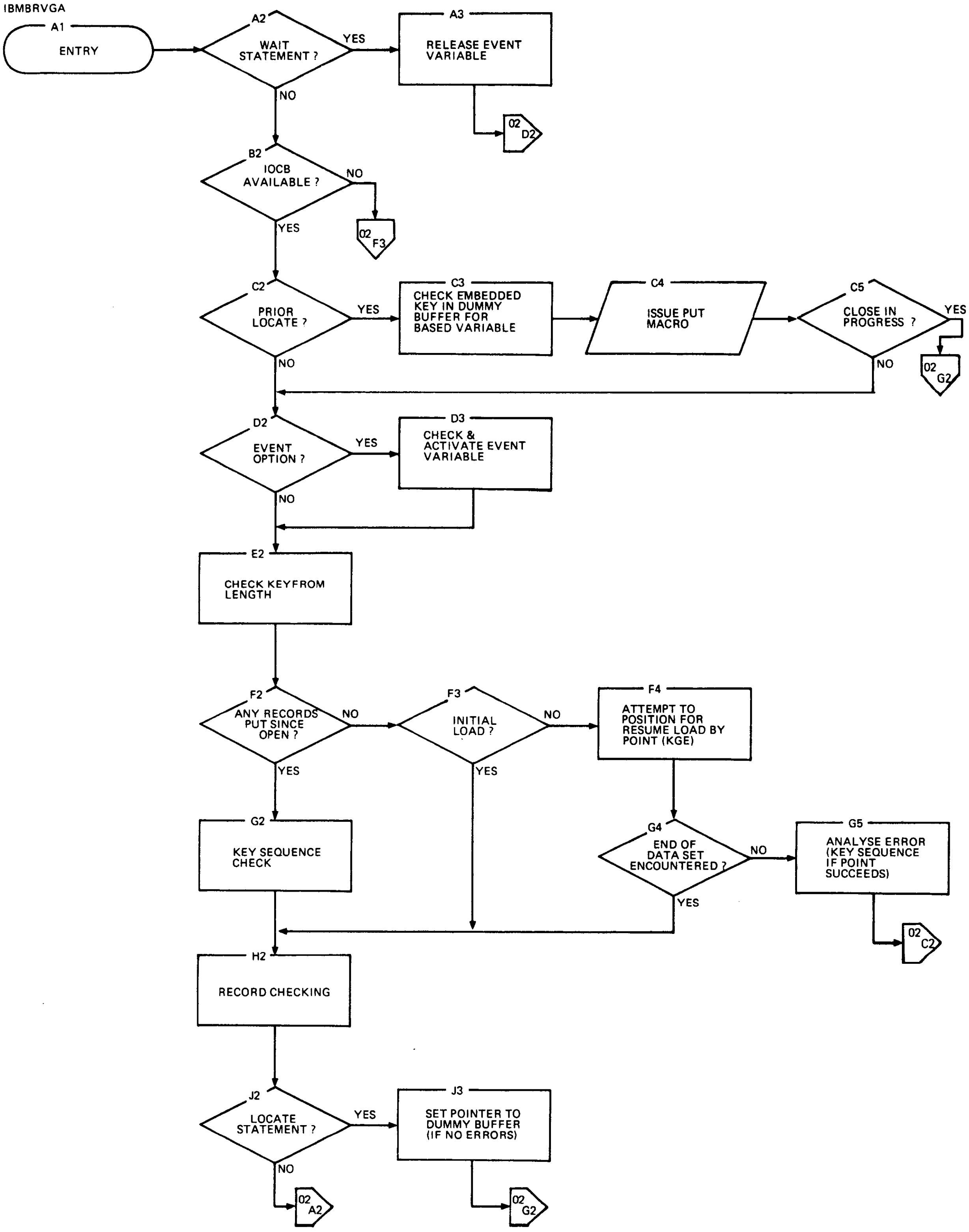
|Chart DRVR. RRDS transmitter (part 2 of 2)



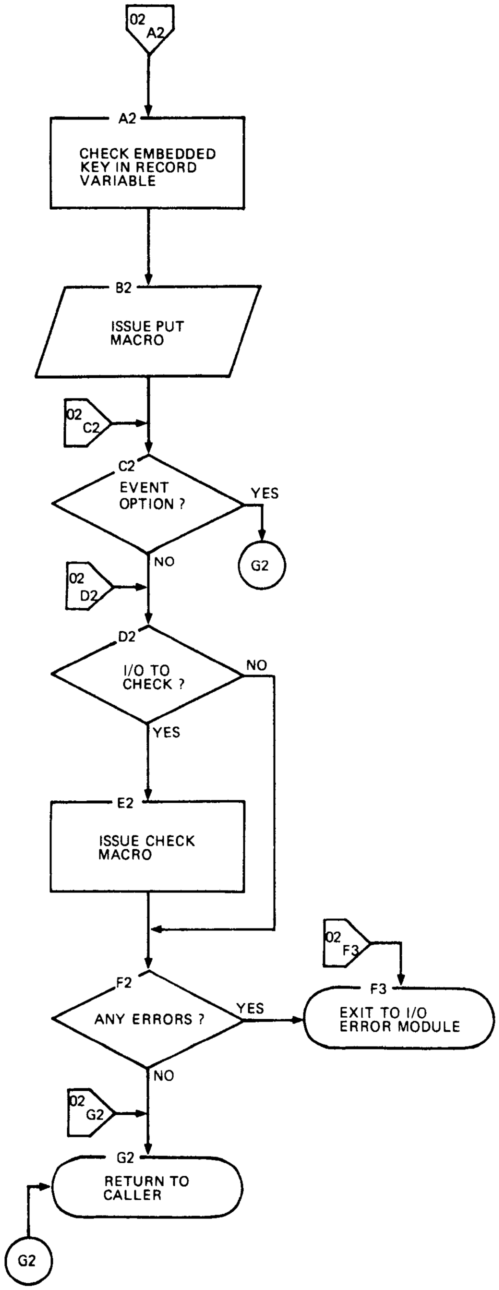
|Chart DRVS. KSDS or PATH input/update/direct transmitter (part 1 of 2)



