Licensed Material — Property of IBM
LY33-6011-1
File No. S360/S370-29

Program Product

DOS PL/I Resident Library: Program Logic

Program Number 5736-LM4
(This product is also distributed as part of composite package 5736-PL3)

Feature Number 8051

IBM

Second Edition (March 1973)

This edition applies to Version 1, Release 5, Modification 0 of the DOS 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.

The purpose of this publication is to summarize the internal logic of the modules contained in the DOS PL/I Resident Library. 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 are 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: Execution Logic,
Order No. SC33-0019

ASSOCIATED PROGRAM PRODUCT PUBLICATIONS

Details of the DOS PL/I Transient Library are given in the following publication:

DOS
PL/I Transient Library: Program Logic,
Order No. LY33-6012

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 language 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, Order No. GC26-3783

DOS/VS Data Management Macro Instruction, Order No. GC26-3793

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

AVAILABILITY OF PUBLICATIONS

The availability of the publication is indicated by its use key, the first letter in the order number. The use keys are:

G - General: available to users of IBM systems, products, and services without charge, in quantities to meet their normal requirements; can also be purchased by anyone through IBM branch offices.

S - Sell: can be purchased by anyone through IBM branch offices.

L - Licensed materials, property of IBM: available only to licensees of the related program products under the terms of the license agreement.

PART I: MODULE DESCRIPTIONS

PART II: FLOWCHARTS

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 Resident Library. The transient library is described in DOS PL/I Transient Library: Program Logic.

THE RESIDENT LIBRARY

The DOS PL/I Resident Library consists of about one hundred and eighty program modules which reside on the relocatable library. Each module performs a single function or a number of closely-related functions and is designed as a single control section.

The modules of the resident library form a set of standard subroutines that are used for the majority of interfaces with the system and for handling those tasks that can be most efficiently executed by means of interpretive subroutines.

The main areas where the library is used are: input/output, error handling, storage management, conversions, mathematical functions, and various string and aggregate handling operations.

MODULE NAMING CONVENTIONS

Control Names

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

IBMabc(d)

The 4th letter of the control name (a) is one of the following characters:

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.

G - The module is a replacement module for a "D" module. For example, the single WAIT module IBMGJWT is a replacement module for the multiple WAIT module IBMDJWT in systems that do not support the WAITM macro.

E - The module is a replacement module for a "B" module.

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.

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 routines
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 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 by another library module or by compiled code.

With the exception of the stream I/O transmitters IBMDSTF and IBMDSTI, entry point names are derived from the module's control name by changing the fourth letter to B and 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. In the case of the above two exceptions, the fourth letter is not changed to B.

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 CSECT 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 CSECT names of IBMBEOC and IBMBCO are IBMBEOC1 and IBMBCO01, respectively.

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: Program Management Routines
Chapter 3: Open and Close Routines
Chapter 4: Record I/O Routines
Chapter 5: Stream I/O Routines
Chapter 6: Conversion Routines
Chapter 7: Computational Routines
Chapter 8: Interlanguage Routines
Chapter 9: Miscellaneous Routines

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

Part II contains flowcharts of all the resident library modules that contain a significant amount of executable code. The flowcharts are identified by the 4th, 5th, 6th, and 7th letters of the module name; for example, the flowchart for module IBMDERR appears on chart DERR. 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. The overview gives 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 resident 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 type 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 is considered helpful.

Linkage: The information provided under this subheading is confined to a list of those registers containing parameters, 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. If the module can be called directly from compiled code, this is also indicated.

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: for example, 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 terminated 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 the module flowchart.

PROGRAM FLAGS

In some program listings, the following 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.
/ Return to internal error routine.
= Switch set.
? Switch test or other decision.

LIBRARY MACRO INSTRUCTIONS

A list of library macro instructions, together with brief descriptions of their functions, is given in appendix B. 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 the Disk Operating System, when the letter "D" is used. The 5th letter of the name is always "X" and identifies the name as that of a library macro instruction. The final two or three letters are mnemonic of the function.

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 instructions 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.

CHAPTER 2: PROGRAM MANAGEMENT ROUTINES

The program management section of the library is concerned with two main subjects: housekeeping and error handling.

The housekeeping section of the resident library contains routines concerned with program initialization and termination, storage management, and various other housekeeping operations. These modules are described below under the heading "Housekeeping Routines." A full description of the storage situation at execution time is contained in DOS PL/I Optimizing Compiler: Execution Logic.

The error-handling modules of the resident library all have "E" as the fifth letter of their control names. They are described below under the heading "Error Handling Routines."

HOUSEKEEPING ROUTINES

MODULE DESCRIPTIONS

DFHPL1I - CICS Initialization Bootstrap

Function

Acts as the entry point for PL/I programs under CICS, and replaces the compiler generated PLISTART. DFHPL1I also supplies bootstraps for the PL/I program to the transient library routines in the CICS nucleus and to CICS services.

Method (chart DFHPL1I)

When called from CICS for initialization, or by the program for library function, DFHPL1I sets up the parameter registers and hands control to the transient library module IBMFPCCA in DFHSAP (found via the CSA). If called for a CICS service, control is transferred to the interface in DFHPCP, which is found via an address stored in this module by the CICS loader.

Linkage

For a call to IBMHPIR

R0 = input value of R13 (DSA into which callers registers are saved).
R1 = input value
R10 = initialization parameter list address
R12 = A(CICS TCA)
R13 = A(CICS CSA)

Calls

IBMFPCCA - for various PL/I functions in DFHSAP
DFHPCP - for CICS service requests

Called By

CICS
Compiled code for services.

IBMBPAF - Controlled Variable Management

Function

To allocate and free controlled variables. The module has two entry points:

IBMBPAFA: Allocate controlled variable.

IBMBPAFB: Free controlled variable.

Method (chart BPAF)

Entry Point IBMBPAFA:

The length of the storage required and the address of the control section containing the address of the most recent allocation of the controlled variable are passed from compiled code. The length is increased by 16 bytes to hold control information, and rounded up to a multiple of eight bytes. Module IBMDPGR is then called to allocate the storage.

On return from IBMDPGR, the control information is initialized in the extra 16 bytes. The allocated storage is then added to the chain of allocations for the variable concerned, and control is returned to compiled code.

Note: Initially, the address in the CSECT is set to zero. The chain back field in the first allocation of a controlled variable thus contains zero.

Entry Point IBMBPAFB:

The most recent allocation of the variable is dechained and module IBMDPGR is called to free the storage. The length of the storage to be freed is obtained from the length field in the control information for that allocation.

Linkage

```
R1    = A(PLIST)
PLIST = A(CSECT)
        A(length of data section) (entry point IBMBPAFA only)
```

Calls

IBMDPGR - Storage management.

Called By

Compiled code.

IBMBPAM - Area Management

Function

To allocate and free storage within an area variable and to support assignment of areas. The module has three entry points:

IBMBPAMA: Allocate a based variable in a specified area variable.

IBMBPAMB: Free a specified based variable in a specified area variable.

IBMBPAMC: Assign the contents of a source area variable to a target area variable.

Method (chart BPAM)

Entry Point IBMBPAMA:

Whenever possible, allocations are made in elements on the free element chain. Each allocation is made in the smallest free element that is large enough, and the free element chain is then rearranged so that the free elements are once more chained in descending order of size.

If there is no free element chain (free-element-chain flag off), or if the largest element in the chain is too small for the allocation, an attempt is made to allocate the storage in the space between the end of extent (EE) pointer and the end of the area. If this space is too small the AREA condition is raised, and the error-handling routine is called at entry point IBMBERRB.

On return from the error handler, control is returned to compiled code at the address given in the parameter list for abnormal return.

Entry Point IBMBPAMB:

Pointers are set to the low-address and high-address ends of the allocation that is to be freed. If there is a free element chain (free-element-chain flag on), it is scanned to see whether the section to be freed is contiguous, at either end, with any of the free elements. Any such free elements are dechained and added to the section to be freed.

When the scan is complete, a test is made to see whether the section to be freed is contiguous with the end of the extent. If it is, EE is updated, and a test is made to see whether there are any elements remaining on the free element chain: if there are not, the free-element-chain flag is turned off. If the section to be freed is not contiguous with the end of the extent, it is placed in the correct position in the free element chain. If this is the last element in the free element chain, the free chain flag in the area variable is turned off.

If there is no free element chain and the section is not contiguous with the end of the extent, the section is made the first element on a free element chain.

Entry Point IBMBPAMC:

The length of the target area is compared with the length of the allocated storage in the source area. If there is sufficient room in the target area for the assignment, all the allocated storage in the source area, including the heading at the top, is moved onto the target area.

If there is insufficient room in the target area, the AREA condition is raised and a call is made to entry point IBMBERRB of the error handler. On return from the error handler, control is returned to compiled code.

Error and Exceptional Conditions

The AREA condition is raised if the storage allocation or the assignment cannot be made.

Linkage

Entry Point IBMBPAMA:

```
R1    = A(PLIST)
PLIST = A(AREA locator)
        A(word to hold address of storage allocated)
        A(length to be allocated)
        A(label constant for abnormal return)
```

Entry Point IBMBPAMB:

```
R1    = A(PLIST)
PLIST = A(AREA locator)
        A(pointer to allocation to be freed)
        A(length to be freed)
```

Entry Point IBMBPAMC:

```
R1    = A(PLIST)
PLIST = A(source area locator)
        A(target area locator)
```

Calls

IBMDERR - Error handler.

Called by

Compiled code.

IBMBPGO - Reset CHECK Enablement

Function

To set current CHECK enablement to that at block entry, during abnormal GOTO out of a block.

Method

The module first tests for any dynamic ONCBs, and if there are none, branches immediately to the target code addressed by the link register.

If there are dynamic ONCBs present, the module places the address of the latest ONCB, contained in the DSA, into register R1, inspects the ONCB, and if necessary, sets the value for CHECK enablement to as it was at block entry.

The process described above is repeated in turn for every other ONCB there may be and when every ONCB concerned has been dealt with, the module branches to the target code addressed by the link register.

Linkage

```
R2   =   A(target base)
LR   =   A(target compiled code for return)
R4   =    (target value)
AR   =    (target value)
```

Called By

Compiled code

IBMDPGR - Storage Management

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 DPGR)

Entry Point IBMBPGRA:

Before allocating storage, the module calls its own entry point IBMBPGRC (see below) to free any empty segments.

Whenever possible, allocations of non-LIFO storage are made from areas on the free area chain. IBMDPGR 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 the 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 word, is then reduced by the number of bytes allocated.

If no free area chain exists, or if all the areas on 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 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 between the NAB and EOS pointers is too small the allocation cannot be made. Under these circumstances IBMBPGRA sets the appropriate return code in the TCA and returns to the program initialization routine IBMDPIR, which terminates the program.

Entry Point IBMBPGRB:

The length of storage that is to be freed is first rounded up to a multiple of 8 bytes.

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, IBMBRGRC 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 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 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 R14 is saved in the caller's save area as well as in the program save area. This is done to enable the point of invocation of IBMDPGR to be found if an "insufficient storage available" message has to be printed.

Error and Exceptional Conditions

Control is returned to IBMDPIR if there is no storage available.

Linkage

Entry Point IBMBPGRA:

On entry:

R0 = length of storage required

On return:

R1 = address of storage obtained

Entry Point IBMBPGRB:

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

Entry Points IBMBPGRC and D:

On entry:

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

On return:

R0 = new NAB
R1 = address of storage for element

Calls

IBMDPIR - Program initialization.

Called By

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

IBMDPIR - Program Initialization from System

Function

To load IBMDPII at the end of the partition, for that module to establish an initial storage area (ISA), and to pass to IBMDPII a parameter list of address constants. IBMDPIR also contains a subroutine which is called from the goto code in the TCA whenever abnormal GOTO out of block processing is required. Furthermore, the module contains a default PLIMAIN in the form of a separate CSECT. This CSECT contains its own entry point address in its first four bytes.

Method (chart DPIR)

The module first addresses the communications region and then loads IBMDPII at the end of the partition. Control is then passed to IBMDPII.

The subroutine which is called from the TCA goto code to perform abnormal GOTC out of block functions, may be used during execution of the PL/I program to call exit DSA processing (e.g. sort exits), return being made to the TCA goto code. IBMDPGR may enter the subroutine to cause termination of the program when there is insufficient storage to continue execution. In such a case, a branch is made to the start of IBMDPIR's CSECT where IBMDPIR overlays itself by loading IBMDPES to produce an "INSUFFICIENT STORAGE" message.

The subroutine is also entered by IBMDERR, via the TCA goto code, to terminate the program after normal return from, or standard system action for the FINISH condition. When so entered, it calls IBMDOCLD to close any open PL/I files, and performs any exit DSA processing required in the following manner. The routine continues processing down the DSA chain until the target DSA for the goto (i.e., the dummy DSA) is reached.

When the dummy DSA is reached during termination, a test is made to determine whether the PL/I program was invoked from another language. If so IBMDPJR is called at entry point IBMDPJRC. Otherwise if the program is terminating due to ERROR being raised, a CANCEL macro is issued to flush any data in the SYSIPT stream. For normal finish, STOP or EXIT, an EOJ macro is issued.

<u>'No Main Procedure' CSECT</u>:

Note: PLIMAIN is generated by the compiler as a result of processing the option MAIN in the OPTIONS option. It contains the address of the PL/I main procedure. If the PL/I program has no main procedure, a dummy PLIMAIN is picked up at link-edit time.

The CSECT when entered, gets library workspace as a save area, and then loads and branches to the transient module IBMDPEP, which issues a 'NO MAIN PROCEDURE' message. Control is then returned to the caller.

Error and Exceptional Conditions

If there is insufficient room in the ISA for the setting-up of the various control blocks, transient module IBMDPES is loaded and invoked to issue a 'NO STORAGE AVAILABLE' message. Control is then returned to the caller.

Linkage

R0 = A(PLIMAIN)

Calls

IBMDOCL - Open/close bootstrap.
IBMBPGO - Reset CHECK enablement.
IBMDPEP - Housekeeping diagnostic message module.
IBMDPES - Storage management diagnostic message module.

IBMDPJR - Program Initialization from Caller

Function:

To establish an initial storage area, (ISA) allowing for an interlanguage communication Fortran buffer and DTF if necessary, and then load IBMDPJI to initialize the ISA. The module has five entry points, two of which are for a PL/I caller who wishes to invoke the same PL/I procedure each time.

PLICALLA: Entry from PL/I caller with the ISA not specified.

PLICALLB: Entry from PL/I caller with the length and address of the ISA specified.

IBMBPJRA: Entry from PL/I caller with a pointer to address of compiled code, but ISA not specified.

IBMBPJRB: Entry from PL/I caller with a pointer to address of compiled code, and ISA specified.

IBMBPJRC: Entry from IBMDPIR for return to caller.

Method (chart DPJR)

The module first determines the length of the ISA and the address of compiled code. If the length is sufficient for the program management area (PMA), flow table and (if required) Fortran buffer and Fortran DTF, the module loads and invokes IBMDPJI to initialize the ISA. If however, the length is insufficient, then IBMDPEP is loaded for that module to produce a "no storage available" message.

To terminate the program, entry point IBMBPJRC is entered from IBMDPIR. IBMDPJRC calculates the return code and resets the caller's program check options and program mask by issuing a STXIT macro. Return is then made to the caller.

Error and Exceptional Conditions

If there is insufficient room in the ISA for the setting-up of the various control blocks, transient module IBMDPES is loaded and invoked to issue a "no storage available" message. Control is then returned to the caller.

Linkage

Entry Point PLICALLA:

R1 = A(compiled code plist)

Entry Point PLICALLB:

```
R1    = A(PLIST)
PLIST= A(complilied code plist)
       A(length of ISA)
       A(passed ISA)
```

Entry Point IBMBPJRA:

```
R1 = A(compiled code plist)
R0 = A(PLIMAIN)
```

Entry Point IBMBPJRB:

```
R1    = A(PLIST)
PLIST= A(compiled code plist)
       A(length of ISA)
       A(passed ISA)

R0    = A(PLIMAIN)
```

Calls

IBMDPJI - Program ISA initialization from caller.

Called By

Non-PL/I Language caller.

IBMDPOL - Overlay Space Saving Module

Function

To provide dummy entry points in overlay phases so that certain modules are not included in DOS overlay phases in which they would never be executed. Thus the space such modules would otherwise occupy is saved. The module has thirteen entry points, which are in fact dummy entry points but which suffice to resolve references to the modules to be excluded. The entry points are as follows:

```
IBMBPGRA, IBMBPGRB, IBMBPGRC, IBMBPGRA
IBMBEFLA, IBMBEFLB, IBMBEFLC
IBMBTOCA, IBMBTOCB
IBMBJWTA
IBMBERCA
IBMBPIRA
IBMBPLRA
```

Note: If such modules are to be used at all they should be in the root phase of the overlay program.

IBMDPOV - Overlay

Function

To implement the PLICVLY built-in subroutine.

Method (chart DPOV)

A LOAD macro is used to load the specified phase into store.

Linkage

R1   = A(PLIST)
PLIST = A(string locator for name of module)

Called By

Compiled code.

IBMBPRC - Return Code Module

Function

To set up the return code specified by the user.

Method

The module moves the specified return code into a halfword field in the TCA.

Linkage

R1   = A(PLIST)
PLIST = A(halfword containing the required return code value)

Called By

Compiled code.

IBMBTOC - COMPLETION Pseudovariable and Event Variable Assignment

Function

Tc implement the COMPLETICN pseudo-variable and to assign event variables. The module has two entry points:

IBMBTOCA: COMPLETION pseudo-variable.

IBMBTOCB: Event variable assignment.

Method (chart BTOC)

Entry Point IBMBTOCA:

The value of the completion bit specified in the parameter list is moved tc the target EV. If the EV is being set complete (completion bit = '1'B) and an EVTAB chain exists, a bit is set in each EVTAB on the chain to indicate completion for the associated WAIT.

Entry Point IBMBTOCB:

The completion and status values of the source EV are moved to the target EV. If the completion bit value is '1'B and an EVTAB chain exists, a bit is set in each EVTAB on the chain to indicate completion for the associated WAIT.

Error and Exceptional Conditions

The ERRCR condition is raised if the target EV is active.

Linkage

Entry point IBMBTCCA:

```
R1    = A(PLIST)
PLIST = A(target EV)
        A(fullword containing the value to be assigned to the
          completion bit)
```

Entry Point IBMBTCCB:

```
R1    = A(PLIST)
PLIST = A(target EV)
        A(source EV)
```

Calls

IBMDERR - Error handler.

Called By

Compiled code.

ERROR HANDLING ROUTINES

The principal module described in this section is the main execution-time-error handling module IBMDERR. This module handles two types of conditions that cause interruption to the main flow of a PL/I program. These are:

1. Program check interrupts.

2. Conditions that are detected by code.

The functions of the error handler are to determine the nature of the error or condition and to take the appropriate action. Several courses of action are possible,including:

1. Enter an on-unit.

2. Print SNAP messages.

3. Print an error message, raise the ERROR condition, and then raise the FINISH condition.

4. Print an error message and return to the caller.

5. No action; return to the caller.

The other modules described in this section include the FLOW option module IBMBEFL, which creates a flow statement table, the CHECK module IBMBERC, which is used as a subroutine by IBMDERR to handle the CHECK condition, and various modules that evaluate condition-handling built-in functions.

The message-generating modules that support IBMDERR are in the DOS PL/I Transient Library and are described in DOS PL/I Transient Library: Program Logic.

MODULE DESCRIPTIONS

IBMDEFL - FLOW option

Function

To create the flow statement table and to fill in the values of the statement numbers and the names of the procedures in which they occur. The module has three entry points:

IBMBEFLA: Create and initialize the table.

IBMBEFLB: Place normal entries in the table.

IBMBEFLC: Place entries in the table in the middle of GOTO code.

Method (chart DEFL)

Entry Point IBMBEFLA:

On entry, register R1 addresses two halfwords. The first halfword contains the number (N) of branch-in/branch-out points that are required in the table. The second contains the number (M) of procedure names or on-unit entries that are required in the table.

If M=0 and N=0, no flow table is created, and control is returned to the caller.

If M=0 and N=1, N is set equal to 2.

If M≠0 and N=0, N is set equal to 2, and a flag is set on in the flow table to indicate to the transient modules IBMDESN and IBMDKTB that statement numbers are not to be printed in trace messages.

If M≠0 and N=1, N is set equal to 2.

N is set equal to 2 in the above cases because at least 2 pairs of statement numbers are needed in the flow table if procedure name or on-unit type entries are to be made. However, if the caller has specified N=0, no statement numbers are required in trace messages.

The module then creates the flow table as follows:

The amount of storage required for the table is calculated as 41+12*N+8*M (rounded up to a multiple of 8 bytes), and this amount of storage is then obtained. The first 41 bytes are for the table header.

The header is initialized for the entries to be made. Different GOTO code, depending on whether FLOW has been specified, is moved into the TCA by the transient modules IBMDPII or IBMDPJI. If there is FLOW then this code picks the address of IBMBEFLC from a position in the space allowed for GOTO code in the TCA and branches to it. This address will have been placed there, during initialization of the table, by IBMDEFL.

The addresses of the flow table and of entry point IBMBEFLB are then stored in the TCA and control is returned to the caller.

Entry Point IBMBEFLB:

The action taken by the module depends upon the values of the flag bits passed to it. The flag bits are used directly to effect a branch to the relevant section of the module.

Flag Bits = '00'B:

This flag bit setting indicates that a normal branch-out entry is to be made in the table. If the previous entry in the table was also a branch-out, the current branch-out is ignored.

If the previous entry was a branch-in, a test is made to determine whether or not it is flagged as an entry from a GOTO. (This flag will have been set by entry point IBMBEFLC if a GOTO out of block has occurred, see below). If it is, the name of the block that was entered by the GOTO must be placed in the name table. This is done by means of an internal subroutine.

The module then moves the statement number of the branch-out point to the number table and returns to the caller.

Flag Bits = '01'B:

This flag bit setting indicates that the entry required in the table is a branch-out entry relating to the end of an on-unit. The module moves the statement number to the table, and follows it by a dummy branch-in entry. The dummy maintains the branch-out/branch-in pairing of the table entries and relates to the statement that caused entry to the on-unit. The name of the block to which return is to be made is placed in the name table.

Flag Bits = '10'B:

This flag bit setting indicates that a normal branch-in entry is to be made in the table.

If the previous entry in the table was also a branch-in, the current branch-in is ignored.

If the previous entry was a branch-out, a test is made to determine whether or not it is flagged as an entry before a GOTO. (This flag will have been set by entry point IBMBEFLC if a GOTO out of block has occurred, see below.) If it is, the name of the block that was entered by the GOTO must be placed in the name table. This is done by means of an internal subroutine.