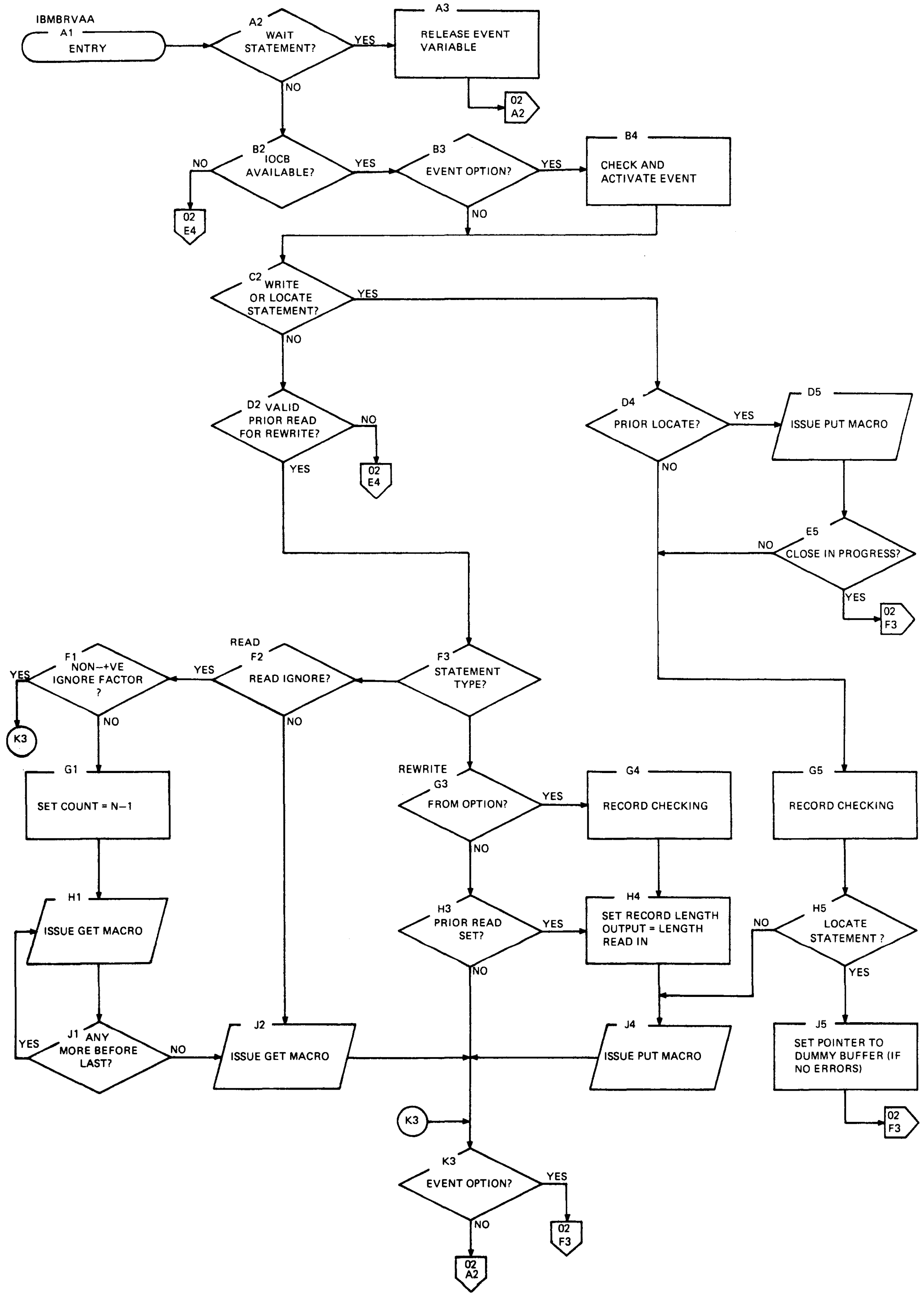
|Chart DRVS. KSDS or PATH input/update/direct transmitter (part 2 of 2)