The module then moves the statement number of the branch-in point to the number table and returns to the caller.

Flag Bits = '11'B:

This flag bit setting indicates that a branch-in entry is required in the table and that the branch-in has an associated block name.

If the previous entry in the table was a branch-out, the module moves the current branch-in point to the number table and then uses the internal subroutine to find the block name and place that in the name table.

If the previous entry was a branch-in, a test is made to determine whether the current branch-in is to a procedure or to an on-unit.

For procedures, the current branch-in is ignored.

For on-units, a branch-out entry corresponding to the point of interrupt is entered in the table. The previous entry to this is then tested to determine whether or not it is flagged as an entry before a GOTO. If it is, the internal subroutine is used to find the name of the block in which the interrupt occurred and place it in the name table.

The current branch-in is then handled exactly as though the previous entry were a branch-out.

Entry Point IBMBEFLC:

On entry, a test is made to see whether or not the branch is to a block other than the originator of the branch. This may only be out of a begin block into its containing procedure or may be from procedure to procedure etc. In all cases, if the target DSA is not the source DSA then a new procedure name is assumed to be needed in the table. If so, a flag is set in the last entry (the branch-out value) to indicate that for the next branch-in entry, a new procedure name or on-unit type must be given.

The test is made by comparing the current DSA with the target DSA.

Linkage

Entry Point IBMBEFLA:

R1 = A(FLOW option numbers) (two halfwords)

Entry Point IBMBEFLB:

LR = A(halfword)
Halfword = flag bits (2 bits)
number value (14 bits)

Entry Point IBMBEFLC:

BR = A(target DSA)

Called By

Compiled code.

IBMBEOC - On-Code Module

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 smaller 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 ON-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 IBMBEOCA 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 operation on the first byte of the error code.

IBMBEOCA finds the error code by picking up the chain back field in the current ONCA and obtaining the error 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 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 error code from 255.

The final value of the ON-code is obtained by zeroing bits 0, 1, and 2 of the second byte (or bits 0 and 1 only for CONVERSION) 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

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

IBMBEOL - ONLOC Built-In Function

Function

To find the value of the ONLOC built-in function. The module has a single entry point: IBMBEOLA.

Method (chart BEOL)

Starting at the current DSA, the module searches back through the DSA chain until an on-unit DSA is found. If the dummy DSA is reached before an on-unit DSA is found, i.e., if the ONLOC statement is out of context, a null string is returned to the caller.

If an on-unit DSA is found, the search is continued back along the DSA chain until a procedure DSA is found. This DSA belongs to the procedure in which the condition that caused entry to the on-unit was raised.

The length of the procedure name (in bytes) immediately precedes the procedure's entry point, which is always aligned on a fullword boundary. The name is located immediately before the length field. The entry point is located by picking up the contents of register 15 from the previous DSA's save area.

Linkage

R1 = A(string locator for the returned name)

Called By

Compiled code.

IBMBERC - CHECK (System Action)

Function

To perform standard system action for the CHECK condition.

Method (chart BERC)

The CHECK module is entered from the error handler, IBMDERR, whenever the CHECK condition has been raised and standard system action is to be taken.

The module first acquires a new copy of library workspace (LWS) for use by the stream I/O modules that will be called. It then accesses the parameter list that was passed to IBMDERR (by extracting the contents of the register R1 save slot in the DSA prior to that of IBMDERR), and obtains from this parameter list the address of the symbol table (or list of symbol tables) for the CHECK.

The module now builds a parameter list to pass to the data-directed output module IBMDSDOA. If the variable for which CHECK has been raised is an array and only a single element is to be printed, the parameter list for IBMDSDOA is set up in the current DSA; otherwise, a VDA is obtained for the parameter list.

The address of an area for a Stream I/O Control Block (SIOCB) is placed in the first word of the parameter list.

If the CHECK module has been entered as a result of a SIGNAL CHECK statement with no identifier, the enablement of the CHECK for each PL/I variable is determined and the SYMTAB addresses of the enabled variables are placed in the parameter list.

If the module has been entered as a result of an ordinary CHECK, the single SYMTAB address is placed in the parameter list.

The SIOCB is then set up in the current DSA and the stream I/O initialization module IBMDSIO is called. On return from IBMDSIO, the data-directed output module IBMDSDO is called. If there is an array address in the parameter list, entry point IBMBSDOB is called, otherwise entry point IBMBSDOA is called.

Linkage

DR = A(IBMDERRs library DSA)

Calls

IBMDSIO - Initialize stream I/O.
IBMDSDO - Data-directed output.

Called By

IBMDERR - Error handler.

IBMDERR - 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.

IBMBERRB: Conditions and errors detected by code.

IBMBERRC: Program check interrupts while control is in IBMDERR.

Method (chart DERR)

Program Check Interrupts:

Program check interrupts are handled by entry point IBMBERRA. PL/I interrupt handling is established by means of a STXIT macro (issued in module IBMDPII) which gives the supervisor the addresses of a small piece of code and a 72-byte save area in the implementation appendage (TIA). When an interrupt occurs, the supervisor saves the old PSW and the contents of registers R0 through BR in the save area and branches to the code in the TIA. This code loads the address of the save area into register R0 and branches to the address held in a word called TERA in the TIA. Provided that control was not already in the error handler when the interrupt occurred, TERA contains the address of IBMBERRA.

IBMBERRA saves the second word of the old PSW (from the save area) and the contents of registers R0 through RY in the current DSA, and then changes the address part of the old PSW in the save area to an address (ER010) in IBMDERR. An EXIT macro is then issued, and the supervisor returns control to ER010.

On entry to ER010, a flag is set to indicate that the module was entered at entry point IBMBERRA.

Processing the Interrupt

IBMDERR creates a two-byte PL/1 internal error code from the interrupt code in its workspace.

If the interrupt was an operation exception, a routine addressed by the TAFF slot in the TIA is called. In a machine with floating point hardware, this routine is merely an immediate return to IBMDERR. For machines without floating point hardware, the routine addressed picks up the PSW, finds the operation code of the instruction that caused the exception, and compares that code with the floating point instruction operation codes. If it was a floating point instruction that was attempted, then the routine sets on a bit in the error code in the DSA belonging to IBMDERR, changing it from FF01 to FF21.

If the interrupt corresponds to a PL/1 on-condition, the floating point registers are saved in IBMBERRAs DSA in case return is to be made to the point of interrupt. If the interrupt was floating point underflow then 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 compiled code will have set a bit in the interrupt qualifier byte in the TCA. Hence if this bit is set, IBMBERRA creates a code for SIZE and sets the "ignore" bit in the qualifier byte in the TCA. If the "ignore" bit is set already when one of the above exceptions occurs, IBMBERRA returns to the point of interrupt. Having created the internal error code, IBMBERRA branches to the main condition handling logic described below.

Conditions and Errors Detected by Code

When a condition is to be raised by the library or compiled code, IBMBERRB is passed an interrupt control block containing a two or four byte code indicating which condition to raise. For PL/1 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 (e.g. attribute conflict for UNDEFINEDFILE). The third and fourth bytes contain flags indicating which ON functions and pseudo-variables are valid for the condition. If the condition is a qualified condition, the interrupt control block (ICB) also contains a qualifier. The ICB can be up to 6 words long for the CHECK condition. For such a condition, the third word is always required, and contains the DSA level number and flags. The fourth word is required for CHECK of an array element. Word five is the address of an implementation defined block to define the variable. The sixth word may be used to address the generation of the variable being checked.

For conditions for which there is no PL/1 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, in twos. If entry has been made via entry point IBMBERRA a code of this type will have been created.

IBMDERR 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 first of all to print a diagnostic message. The message is produced by the transient module IBMDESM, which is dynamically loaded and invoked. This module is described in DOS PL/I Transient Library: Program Logic.

On return from the transient message module, a code for the ERROR condition is created and the action now taken is that for on-conditions.

For each on-condition, IBMDERR 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, IBMDERR 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, IBMDERR 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.

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.

Having determined the enablement of the condition, the establishment must now be found. 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 IBMDERR. If a matching on-cell has not been not located when the dummy DSA is reached, standard system action is taken for the condition, the action on SSA 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, SSA 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 has been found or a matching ONCB which does not specify unestablished, then 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 IBMDESM. This module is described in DOS PL/I Transient Library: Program Logic.

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

3. If there is a GO TO only in the on-unit then the 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, IBMDERR invokes it.

Before branching to the on-unit, IBMDERR must perform some housekeeping duties. The contents of the interrupt qualifier byte in the TCA are saved in IBMDERRA's DSA and the qualifier byte is set to zero. IBMDERRA 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 pseudo-variables which may be used in the on-unit. If the condition is I/O, a string locator is placed in the ONCA for ONFILE. 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, IBMDERR 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 IBMDERR to enter an on-unit or perform a GOTO, the interrupt linkage to IBMBERRA must be restored. This involves replacing the address of IBMBERRC in the TCA by the address of IBMBERRA. If there is no GO TO out of the on-unit the address of IBMBERRA is replaced by IBMBERRC on return to IBMDERR.

On return from an on-unit the interrupt qualifier is restored and the new LWS freed. The action byte for the conditionis tested to see if a return is to be made or if special action is to be taken. If the action specified is return, IBMDERR 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 program 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 program initialisation routine IBMDPIR. If FINISH was raised by the SIGNAL statement, IBMDERR 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.

5. SUBSCRIPTRANGE - The ERROR condition is raised.

<u>System Action</u>

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 IBMDESM is loaded and invoked. This module is described in DOS PL/I Transient Library: Program Logic. On return from the transient message module, the action byte is tested to determine the next action. 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 IBMDERR depends on the entry point at which it was entered. If the entry point was IBMBERRB, return is made to the caller in the normal way by a branch on register 14. However, if entry was made via IBMBERRA, a return to the point of the program check interrupt must be made. This is done in the following way:

1. The module restores the floating point registers and changes the address in TERA to an address within itself. It then causes an interrupt.

2. The supervisor gains control after the interrupt and branches to code in the TCA. That code causes a branch to the address in TERA, passing the address of the save area in register 1.

3. The module then changes the address in TERA to that of IBMBERRA and resets the contents of the save area to what it was when IBMBERRA was first entered.

4. An EXIT macro is issued, and the supervisor returns control to the point of the original interrupt.

Entry Point IBMBERRC

Transient module IBMDPEP is loaded and invoked to print the message, and a DUMP macro is issued to dump main storage and terminate the job. Module IBMDPEP is described in DOS PL/I Transient Library: Progam Logic.

Linkage

Entry Points IBMBERRA and C:

R1 = A(save area)

Entry Point IBMBERRB:

R1 = A(interrupt control block)

Calls

IBMBERC - CHECK (system action).
IBMDESM - Error message module (transient).
IBMDPEP - Housekeeping diagnostic message module.
IBMDPIF - Operation Exception Checking, (No floating-point support).

CHAPTER 3: OPEN AND CLOSE ROUTINES

The resident library contains one module concerned with opening and closing PL/I files: IBMDOCL. This module is a bootstrap routine that effects linkage between compiled code or module IBMDRIO (chapter 4) and the DOS PL/I Transient Library. The transient library is described in DOS PL/I Transient Library: Program Logic.

MODULE DESCRIPTIONS

IBMDOCL - OPEN/CLOSE Bootstrap

Function

IBMDOCL is used by compiled code and by the library to open and close files. It has five entry points:

IBMBOCLA: Explicit open.

IBMBOCLB: Implicit open.

IBMBOCLC: Explicit close.

IBMBOCLD: Implicit close.

IBMBOCLG: Implicit open by in-line compiled code.

|Note: On CICS these functions are provided by entry points to DFHPL1I
|which acts as a bootstrap to the transient library modules.

Method

Entry Points IBMBOCLA, B, C, and D:

These four entry points of the module share the same code. The entry point that was called is determined by examining the contents of the caller's branch register.

The link-edit name of the required transient module is IBMDOCAA for Close and IBMDOP*A for Open, where * is filled in from the FCB, having been set there by the compiler. The link-edit name is set up in workspace in a field called ZNAM. The module then acquires storage for the transient OPEN/CLOSE modules and if necessary, any buffers and loads the module specified in ZNAM.

The callers branch register is then exmained to see whether IBMDOCL was entered at an "explicit" or an "implicit" entry point, and the required entry point of the transient open or close module is set up in the branch register. IBMDOCL then branches and links to the transient module.

On return from the transient open or close module, the storage that was obtained for the transient module is freed. For Open, part of the space obtained may have been used for the transmitter, in which case the actual amount of the space to be freed will have already been modified, causing IBMDOCL to automatically free only the right amount. A test is then made to determine whether or not the operation that has just been performed was an implicit open. If it was not, control is returned to the caller. If it was, the following action is taken.

The module chains back to the previous DSA. If this is not a compiled code DSA, it must belong to IBMDRIO (chapter 4); i.e., the record I/O statement that caused the implicit open is being implemented by the library. In this case, IBMDOCL chains back to the compiled code DSA and reestablishes the compiled code environment. It then executes a branch and link on the branch register, i.e., it recalls IBMBRIOA to execute the I/O statement.

If the previous DSA is a compiled code DSA, the record I/O statement that caused the implicit open is being implemented by compiled code. In this case, IBMDOCL sets the branch register save slot to the address of the LIOCS routine, and then restores the compiled code environment. It then tests the FCB to determine whether the file is an INPUT or and OUTPUT file, and branches to the GET or the PUT entry point of the LIOCS routine as appropriate.

Entry Point IBMBOCLG:

For consecutive buffered input and output files, the calls to the LIOCS routine for GET and PUT may be made directly from compiled code.

The address of the LIOCS routine is held in a field in the DTF. Before the file has been opened, this field contains the address of a point in IBMDOCL 8 bytes before entry point IBMBOCLG. Compiled code calls the LIOCS routine by branching to an address at an offset of either 8 or 12 from the LIOCS address in the DTF; an attempt to effect I/O on an unopened file thus causes entry to IBMBOCLG.

IBMBOCLG sets register 15 to point to entry point IBMBOCLB (implicit open) and then branches on it.

Linkage

Entry Point IBMBOCLA:

```
R1    = A(PLIST)
PLIST = A(number of files)
        A(FCB)
        A(OCB) or zero
        A(title) or zero
        A(pagesize) or zero
        A(linesize) or zero
```

The last five items in the parameter list are repeated for each file that is to be opened.

Entry Point IBMBOCLB:

```
R1    = A(PLIST)
PLIST = A(FCB)
        A(RCB)

R2    = A(FCB)
```

Entry Point IBMBOCLC:

```
R1    = A(PLIST)
PLIST = A(number of files)
        A(FCB)
        A(disposition word)
```

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

Entry Point IBMBOCLD:

No parameters are passed to this entry point.

Entry Point IBMBOCLG:

```
R2    = A(FCB)
```

Calls

IBMDOPM - Open module (transient).
IBMDOPP - Open module (transient).
IBMDOPS - Open module (transient).
| IBMDOPV - Open module - VSAM (transient).
IBMDOPX - Open module (transient).
IBMDOCA - Close module (transient).
| IBMDOCV - Close module - VSAM (transient).
IBMDPGR - Storage management.

Called By

Entry Points IBMBOCLA, C, and G:

Compiled code.

Entry Point IBMBOCLB:

IBMDRIO - Record I/O interface module.

Entry Point IBMBOCLD (at termination):

IBMDPIR - Program initialization from System.

CHAPTER 4: RECORD I/O ROUTINES

The resident library contains one module concerned with RECORD I/O: IBMDRIO. This module provides an interface between compiled code and the DOS PL/I Transient Library. The transient library is described in DOS PL/I Transient Library: Program Logic.

MODULE DESCRIPTIONS

IBMDRIO - Record I/O Interface Module

Function

To act as an interface between the record I/O transmitter modules and the calling programs and to load record I/O error modules. The module has four entry points:

IBMBRIOA: Check the validity of the I/O statement and call the transmitter.

IBMBRIOB: Load a record I/O error module and branch to it.

IBMBRIOC: Raise ERROR for invalid statements.

IBMBRIOD: Interface between record I/O statements implemented by in-line code and the record I/O error module.

Method

Entry Point IBMBRIOA: IBMBRIOA loads the parameters required by the record I/O transmitters into registers (see Linkage below) and then tests the validity of the information in the request control block (RCB) by executing a test under mask instruction in the RCB. If the information is valid, the routine branches to an address held in field FATM in the file control block (FCB). This field holds the address of the appropriate transmitter if the file is open, or the address of the implicit open routine IBMBOCLB if it is not. If the information in the RCB is invalid, the routine branches to the address held in field FAIS in the FCB. This field holds the address of entry point IBMBRIOC if the file is open, or the address of the implicit open routine IBMBOCLB if it is not.

Entry Point IBMBRIOB : IBMBRIOB creates the name of the required error module by inserting the 7th letter, obtained from the FCB, into the skeleton name IBMDRE*A. The routine then scans the the transmitter chain, searching for the required module. If the module is found, its responsibility count is incremented by one; otherwise, IBMBRIOB obtains storage, loads the error module, and adds it to the chain. Finally, IBMBRIOB loads the address of the error module into the FCB and branches to it.

Entry Point IBMBRIOC: IBMBRIOC branches to the error handler IBMBERRB, passing the address of the file name and the "unsupported op" error code.

Entry Point IBMBRIOD: This entry point is entered when compiled code is executing a record I/O statement in-line and a condition is to be raised.

If the required record I/O error module is already loaded, it is entered using the address in the FCB. Otherwise, the error module is loaded and entered as described for entry point IBMBRIOB above.

Linkage

Entry Point IBMBRIOA:

Input:

```
R1    = A(PLIST)
PLIST = A(FCB)
        A(RCB)
        A(RD) or A(ignore factor) or A(target for buffer address
          in READ SET statements)
        A(KD) or zero
        A(EV) or zero
        A(abnormal locate return address for LOCATE statements)
```

Output to transmitter:

```
R1    = A(PLIST)  (see above)
R2    = A(FCB)
R5    = A(RCB)
R6    = A(RD) or A(ignore factor) or A(target for buffer address
          in READ SET statements)
R7    = A(KD)
R8    = A(EV)
```

Entry Points IBMBRIOB, C, and D:

```
R2    = A(FCB)
```

Calls

IBMDERR - Error handler.
IBMDOCL - Open/close bootstrap (implicit open).
IBMDR** - RECORD I/O transmitter.
IBMDRE* - RECORD I/O error module.

Called By

Compiled code
IBMDJWT - WAIT module.
IBMGJWT - WAIT module.

CHAPTER 5: STREAM I/O ROUTINES

Communication between compiled code and the library is via a STREAM I/O control block (SIOCB), which exists for the duration of a single GET or PUT statement and is situated in the compiled code DSA. A full discussion of the organization of STREAM I/O and a description of the SIOCB are given in DOS PL/I Optimizing Compiler: Execution Logic. The use of the SIOCB in stream input/output is shown in Figure 5.1

The resident library modules used for STREAM I/O may be divided into the following categories:

1. Initialization modules.

2. Input/output director modules.

3. Conversion director modules.

4. Format and option modules.

5. Transmitter modules.

In the execution of a GET FILE or PUT FILE statement, the first call from compiled code is to an initialization module. The initialization modules are:

IBMDSII - GET FILE initialization.

IBMDSIL - GET and PUT FILE initialization.

IBMDSIO - PUT FILE initialization.

The initialization module tests the validity of the I/O statement, opens the file if necessary by a call to module IBMDOCL, and completes the SIOCB. If the statement contains the PAGE, LINE, or SKIP option, the initialization module calls the appropriate option module to handle it.

A fourth initialization module, IBMDSIS, is provided to handle GET and PUT STRING statements.

The movement of data items between external media and PL/I variables involves two separate processes: the movement of records between the external medium and an internal buffer, and the movement of data items between the buffer and the PL/I variable. There is not necessarily a one-to-one correspondance between these moves, since a record may contain more than one data item, and a data item may span one or more record boundaries.

In general, the movement of data is controlled by input/output director modules. These modules are:

IBMDSDI - Data-directed input.
IBMDSDO - Data-directed output.
IBMDSEI - Edit-directed input.
IBMDSED - Edit-directed input/output.
IBMDSEO - Edit-directed output.
IBMDSLI - List-directed input.

IBMDSLC - List-directed output.
IBMDSEE - Edit-directed combination module.
IBMDSEH - Edit-directed combination subset module.
IBMDSDJ - Data-directed input (restricted conversions).
IBMDSLJ - List-directed input (restricted conversions).

The edit-directed modules IBMDSEI and IBMDSEO are called only when a data item spans a record boundary and the conversion, if any, is to be done by compiled code. Module IBMDSED is used for complete library control over the edit-directed input or output of a single item.

The director modules effect movement of records between the external medium and the buffer by calling the appropriate STREAM I/O transmitter module. With the exception of modules IBMDSTF and IBMDSTI, which are described in this chapter, the STREAM I/O transmitter modules are in the transient library and are described in the DOS PL/I Transient Library: Program Logic.

The movement of data from the buffer to a PL/I variable involves the identification of the required data item and the selection of the required conversion. If the end of the record is reached before the end of the data item, temporary storage must be obtained for the first part of the data item and the next record read. Similarly, on output, the data item must be converted to the required external format and placed in the buffer or buffers.

The input/output directors thus have to acquire the following information:

1. Where in the record is the required variable situated, or where in the record is the variable to be placed.

2. Must a new record be written or read.

3. Does the variable span a record boundary.

4. What conversion is needed.

GET and PUT STRING statements are handled by the input/output directors just as if they were GET and PUT FILE statements. A dummy FCB for the statement is set up by the initialization module, and the source or target string, as appropriate, is treated as an I/O buffer. For "output", the "transmitter" addressed by the dummy FCB is entry point IBMBERRB of the error handler. Thus any attempt to move a string that is too large for the "buffer" results in an entry to the error handler. For "input", the "transmitter" consists of code set up by module IBMBSIS in the dummy FCB. This code sets the end-of-file flag and then returns.