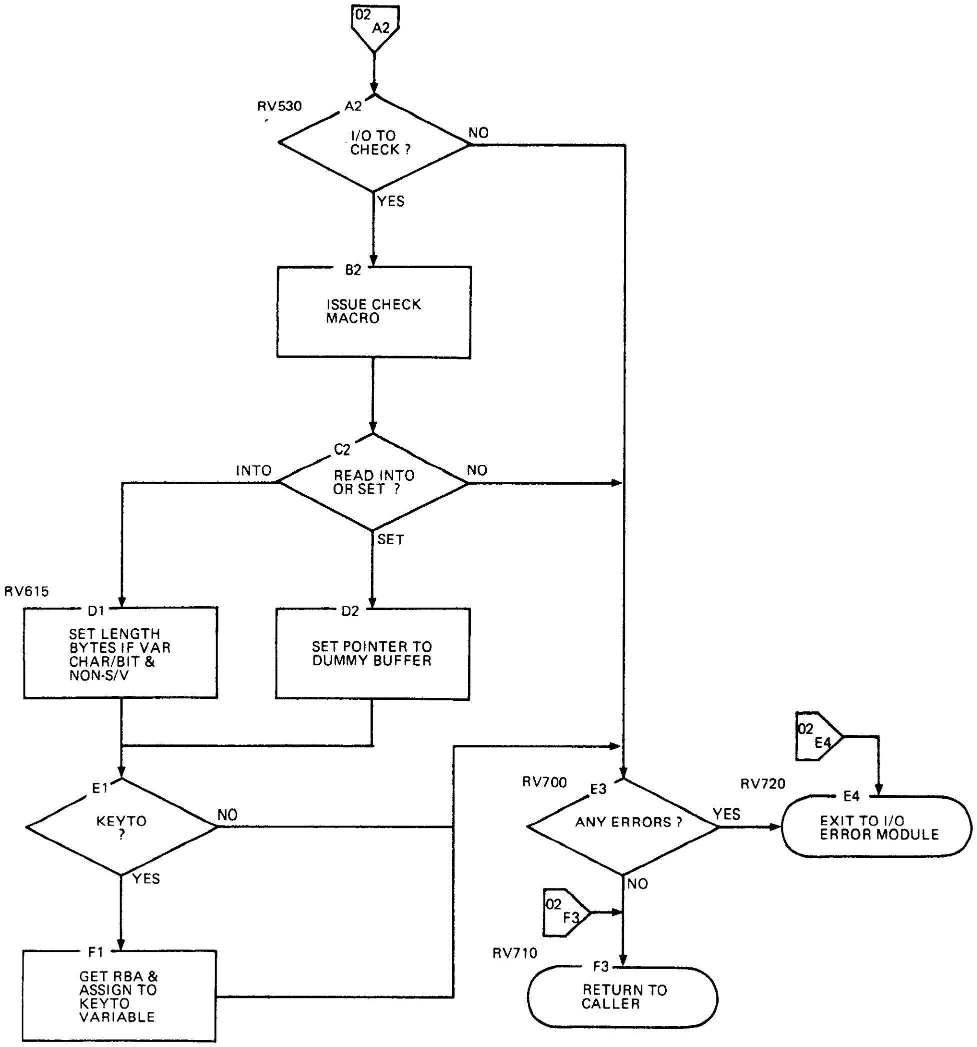
|Chart DRVT. KSDS Sequential output transmitter (part 1 of 2)



|Chart DRVT. KSDS Sequential output transmitter (part 2 of 2)



|Chart DRVZ. ESDS transmitter (part 1 of 2)



|Chart DRVZ. ESDS transmitter (part 2 of 2)