Conversions in edit-directed I/O are handled by a number of conversion director modules which select the appropriate conversion modules in the conversion package (chapter 6). The conversion director modules are:

IBMBSAI - Input conversion director
(A, character-P, and B formats)
IBMBSAO - Output conversion director
(A format)
IBMBSBO - Output Conversion director
(character-P and B formats)
IBMBSCI - Input conversion director
(C format)

IBMBSCO - Output conversion director
(C format)

IBMBSFI - Input conversion director
(F and E formats)

IBMBSFO - Output conversion director
(F and E formats)

IBMBSPI - Input conversion director
(arithmetic-P format)

IBMBSPO - Output conversion director
(arithmetic-P format)

module name *	SIOCB Fields Referenced	Altered	comments
IBMDSII or IBMDSIL	STYP		code defining the type of statement
	SRTN		address of the dummy label for the next statement
	SDSA		DSA level number (for GET DATA)
		SFCB	address of the FCB
		SFLG	cleared
		SCNT	cleared
		SOCA	address of the ONCA
IBMDSIO	STYP		code defining the type of statement
		SFCB	address of the FCB
	SDSA		DSA level number (for PUT DATA)
		SFLG	cleared
		SDFL	cleared
		SCNT	cleared
		SOCA	address of the ONCA
IBMDSIS	STYP		code defining type of statement
	SDSA		DSA level number (for GET/PUT DATA)
		SFCB	address of the dummy FCB
		SFLG	cleared
		SDFL	cleared
		SOCA	address of the ONCA
		SSTR	the dummy FCB
IBMDSLI (IBMBSLIA)	STRG		address of the target data or locator
	STDD		address of the target DED
IBMDSLJ (IBMBSLJA)	STRG		address of the target data or locator
	STDD		address of the target DED
IBMDSLO (IBMBSLOA)	SSRC		address of the source data or locator
	SSDD		address of the source DED
	STRG		address of varying string for name
	STDD		address of subscripts string locator
			Fields STRG and STDD are used only when the module is called for PUT DATA. Note that when STDD = 0, only the name field addressed by STRG is put out by IBMBSLOA. In this case fields SSRC and SSDD are not used. This technique is used for data-directed output (including CHECK) of program control data.
IBMDSEI	SSDD		address of the source format DED
		SSRC	address of the source in a buffer or a VDA

* NOTE: In the above figure, where the information is applicable only to a specific entry point, then that entry point name is given in parenthesis, after the module name.

Figure 5.1. (Part 1 of 2). Use of the stream input/output control block (SIOCB)

module name *	SIOCB Fields Referenced	Altered	comments
IBMDSEO	STRG		address of VDA containing the converted field
	STDD		address of the target format DED
IBMDSED (IBMBSEDA)	SSDD		address of the source format DED
	STRG		address of the target data or locator
	STDD		address of the target DED
IBMDSED (IBMBSEDB)	SSRC		address of the source data or locator
	SSDD		address of the source DED
	STDD		address of the target format DED
IBMDSEE (IBMBSEEA)	SSDD		address of the source format DED
	STRG		address of the target data or locator
	STDD		address of the target DED
IBMDSEH (IBMBSEHA)	SSRC		address of the source data or locator
	SSDD		address of the source DED
	STDD		address of the target format DED
IBMDSEH (IBMBSEHB) or (IBMBSEHC)	STDD		address of a format DED containing the parameter in the second halfword (PUT statement only)
IBMDSPL	SSDD		address of a format DED containing the parameter in the second halfword (GET statement only)
	STDD		address of a format DED containing the parameter in the second halfword (PUT statement only)
IBMDSXC (IBMBSXCA) or (IBMBSXCC)	SSDD		address of a format DED containing the parameter in the second halfword (GET statement only)
IBMDSXC (IBMBSXCB) or (IBMBSXCD)	STDD		address of a format DED containing the parameter in the second halfword (PUT statement only)

* NOTE: In the above figure, where the information is applicable only to a specific entry point, then that entry point name is given in parenthesis, after the module name.

Figure 5.1. (Part 2 of 2). Use of stream input/output control block (SIOCB)

MODULE DESCRIPTIONS

INITIALIZATION MODULES

IBMDSII - GET FILE Initialization

Function

Tc initialize GET FILE statements, checking the validity of the statement for the file and opening the file if necessary. The SIOCB is established in the area passed by compiled code. The module has seven entry points:

IBMBSIIA: GET FILE

IBMBSILA: GET FILE

IBMBSIIB: GET FILE SKIP

IBMBSILB: GET FILE SKIP

IBMBSIIC: GET FILE COPY

IBMBSIID: GET FILE SKIP COPY

IBMBSIIT: TERMINATE COPY

Method

A flowchart of the module is given in chart DSII.

Error and Exceptional Conditions

The ENDFILE condition is raised if end of file has been reached in a previous GET statement.

The ERROR condition is raised:

1. If the file referenced in the GET statement is not a stream input file.
2. If a file referenced in a COPY option is not a stream output file.
3. If further output on the copy file is prevented by a prior condition.
4. If an attempt to open the file implicitly, fails.

Linkage

Entry Points IBMBSIIA and IBMBSILA:

```
R1    = A(PLIST)
PLIST = A(FCB)
        A(SIOCB)
```

Entry Points IBMBSIIB and IBMBSILB:

```
   R1    = A(PLIST)
   PLIST = A(FCB)
           A(SIOCB)
           A(fullword SKIP parameter)
```

Entry Point IBMBSIIC:

```
   R1    = A(PLIST)
   PLIST = A(FCB)
           A(SIOCB)
           A(FCB of COPY file)
```

Entry Point IBMBSIID:

```
   R1    = A(PLIST)
   PLIST = A(FCB)
           A(SIOCB)
           A(fullword SKIP parameter)
           A(FCB of COPY file)
```

Entry Point IBMBSIIT:

```
R1    = A(SIOCB)
```

Calls

IBMDOCI - Open/close bootstrap.
IBMDSCP - COPY module.
IBMDSPL - PAGE, LINE, and SKIP options (called only for SKIP).

Called By

Compiled code.

IBMDSIL - GET FILE and PUT FILE Initialization

Function

To initialize GET FILE and PUT FILE statements, checking the validity of the statement for the file and opening the file if necessary. The SIOCB is established in the area passed by compiled code. The module is also used to terminate a PUT FILE statement. It has eight entry points:

IBMBSILA: GET FILE

IBMBSIOA: PUT FILE

IBMBSILB: GET FILE SKIP

IBMBSIOB: PUT FILE PAGE

IBMBSIOC: PUT FILE LINE

IBMBSIOD: PUT FILE PAGE LINE

IBMBSIOE: PUT FILE SKIP

IBMBSIOT: TERMINATE PUT FILE (dummy)

Method

A flowchart of the module is given in chart DSIL.

Error and Exceptional Conditions

The ENDFILE condition is raised if end of file has been reached in a previous GET statement.

The ERROR condition is raised if:

1. The file referenced in a GET statement is not a stream input file, or if the file referenced in a PUT statement is not a stream output file.

2. There is a PRINT option on a non-PRINT file.

3. Further output on an output file is prevented by a previously raised condition.

4. An attempt to open the file implicitly fails.

Linkage

Entry Points IBMBSILA, IBMBSIOA and IBMBSIOB:

```
R1    = A(PLIST)
PLIST = A(FCB)
        A(SIOCB)
```

Entry Points IBMBSILB, IBMBSIOC, IBMBSIOD and IBMSIOE:

```
R1    = A(PLIST)
PLIST = A(FCB)
        A(SIOCB)
        A(fullword SKIP or LINE parameter)
```

Calls

IBMDOCL - Open/close bootstrap.
IBMDSPL - PAGE, LINE, and SKIP options.

Called By

Compiled code.

IBMDSIO - PUT FILE Initialization

Function

To initialize PUT FILE statements. The module has six entry points:

IBMBSIOA: PUT FILE

IBMBSIOB: PUT FILE PAGE

IBMBSIOC: PUT FILE LINE

IBMBSIOD: PUT FILE PAGE LINE

IBMBSIOE: PUT FILE SKIP

IBMBSIOT: TERMINATE PUT FILE (This is a dummy entry point)

Method (chart DSIO)

The module checks the validity of the statement, opens the file if necessary by a call to IBMDOCL, and completes the SIOCB. If the PAGE, LINE, or SKIP option is present, module IBMDSPL is called.

Error and Exceptional Conditions

The ERROR condition is raised if:

1. The file referenced in the PUT statement is not a STREAM OUTPUT file.
2. A print control option is specified for a non-PRINT file.
3. Further output is prevented by a prior condition.
4. An attempt to open the file implicitly, fails.

Linkage

Entry Points IBMBSIOA and B:

```
R1    = A(PLIST)
PLIST = A(FCB)
        A(SIOCB)
```

Entry Points IBMBSIOC and D:

```
R1    = A(PLIST)
PLIST = A(FCB)
        A(SIOCB)
        A(fullword LINE parameter)
```

Entry Point IBMBSIOE:

```
R1    = A(PLIST)
PLIST = A(FCB)
        A(SIOCB)
        A(fullword SKIP parameter)
```

Entry Point IBMBSIOT:

No linkage

Calls

|IBMDOCL - Open/close bootstrap - for non-CICS systems.
IBMDSPL - PAGE, LINE, and SKIP options.
|DFHPL1I - IBMDOCL entry points - Open/close bootstrap.

Called By

Compiled code.

IBMDSCP - COPY module.
IBMBERC - CHECK (system action) module.

IBMDSIS - GET or PUT STRING Initialization

Function

To initialize GET or PUT STRING statements by creating a dummy FCB in the SIOCB, and to perform subsequent housekeeping for GET or PUT STRING statements. The module has three entry points:

IBMBSISA: Initialize GET STRING

IBMBSISB: Initialize PUT STRING

IBMBSIST: GET and PUT STRING housekeeping.

Method (chart DSIS)

Entry Point IBMBSISA:

The module sets up a dummy FCB in the SIOCB and moves a short length of code into it. The address of this code is then placed in the transmitter address field FATM in the dummy FCB.

The code moved to the dummy FCB consists of two instructions; the first is to set the end of file flag, the second is a return to the caller.

Entry Point IBMBSISB:

The module sets up a dummy FCB in the SIOCB. The address of entry point IBMBERRB of the error handler is then set up in the transmitter address field FATM and an error code is set up at the beginning of the FCB.

Entry Point IBMBSIST:

This entry point is called after the first assignment to a FIXED string and after every assignment to or from a VARYING string.

For PUT STRING statements, the module blanks out any unused bytes in a fixed length target, or updates the current length of a varying length target.

For GET STRING statements, if the target is the string itself, its characteristics are changed during the execution of the statement.

Linkage

Entry Points IBMBSISA & B:

```
R1    = A(PLIST)
PLIST = A(string locator)
        A(SIOCB)
```

Entry Point IBMBSIST:

R1 = A(SIOCB)

Called By

Compiled code.
IBMDSED - Edit-directed input/output.
IBMDSEE - Edit-directed combination module.
IBMDSEH - Edit-directed combination subset module.
IBMDSEC - Edit-directed output.
IBMDSLI - List-directed input.
IBMDSLO - List-directed output.
IBMDSXC - X and COLUMN format items.

INPUT/OUTPUT DIRECTOR MODULES

IBMDSDI - Data-directed Input

Function

To implement PL/I GET DATA statements. The module has four entry points:

IBMBSDIA: GET DATA statement with data list.

IBMBSDJA: GET DATA statement with data list.

IBMBSDIB: GET DATA statement without data list.

IBMBSDJB: GET DATA statement without data list.

Method (chart DSDI)

The module compares the names of the variables in the input stream with those in the data list (entry point IBMBSDIA/IBMBSDJA) or with all the names known at the point of execution (entry point IBMBSDIB/IBMBSDJB). When a match is found, the module evaluates the address of the target variable and then calls IBMDSLI to scan the input and convert and assign it to its target.

Error and Exceptional Conditions

The NAME condition is raised if a variable name in the input stream is not in the data list (entry point IBMBSDIA/IBMBSDJA) or is not known at the point of execution (entry point IBMBSDIB/IBMBSDJB), unless the STRING option has been specified, in which case the ERROR condition is raised.

The ENDFILE condition is raised if end-of-file is detected while scanning the leading blanks for an item. For GET STRING statements, the ERROR condition is raised if the end of the string is detected under these circumstances.

The CHECK condition is raised for each element in the input stream that may appear in a CHECK list.

The TRANSMIT condition is raised if an input error has been detected on the file in the following cases:

1. Null item.

2. Invalid item causing NAME to be raised.

3. Terminating semicolon.

4. End-of-file detected while scanning leading blanks on an item.

Linkage

Entry Points IBMBSDIA and IBMBSDJA:

```
R1    = A(PLIST)
PLIST = A(SICCB)
        A(symbol table 1)
        A(symbol table 2)
        .
        .
        .
        A(symbol table n)
```

The high order bit of the last argument is set to '1'B.

Entry Points IBMBSDIB and IBMBSDJB:

```
R1    = A(PLIST)
PLIST = A(SICCB)
        A(first list element in chain of
          symbol table elements)
```

Calls

IBMDSLI - List-directed input.
IBMDSLJ - List-directed input (restricted conversions).
Relevant stream I/O transmitter.

Called By

Compiled code.

IBMDSDJ - Data-directed Input (restricted conversions)

Function

To implement PL/I GET DATA statements. The module has two entry points:

IBMBSDJA: GET DATA statement with data list.

IBMBSDJB: GET DATA statement without data list.

Method (chart DSDJ)

The module compares the names of the variables in the input stream with those in the data list (entry point IBMBSDJA) or with all the names known at the point of execution (entry point IBMBSDJB). When a match is found, the module evaluates the address of the target variable and then calls IBMDSLI to scan the input and convert and assign it to its target.

Error and Exceptional Conditions

The NAME condition is raised if a variable name in the input stream is not in the data list (entry point IBMBSDJA) or is not known at the point of execution (entry point IBMBSDJB), unless the STRING option has been specified, in which case the ERROR condition is raised.

The ENDFILE condition is raised if end-of-file is detected while scanning the leading blanks for an item. For GET STRING statements, the ERROR condition is raised if the end of the string is detected under these circumstances.

The CHECK condition is raised for each element in the input stream that appears in a CHECK list.

The TRANSMIT condition is raised if an input error has been detected on the file in the following cases:

1. Null item.
2. An invalid item has caused the NAME condition to be raised.
3. Terminating semicolon.
4. The ENDFILE condition has been raised.

Linkage

<u>Entry Point IBMBSDJA</u>:

```
R1    = A(PLIST)
PLIST = A(SIOCB)
        A(symbol table 1)
        A(symbol table 2)
        .
        .
        .
        A(symbol table n)
```

The high order bit of the last argument is set to '1'B.

<u>Entry Point IBMBSDJB</u>:

```
R1    = A(PLIST)
PLIST = A(SIOCB)
        A(first list element in chain of
          symbol table elements)
```

Calls

IBMDSLI - List-directed input.
IBMDSLJ - List-directed input (restricted conversions).
Relevant stream I/O transmitter.

Called By

Compiled code.

IBMDSDO - Data-directed Output

Function

To implement all PL/I PUT DATA statements, and to produce equate strings for the CHECK condition. The module has five entry points.

IBMBSDOA: Scalars and whole arrays.

IBMBSDOB: Single array elements.

IBMBSDCC: All known variables.

IBMBSDOD: CHECK output of a single variable.

IBMBSDCT: Output a final semicolon only. This entry point is used for the special case of repetitive specification.

Method

A flowchart of the module is given in chart DSDO.

Linkage

Entry Point IBMBSDOA:

```
R1      = A(PLIST)
PLIST   = A(SIOCB)
          A(symbol table 1)
          A(symbol table 2)
          .
          .
          A(symbol table n)
```

The high order bit of the last argument is set to '1'B.

Entry Point IBMBSDOB:

```
R1    = A(PLIST)
PLIST = A(SIOCB)
        A(symbol table 1)
        A(array element 1)
        .
        .
        A(symbol table n)
        A(array element n)
```

The high order bit of the last argument is set to '1'B.

Entry Point IBMBSDOC:

```
R1    = A(PLIST)
PLIST = A(SIOCB)
        A(first list element in chain
          of symbol table entries)
```

Entry Point IBMBSDOD:

```
R1    = A(PLIST)
PLIST = A(SICCB)
        A(interrupt control block)
```

Entry Point IBMDSDOT:

```
R1    = A(SICCB)
```

Calls

IBMDSLO - List-directed output.
Stream output transmitter.
(for CHECK only)

Called By

Compiled code.
IBMBERC - CHECK (system action) module.

IBMDSED - Edit-directed Input/Output

Function

To control the housekeeping for an item of edit-directed input or output. The module has two entry points:

IBMBSEDA: Control the edit-directed input of one item.

IBMBSEDB: Control the edit-directed output of one item.

Method

A flowchart of the module is given in chart DSED.

Error and Exceptional Conditions

The TRANSMIT condition can occur in this module.

The ERROR condition is raised if:

1. End of file is encountered in the middle of a data item.
2. An attempt is made to read or write beyond the end of the named string in a GET or a PUT string statement.

The ENDFILE condition is raised if end-of-file is detected at a legitimate point in the input.

Linkage

R1 = A(SIOCB)

Calls

IBMDSIS - GET or PUT STRING initialization.
Stream I/O transmitter module.
Conversion format director modules.

Called By

Compiled code.

IBMDSEE - Edit-directed Combination Module

Function

To control the housekeeping for a data item or for an X or COLUMN format item in edit-directed input or output. The module has eight entry points:

IBMBSEEA: Edit-directed input of a data item.

IBMBSEHA: Edit-directed output of a data item.

IBMBSXCA: X format input.

IBMBSXCB: X format output.

IBMBSEHB: X format output.

IBMBSXCC: COLUMN format input.

IBMBSXCD: COLUMN format output.

IBMBSEHC: COLUMN format output.

Method

A flowchart of the module is given in chart DSEE.

Error and Exceptional Conditions

The TRANSMIT condition or the ENDFILE condition may be raised by this module.

The ERROR condition is raised for:

1. End-of-file encountered in the middle of a data item.
2. A GET or PUT STRING that exceeds the string size.
3. An invalid control format on a GET or PUT STRING statement.

Linkage

R1 = A(SIOCB)

Calls

IBMDSIS - GET or PUT STRING Initialization.
IBMBSMW - Missing output width module.
IBMBSAI - Input conversion director.
IBMBSBO - Output conversion director.
IBMBSPI - Input conversion director.
IBMBSPO - Output conversion director.
IBMBCCA - Conversion (character to arithmetic).
IBMBCCB - Conversion (character to bit).
IBMBCCQ - Conversion director (character to pictured character).
IBMBCAC - Conversion director (arithmetic to character).
IBMBCBC - Conversion (bit to character).
IBMBCCC - HIGH, LOW, Assign (character strings).
IBMDCCS - String conversion director bootstrap.
IBMBCH - Conversion (fixed binary - float - free decimal).
IBMBCM - Conversion (pictured decimal to packed decimal).
IBMBCV - Conversion (packed decimal to E-format).
IBMBCW - Conversion (packed decimal to F-format).
IBMBCT - Conversion (decimal constant to packed decimal).
Appropriate STREAM I/O transmitter.

Called By

Compiled code.

IBMDSEH - Edit-directed Combination Subset Module

Function

To control the housekeeping for a data item or for an X or COLUMN format item in edit-directed output. The module has three entry points:

IBMBSEHA: Edit-directed output of a data item.

IBMBSEHB: X format output.

IBMBSEHC: COLUMN format output.

Method

A flowchart of the module is given in chart DSEH.

Error and Exceptional Conditions

The ERROR condition is raised:

1. If an attempt is made to write beyond the end of the named string in a PUT STRING statement.

2. For an invalid control format on a PUT STRING statement.

Linkage

R1 = A(SIOCB)

Calls

IBMDSIS - GET or PUT STRING Initialization.
IBMBSMW - Missing output width module.
IBMBSBO - Output conversion director.
IBMBSPO - Output conversion director.
IBMBCAC - Conversion director (arithmetic to character).
IBMBCBC - Conversion (bit to character).
IBMBCCC - HIGH, LOW, Assign (character strings).
IBMDCCS - String conversion director bootstrap.
IBMBCH - Conversion (fixed binary - float - free decimal).
IBMBCM - Conversion (pictured decimal to packed decimal).
IBMBCV - Conversion (packed decimal to E-format).
IBMBCW - Conversion (packed decimal to F-format).
Appropriate stream I/O transmitter.

Called By

Compiled code.

IBMDSEI - Edit-directed Input

Function

To perform the housekeeping for an edit-directed item spanning a record boundary. The module has two entry points:

IBMBSEIA: Housekeeping for spanning item.

IBMBSEIT: Raise TRANSMIT for edit-directed input that has been partially controlled by compiled code.

Method

A flowchart of the module is given in chart DSEI.

Error and Exceptional Conditions

The TRANSMIT and the "unexpected end-of-file" error conditions can occur in this module.

The ENDFILE condition is raised if end-of-file is detected at a legitimate point in the input.

The ERROR condition is raised if an attempt is made to read beyond the end of the named string in a GET STRING statement.

Linkage

R1 = A(SIOCB)

Calls

IBMDERR - Error handler.
Relevant STREAM I/O transmitter.

Called By

Compiled code.

IBMDSEO - Edit-directed Output

Function

To perform the housekeeping for an edit-directed output item spanning a record boundary.

Method

A flowchart of the module is given in chart DSEO.

Error and Exceptional Conditions

The ERROR condition is raised if an attempt is made to write beyond the end of a string in a PUT STRING statement.

Linkage

R1 = A(SIOCB)

Calls

IBMDSIS - GET or PUT STRING initialization.
Relevant stream I/O transmitter.

Called By

Compiled code.

IBMDSLI - List-directed Input

Function

To move data from the input buffer to a PL/I scalar or array variable specified in a GET LIST statement and to direct conversion of the data from its external form to its internal form according to the rules of list-directed input. The module has four entry points:

IBMBSLIA: Scalar data.

IBMBSLJA: Scalar data.

IBMBSLIB: Array data.

IBMBSLJB: Array data.

Method (chart DSLI)

If an input item spans a record boundary, a VDA is obtained large enough to hold that part of the item in the first record and also the whole of the second record. If the item then spans a further record boundary, a second VDA is obtained large enough to contain the next record, and so on. Successive VDAs are chained together. Finally, if more than one VDA has been obtained, a further VDA is set up to contain the whole item.

Error and Exceptional Conditions

The ERROR condition is raised if:

1. End of file occurs in the middle of a data item.

2. If, for a GET STRING statement, an attempt is made to read beyond the end of the string.

TRANSMIT is raised if there is a permanent transmission error on the file.

CONVERSION will be raised by this module if the attributes of the source can not be determined, e.g., 'ABC'34.

STRINGSIZE is raised if an attempt is made to assign a string too large for the target.

ENDFILE is raised if end of file is found between data items.

Linkage

Entry Points IBMBSLIA and IBMBSLJA:

```
R1    = A(SIOCB)
```

Entry Points IBMBSLIB and IBMBSLJB:

```
R1    = A(PLIST)
PLIST = A(SIOCB)
        A(array locator)
        A(DED)
        A(halfword number of dimensions)
```

Calls

IBMBCCA - Conversion director (character to arithmetic).
IBMBCCB - Conversion (character to bit).
IBMBCCQ - Conversion (character to pictured character).
IBMDCCS - String conversion director bootstrap.
IBMBCP - Conversion (bit to binary).
IBMDSCV - Conversion fix-up bootstrap.
IBMDSIS - GET or PUT STRING initialization.
Stream I/O transmitter

Called By

IBMDSDI - Data-directed input.
IBMDSDJ - Data-directed input.
Compiled code.