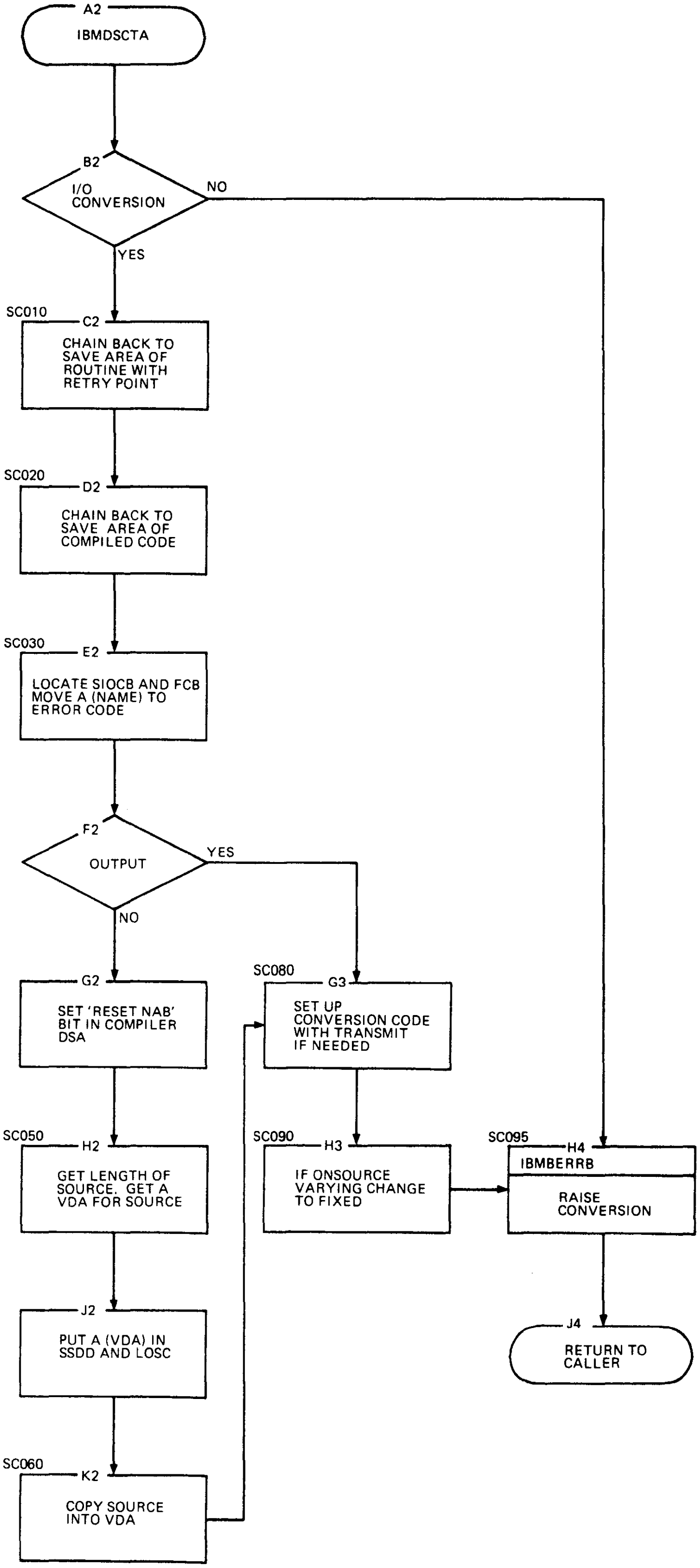

A2
IBMDSCTA
B2
I/O CONVERSION
NO
YES
SC010
C2
CHAIN BACK TO SAVE AREA OF ROUTINE WITH RETRY POINT
SC020
D2
CHAIN BACK TO SAVE AREA OF COMPILED CODE
SC030
E2
LOCATE SIOCB AND FCB MOVE A (NAME) TO ERROR CODE
F2
OUTPUT
YES
NO
G2
SET 'RESET NAB' BIT IN COMPILER DSA
SC080
G3
SET UP CONVERSION CODE WITH TRANSMIT IF NEEDED
SC050
H2
GET LENGTH OF SOURCE. GET A VDA FOR SOURCE
SC090
H3
IF ONSOURCE VARYING CHANGE TO FIXED
SC095
H4
IBMBERRB
RAISE CONVERSION
J2
PUT A (VDA) IN SSDD AND LOSC
J4
RETURN TO CALLER
SC060
K2
COPY SOURCE INTO VDA