IBMDSLJ - List-directed Input (restricted conversions)

Function

To move data from the input buffer to a PL/I scalar or array variable specified in a GET LIST statement, and to direct conversion of the data from its external form to its internal form according to the rules of restricted list-directed input. The module has four entry points:

IBMBSLIA: Scalar data.

IBMBSLIB: Array data.

IBMBSLJA: Scalar data.

IBMBSLJB: Array data.

Method (chart DSLJ)

If an input item spans a record boundary, a VDA is obtained that is large enough to hold that part of the item in the first record and also the whole of the second record. If the item then spans a further record boundary, a second VDA is obtained large enough to contain the next record, and so on. Successive VDAs are chained together. Finally, if more than one VDA has been obtained, a further VDA is set up to contain the whole item.

Error and Exceptional Conditions

The ERROR condition is raised:

1. If end-of-file occurs in the middle of a data item.

2. If, for a GET STRING statement, an attempt is made to read beyond the end of the string.

TRANSMIT is raised if there is a permanent transmission error on the file.

CONVERSION is raised:

1. If the attributes of the source can not be determined, for example, 'ABC'34.

2. If, because of the rules of restricted list-directed input, the required conversion is not one supported by the module.

STRINGSIZE is raised if an attempt is made to assign a string too large for the target.

ENDFILE is raised if end-of-file is found between data items.

Linkage

Entry Points IBMBSLIA and IBMBSLJA:

R1 = A(SIOCB)

Entry Points IBMBSLIB and IBMBSLJB:

```
R1    = A(PLIST)
PLIST = A(SIOCB)
        A(array locator)
        A(DED)
        A(halfword number of dimensions)
```

Calls

IBMBCCA - Conversion director (character to arithmetic).
IBMBCCB - Conversion (character to bit).
IBMBCCQ - Conversion (character to pictured character).
IBMDSCV - Conversion fix-up bootstrap.
IBMDSIS - GET or PUT STRING initialization.
Stream I/O transmitter

Called By

IBMDSDI - Data-directed input.
IBMDSDJ - Data-directed input.
Compiled code.

IBMDSLO - List-directed Output

Function

To move data from a PL/I scalar or array variable specified in a PUT LIST statement to the output buffer and to direct conversion of the data from its internal to its external form. The module has two entry points:

IBMBSLOA: scalar data.

IBMBSLOB: array data.

Method

A flowchart of the module is given in chart DSLO.

Error and Exceptional Conditions

The ERROR condition is raised if an attempt is made to write beyond the end of the string in a PUT STRING statement.

Linkage

Entry Point IBMBSLOA:

```
R1    = A(SIOCB)
```

Entry Point IBMBSLOB:

```
R1    = A(PLIST)
PLIST = A(SICCB)
        A(array locator)
        A(DED)
        A(halfword number of dimensions)
```

Calls

IBMBCAC - Conversion director (arithmetic to character).
IBMBCBC - Conversion (bit to character).
IBMDSIS - GET or PUT STRING initialization.
Stream I/O transmitter.

Called By

Compiled code.
IBMDSDO - Data-directed output (Entry point IBMBSLOA only).

IBMBSMW - Missing Output Width Module

Function

To calculate the field width for all source data types for both A and B output formats. The calculations of the field width are based upon the rules of PL/I conversions.

Method

A flowchart of the module is given in chart BSMW.

Linkage

```
R1   =   A(SIOCB)

     with the following fields set:

SSRC =   A(source data or locator)
SSDD =   A(source DED)
STDD =   A(skeleton target FED to be completed)
```

Called By

Compiled code
IBMDSEE - Edit-directed combination module.
IBMDSEH - Edit-directed combination subset module.

CONVERSION DIRECTOR MODULES

IBMBSAI - Input Conversion Director (A, character-P, and B Formats)

Function

To direct A-format, character-P-format, and B-format input conversions.

Method

A flowchart of the module is given in chart BSAI.

Error and Exceptional Conditions

The ERROR condition is raised for invalid format specifications.

Linkage

R1 = A(SICCB)

Calls

IBMBCCA - Conversion director (character to arithmetic).
IBMBCCB - Conversion (bit to bit).
IBMBCCC - Assign (character string).
IBMBCCQ - Conversion (character to pictured character).
IBMBCGQ - Check input (pictured character).
IBMBCP - Conversion (bit to fixed binary or float).

Called By

IBMDSED - Edit-directed input/output.
IBMDSEE - Edit-directed combination module.
Compiled code.

IBMBSAO - Output Conversion Director (A Format)

Function

To direct A-format conversions.

Method

A flowchart of the module is given in chart BSAO.

Error and Exceptional Conditions

STRINGSIZE can occur in this module.

Linkage

R1 = A(SIOCB)

Calls

IBMBCAC - Conversion director (arithmetic to character)
IBMBCBC - Conversion (bit to character)
IBMBCCC - Assign (character string)

Called By

Compiled code.

IBMBSBO - Output Conversion Director (character-P and B Formats)

Function

Tc direct B-format and P(character)-format conversions.

Method

A flowchart of the module is given in chart BSBO.

Error and Exceptional Conditions

CONVERSION and STRINGSIZE can occur in this module.

Linkage

R1 = A(SIOCB)

Calls

IBMDSCV - Conversion fix-up bootstrap.
IBMBCAC - Conversion director (arithmetic tc character).
IBMBCBC - Conversion (bit to character)
IBMBCBQ - Conversion (bit to pictured character)
IBMBCCQ - Conversion (character to pictured character)
IBMBCE - Conversion (fixed decimal - free decimal - float - fixed binary).
IBMBCM - Conversion (pictured decimal to packed decimal)
IBMBCR - Conversion (fixed binary or float, to bit)

Called By

Compiled code
IBMDSEE - Edit-directed combination module.
IBMDSEH - Edit-directed combination subset module.

IBMBSCI - Input Conversion Director (C Format)

Function

To direct C-format input conversion.

Method

A flowchart of the module is given in chart BSCI.

Linkage

R1 = A(SIOCB)

Calls

IBMDCCS - String conversion director bootstrap.
IBMBSFI - Input conversion director.
IBMBSPI - Input conversion director.
IBMBCGZ - Set a subfield of a complex number to zero.

Called By

Compiled code.
IBMDSED - Edit-directed input/output.

IBMBSCO - Output Conversion Director (C Format)

Function

To direct C-format output conversion.

Method

A flowchart of the module is given in chart BSCO.

Linkage

R1 = A(SIOCB)

Calls

IBMDCCS - String conversion director bootstrap.
IBMBSFO - Output conversion director.
IBMBSPO - Output conversion director.
IBMBCGZ - Set a subfield of a complex number to zero.

Called By

Compiled code.
IBMDSED - Edit-directed input/output.

IBMDSCV - Conversion Fix-up Bootstrap

Function

To load the transient module IBMDSCT into non-lifo, and to pass control to that module.

Method

The module tests the field TSCT in the TCA Appendage to see whether or not the transient module IBMDSCT has already been loaded; if it has, the field will contain the entry point address. If the field is zero, IBMDSCV loads IBMDSCT and places the entry point address of that module into TSCT. Finally, control is passed to the loaded module.

Called By

Compiled code.
IBMBCCA - Conversion director (character to arithmetic).
IBMBCCB - Conversion (character to bit).
IBMBCCQ - Conversion director (character to pictured character).

IBMBCCL - Conversion director - complex strings (transient).
IBMDCCR - Conversion director - non complex strings (transient).
IBMBCGP - Check input (pictured decimal).
IBMBCGQ - Check input (pictured character).
IBMBCT - Conversion (decimal constant to packed decimal).
IBMBCU - Conversion (binary constant to packed decimal).
IBMDSLI - List-directed input.
IBMDSLJ - List-directed input (restricted conversions).
IBMBSBO - Output conversion director.

IBMBSFI - Input Conversion Director (F and E Formats)

Function

To direct F-format and E-format input conversions.

Method

A flowchart of the module is given in chart BSFI.

Error and Exceptional Conditions

The ERROR condition is raised for invalid format specifications.

Linkage

R1 = A(SICCB)

Calls

IBMBCGZ - Set a subfield of a complex number to zero.
IBMDCCS - String conversion director bootstrap.
IBMBCT - Conversion (decimal constant to packed decimal).

Called By

IBMDSED - Edit-directed input/output.
IBMBSCI - Input conversion director.
Compiled code.

IBMBSFO - Output Conversion Director (F and E Formats)

Function

To direct F-format and E-format output conversions.

Method

A flowchart of the module is given in chart BSFO.

Linkage

R1 = A(SICCB)

Calls

IBMBCH - Conversion (fixed binary - float - free decimal).
IBMBCV - Conversion (Packed decimal to E-format).
IBMBCW - Conversion (packed decimal to F-format).
IBMDCCS - String conversion director bootstrap.
IBMBCM - Conversion (pictured decimal to decimal).

Called By

Compiled code.
IBMBSCO - Output conversion director.
IBMDSED - Edit-directed input/output.

IBMBSPI - Input Conversion Director (P Format)

Function

To direct P-format input conversions.

Method

A flowchart of the module is given in chart BSPI.

Linkage

R1 = A(SICCB)

Calls

IBMBCM - Conversion (pictured decimal to packed decimal).
IBMBCGP - Check input (pictured decimal).
IBMBCGZ - Set a subfield of a complex number to zero.
IBMBCCC - Assign (character string).
IBMBCCQ - Conversion (character to pictured character).

Called By

Compiled code.
IBMBSCI - Input conversion director.
IBMDSED - Edit-directed input/output.
IBMDSEE - Edit-directed combination module.

IBMBSPO - Output Conversion Director (P Format)

Function

To direct P-format output conversions.

Method

A flowchart of the module is given in chart BSPO.

Linkage

R1 = A(SIOCB)

Calls

IBMBCH - Conversion (fixed binary - float - free decimal).
IBMBCK - Conversion (fixed decimal - free decimal - fixed decimal).
IBMBCM - Conversion (pictured decimal to packed decimal).
IBMBCO - Conversion (packed decimal to pictured decimal).
IBMBCP - Conversion (bit to fixed binary or float).
IBMBCAC - Conversion director (arithmetic to character).
IBMDCCS - String conversion director bootstrap.

Called By

Compiled code.
IBMBSCO - Output conversion director.
IBMDSED - Edit-directed input/output.
IBMDSEH - Edit-directed combination subset module.

FORMAT AND OPTION MODULES

IBMDSCP - COPY

Function

To direct the writing of a field specified by a COPY option in a GET FILE statement onto the specified file.

Method

A flowchart of the module is given in chart DSCP.

Linkage

R1 = A(FCB of input file)

Calls

IBMDSIO - PUT FILE initialization.
Appropriate stream output transmitter.

Called By

IBMDSII - GET FILE initialization.
IBMDOCL - Open/close bootstrap.
Stream input transmitter.

IBMDSPL - PAGE, LINE, and SKIP

Function

To position a STREAM file according to the PAGE, LINE, or SKIP options or format items, raising ENDPAGE on PRINT files if necessary. The module has three entry points:

IBMBSPLA: PAGE option or format item.

IBMBSPLB: LINE option or format item.

IBMBSPLC: SKIP option or format item.

Method

A flowchart of the module is given in chart DSPL.

Error and Exceptional Conditions

The ERROR condition is raised if a PRINT option or format item is specified for a non-PRINT file.