Chart DSCT. Conversion condition interface

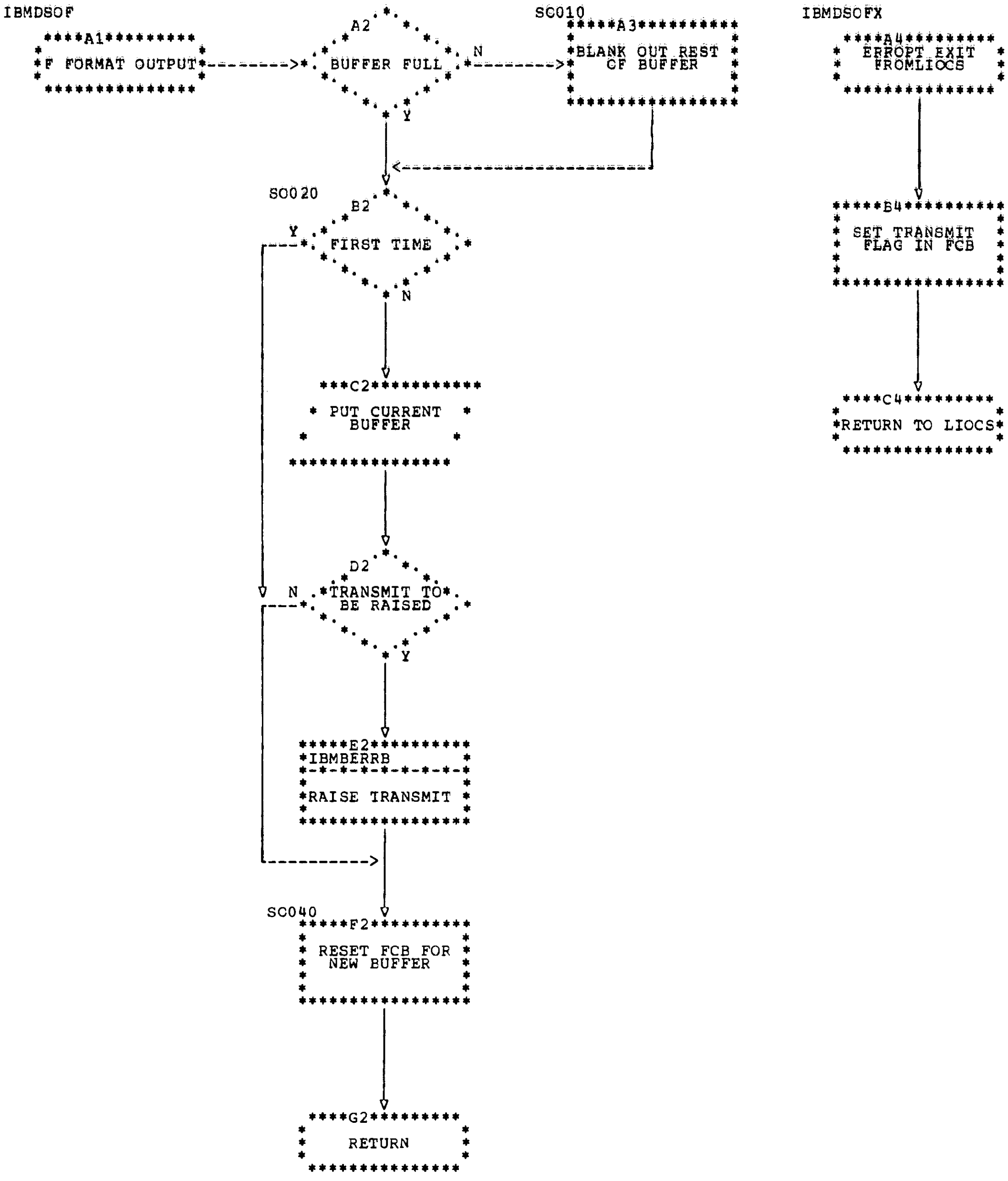


Chart DSOF. Stream output F-format transmitter

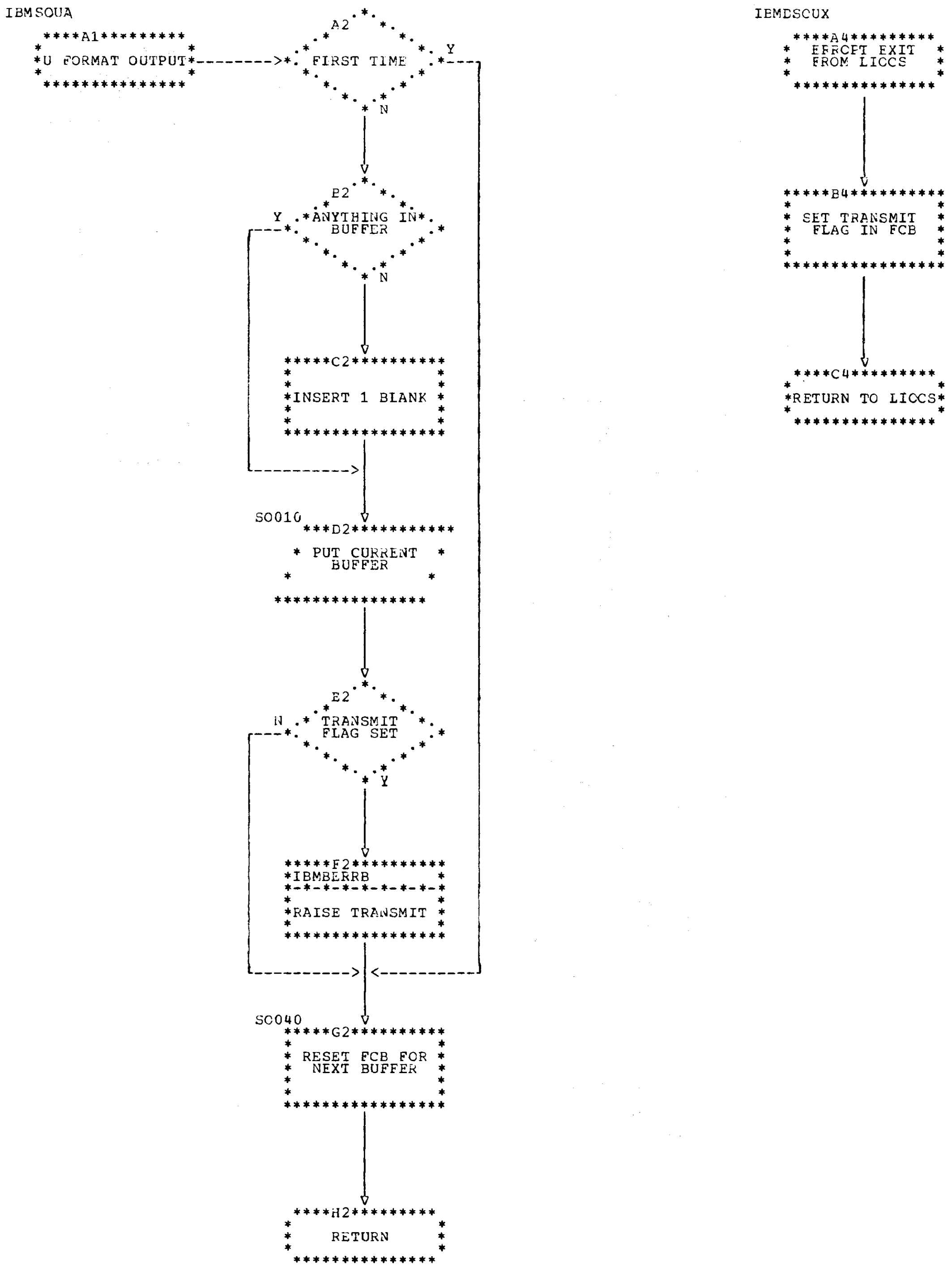


Chart DSOU. Stream output U-format transmitter

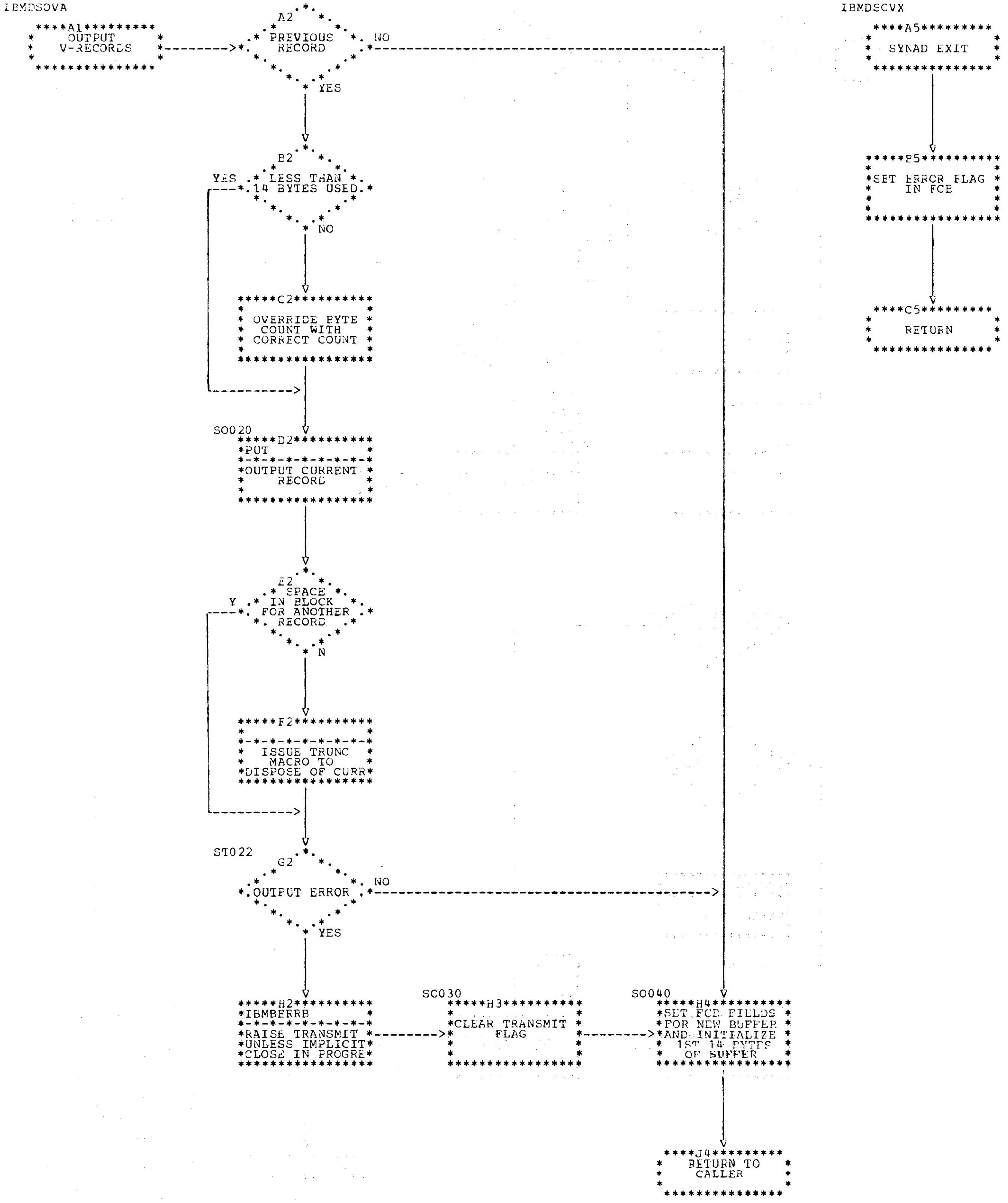


Chart DSOV. Stream output V-format transmitter

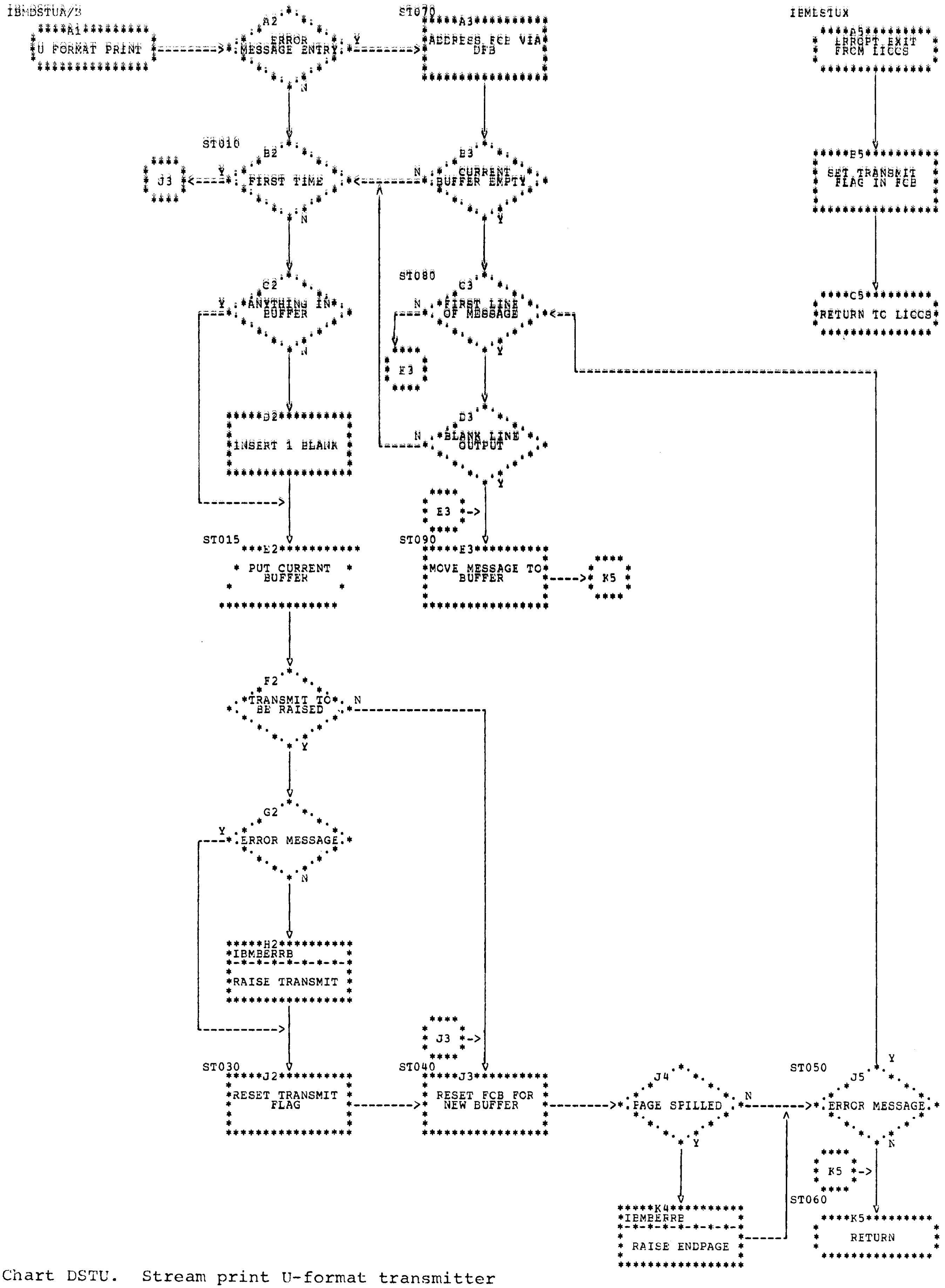


Chart DSTU. Stream print U-format transmitter

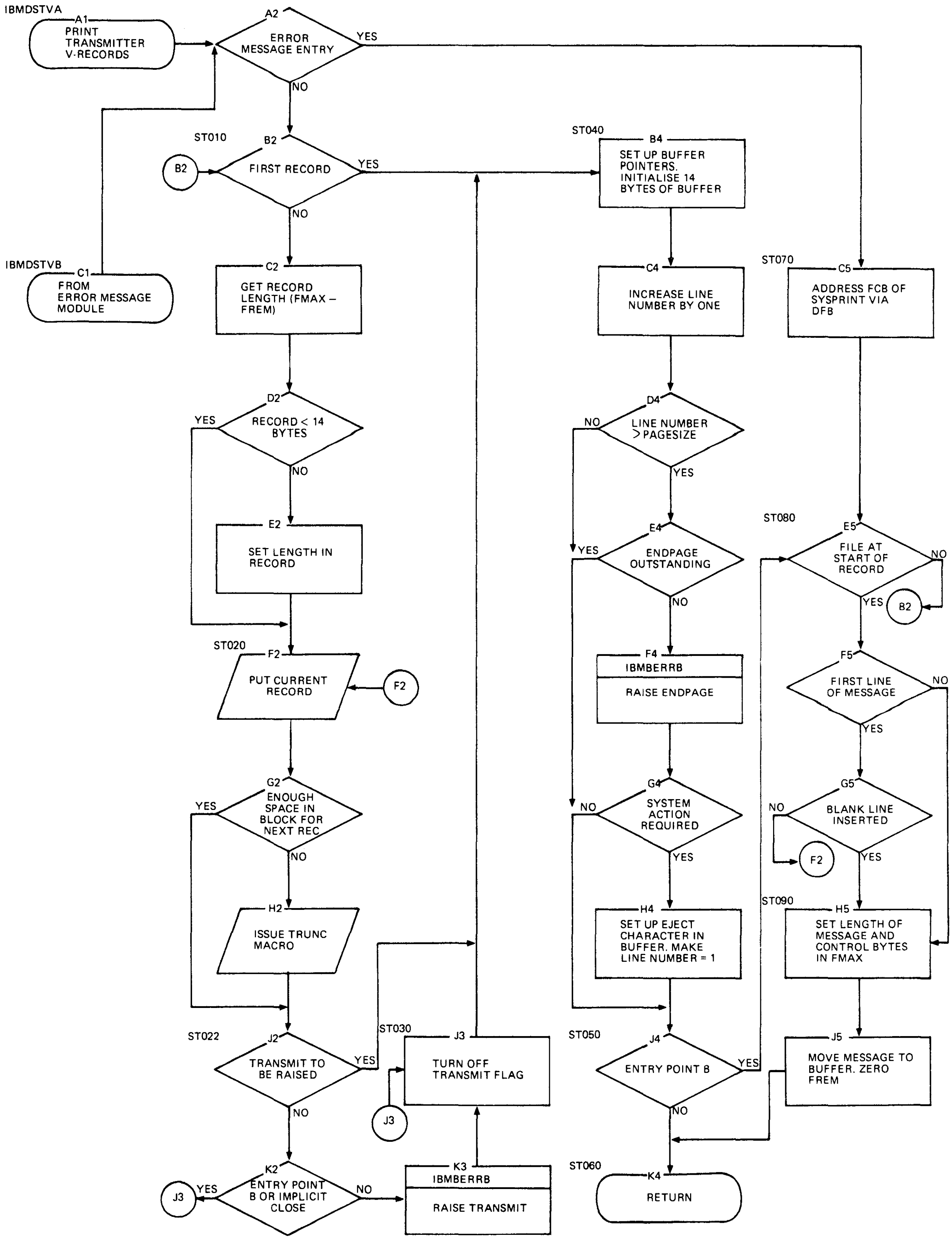
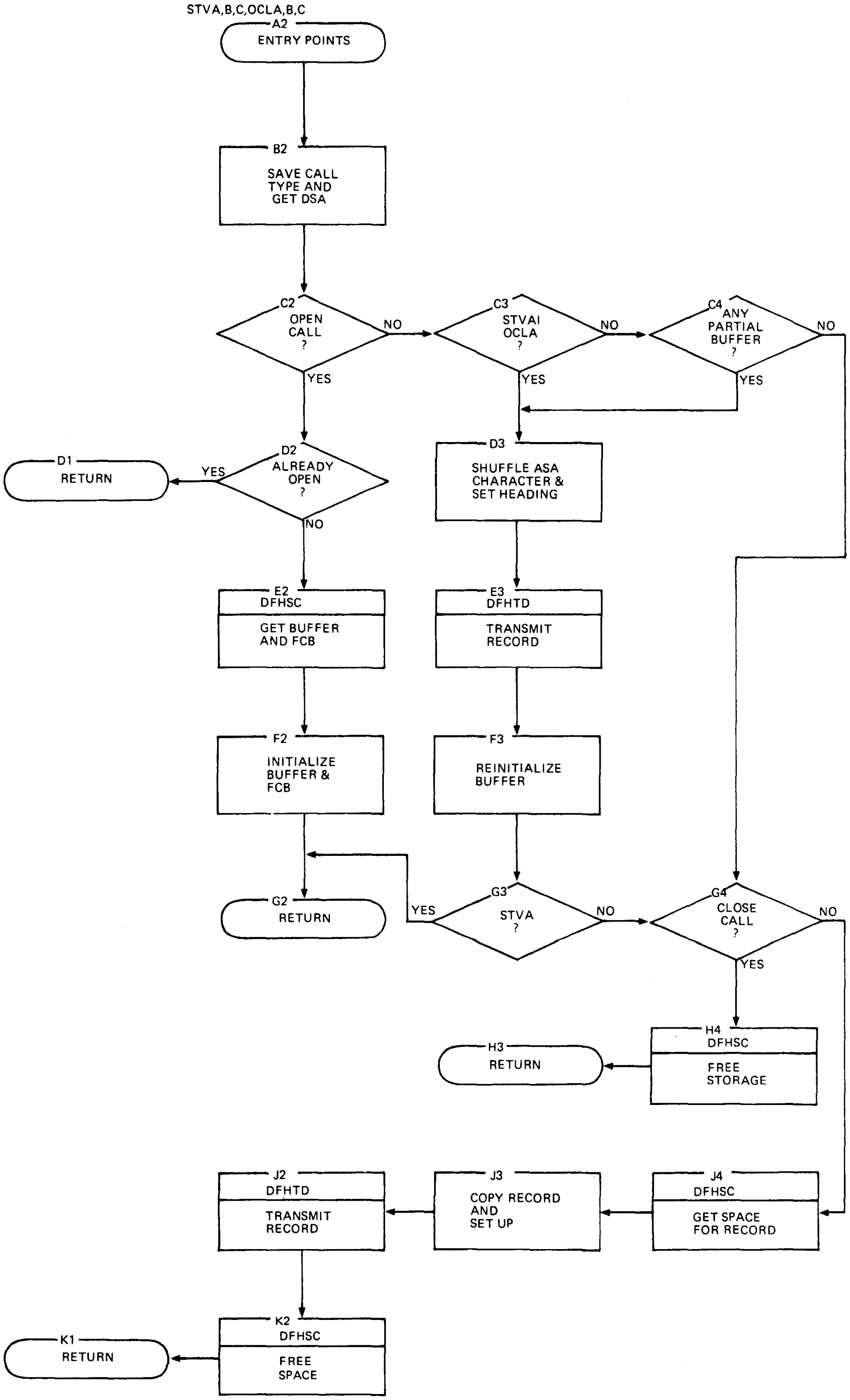


Chart DSTV. Stream print V-format transmitter



|Chart FSTV. Stream output module - CICS

Index

IBM Technical Newsletter

This Newsletter No.	LN33-6180
Date	October, 1976
Base Publication No.	LY33-6012-1
File No.	S360/S370-29
Previous Newsletters	None

DOS
PL/I Transient Library:
Program Logic

This Technical Newsletter, a part of Version 1, Release 5, Modification 0 of the IBM DOS PL/I Transient Library, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent versions, releases, and modifications of the compiler unless specifically altered. Pages to be inserted and/or removed are:

Front cover/Edition notice
5 - 10
10.1 (added)
15, 16
16.1 (added)
21, 22
22.1 - 22.8 (added)
23, 24
24.1 (added)
31 - 34
34.1, 34.2 (added)
35, 36

36.1 (added)
39, 40
40.1 - 40.6 (added)
41, 42
49, 50
50.1 (added)
53 - 60
95 - 104
109 - 110
113, 114
119 - 122
130.1 - 130.8 (added)

134.1 - 134.4 (added)
146.1 - 146.3 (added)
147, 148
150.1 - 150.4 (added)
152.1, 152.2 (added)
161, 162
166.1 - 166.5 (added)
168.1 (added)
187, 188
201 - 210
215 - 219
Reader's comment form
Back cover

A change to the text is indicated by a vertical line to the left of the change.

Summary of Amendments

Changes for Release 5 of the transient library and general updating of the manual.

Note: *Please file this cover letter at the back of the manual to provide a record of changes.*

IBM United Kingdom Laboratories Ltd, Publications Dept, Hursley Park, Winchester, Hants, England.

DOS PL/I Transient Library:
Program Logic

Order No. LY33-6012-1

Reader's Comment Form

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name and address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

Fold and Staple

First Class Permit
Number 6090
San Jose, California

Business Reply Mail

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

Postage will be paid by:

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and Staple

IBM®

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

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

DOS PL/I Transient Library: Program Logic File No. S360/S370-29 Printed in U.S.A. LY33-6012-1

LY33-6012-1

DOS PL/I Transient Library: Program Logic (File No. S360/S270-29) Printed in U.S.A. LY33-6012-1

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)