The ENDPAGE condition is raised if the line number rises above the pagesize.

The ENDFILE condition is raised if end-of-file is encountered on input.

The ERROR condition is raised for invalid control formats on GET and PUT STRING statements.

Linkage

R1 = A(SIOCB)

Calls

Relevant stream I/O transmitter.

Called By

Compiled code.
IBMDSIO - PUT FILE initialization.
IBMDSII - GET FILE initialization.
IBMDSIL - GET FILE and PUT FILE initialization.

IBMDSXC - X and COLUMN Format Items

Function

To position a stream file according to specified X and COLUMN format items and, on output, to fill the intervening positions with blanks. The module has four entry points:

IBMBSXCA: X format item in GET statement.

IBMBSXCB: X format item in PUT statement.

IBMBSXCC: COLUMN format item in GET statement.

IBMBSXCD: COLUMN format item in PUT statement.

Method

A flowchart of the module is given in chart DSXC.

Error and Exceptional Conditions

The ENDFILE condition is raised if end-of-file is encountered at a legitimate point in the input.

The ERROR condition is raised:

1. For end-of-file occurring in the middle of an X-format field.

2. For a GET or PUT STRING that exceeds the string size.

3. For an invalid control format on a GET or PUT STRING statement.

Linkage

R1 = A(SIOCB)

Calls

Appropriate stream I/O transmitter.
IBMDSIS - GET or PUT String initialization.

Called by

Compiled code.

TRANSMITTER MODULES

IBMDSTF - PRINT F-format Transmitter

Function

To write one F-format record from an output buffer onto a data set accessed by a STREAM PRINT file. The module has two entry points:

IBMDSTFA: Normal output.

IBMDSTFB: Output of error messages when transmitter is being used for SYSPRINT.

Method

A flowchart of the module is given in chart DSTF.

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 IBMDSTFA:

R1 = A(file control block)

Entry Point IBMDSTFB:

R1 = A(string locator for message)
R2 = A(request control flag byte)

Calls

IBMDERR - Error handler.

Called By

IBMDSCP - Copy module.
IBMDSDO - Data-directed output.
IBMDSEO - Edit-directed output.
IBMDSED - Edit-directed input/output.
IBMDSLO - List-directed output.
IBMDSPL - PAGE, LINE, and SKIP formats and options.
IBMDSXC - X and COLUMN formats.
IBMDESN - Error message module.
IBMDPEP - Housekeeping diagnostic message module.
IBMDPES - Storage management diagnostic message module.
IBMDSEE - Edit-directed combination module.
IBMDSEH - Edit-directed combination subset module.

IBMDSTI - INPUT Transmitter

Function

To read one record from a data set accessed by a STREAM INPUT file into an input buffer.

Method

A flowchart of the module is given in chart DSTI.

Error and Exceptional Conditions

The TRANSMIT condition or the ENDFILE condition may be raised by this module.

Linkage

R1 = A(file control block)

Calls

IBMDERR - Error Handler.
IBMDSCP - COPY module.

Called By

IBMDSDO - Data-directed output.
IBMDSED - Edit-directed input/output.
IBMDSEO - Edit-directed output.
IBMDSLO - List-directed output.
IBMDSPL - PAGE, LINE, and SKIP formats and options.
IBMDSXC - X and COLUMN formats.

CHAPTER 6: CONVERSION ROUTINES

The conversion package of the resident library handles two basic types of conversion: external and internal. External conversions are the conversions to and from character format that are necessary in STREAM I/O. Internal conversions are conversions that arise in assignments, in the evaluation of expressions involving mixed data types, and in similar situations.

The conversion modules described in this chapter enable any valid conversion to be made between data types. Each conversion can be made by means of a single call to the library.

Each conversion is handled by a unique conversion path which passes through one, two, or three conversion modules and which may involve conversion to various intermediate data types. For example, the conversion from "float" to "fixed pictured decimal" involves a conversion from "float" to "fixed decimal" followed by a conversion from "fixed decimal" to "fixed pictured decimal."

Each conversion path is entered by means of a unique entry point in the conversion package. Because a conversion path may include up to three modules, it follows that some entry points in the conversion package are sometimes called as the first module in a conversion path and sometimes as the second or the third. If a module is called as the second or third module in a given conversion path, it is not required to store registers or to acquire workspace. Each module, therefore, determines whether or not it is the first in the path by testing a bit in the current DSA. If this bit is not set, indicating that the module is the first in the path, the module stores registers, acquires workspace, and sets the appropriate bit on for examination by the next module.

Each entry point in the conversion package is passed source and target parameters. If the conversion path contains more than one module, the source parameters are required only by the first module and the target parameters are required only by the last. Parameters must therefore be created for the intermediate forms through which the conversion passes. The modules handle the parameters in the following way:

First module: On entry, save the target parameters and create temporary target parameters for the intermediate form. On exit, make the current target parameters into source parameters for the next module.

Intermediate module: On entry, create temporary target parameters for the next intermediate form. On exit, make the current target parameters into source parameters for the next module.

Each module also passes the final target parameters to the next module in the conversion path.

INTERNAL CONVERSIONS

Internal conversions may be subdivided into two types: arithmetic and string.

Conversions from arithmetic to arithmetic and from string to string are made by means of a direct entry to the appropriate entry point of the conversion package. Direct entry points are also provided for conversions from bit string to arithmetic and vice versa.

Conversions between arithmetic and character string data types are handled by two conversion director modules: IBMBCAC (arithmetic to character) and IBMBCCA (character to arithmetic). These modules examine the attributes of the source and target data types and, if the conversion is valid, select the appropriate conversion package entry point for the conversion.

The conversion paths for conversions between internal data types are illustrated in figures 6.1 through 6.8. For any particular conversion, these figures show the conversion path taken through the conversion package and also whether or not a conversion director module is involved. The figures do not show the multiple calls that may occur in the conversion of complex data types (these are discussed below), nor do they show the return of control to the conversion directors when these are involved in the conversion.

Note: "Free decimal", which occurs in these figures, is an intermediate data format used during conversion. It comprises a 17-digit packed decimal mantissa and a fullword binary exponent.

Conversions between complex arithmetic types are made by means of two calls to the library. Conversions from complex arithmetic data types to character string are controlled by the arithmetic to character conversion director IBMBCAC, which makes two calls to the conversion package, one to convert the real part and one to convert the complex part. The insertion of special characters (e.g. "I") is done by the conversion director.

In conversions from character to arithmetic, several possibilities arise, depending on the modes of the source and the target. The following actions are taken by the conversion director:

Real to Real:	Convert to the target.
Real to complex:	Convert to the real part of the target and set the imaginary part to zero.
Complex to real:	Convert the real part of the source to the target.
Complex to complex:	Convert both parts.
Imag. to real:	Set the target to zero.
Imag. to complex:	Set the real part of the target to zero and convert the source to the imaginary part.

The conversions listed above are made by calls from the conversion director to the conversion package. A special module, IBMBCGZ, is provided for addressing the imaginary part of a complex number, or for setting one of its subfields to zero.

EXTERNAL CONVERSIONS

External conversions are conversions between internal data types and the following external formats: A-format, B-format, C-format, E-format, F-format, arithmetic-P format, and character-P format. In general these conversions are controlled by external conversion director modules, although in some instances sufficient is known about the required conversion at compile time to enable the conversion package to be called directly by compiled code. The external conversion directors, being concerned with STREAM I/O, have a fifth letter "S" in their control names; they are therefore described in chapter 5: STREAM I/O.

The external conversion directors make full use of the facilities of the conversion package, sometimes calling conversion entry points directly, and sometimes calling the internal conversion directors to assist in the conversion.

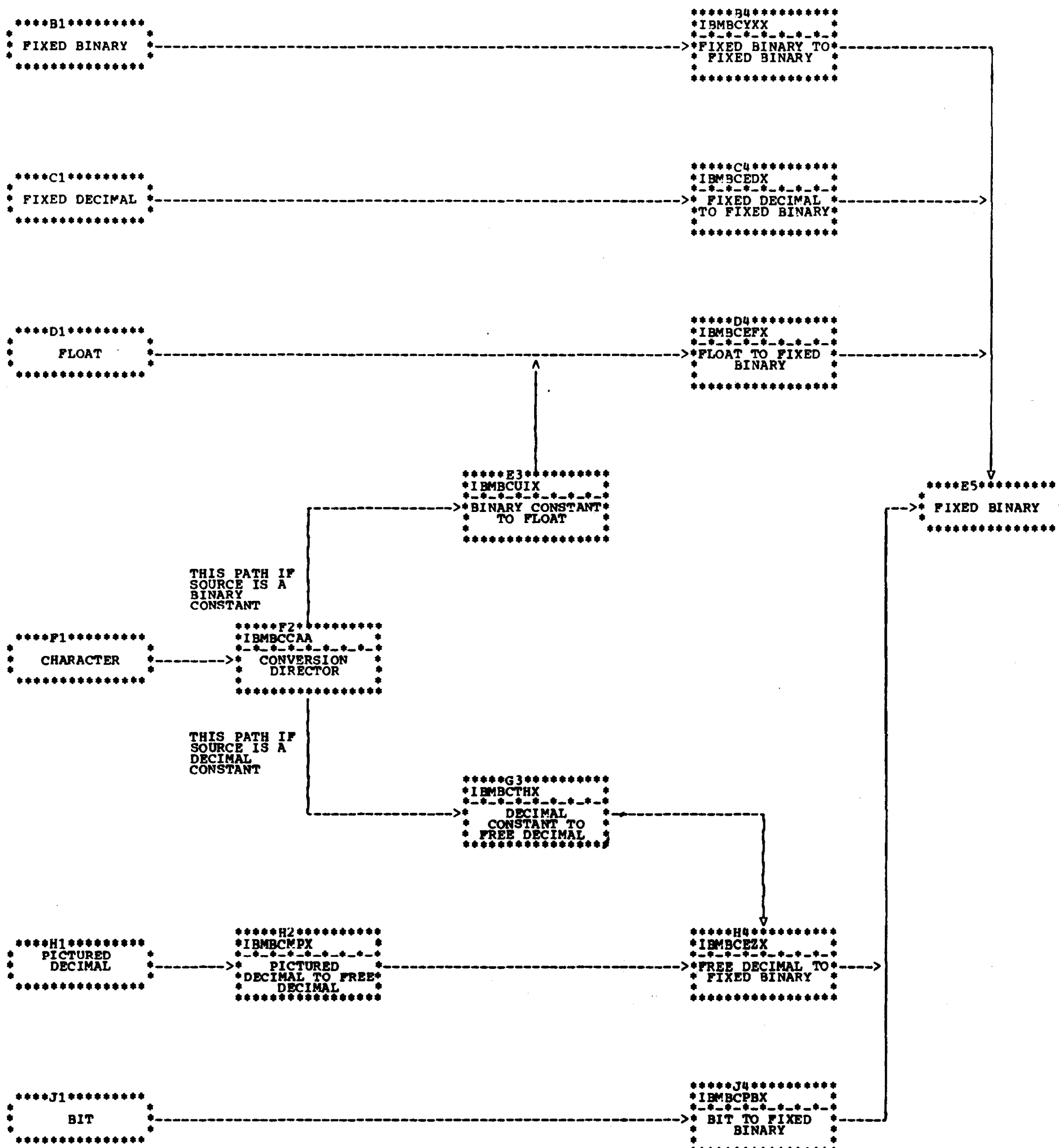


Figure 6.1: Conversion Paths; fixed binary target

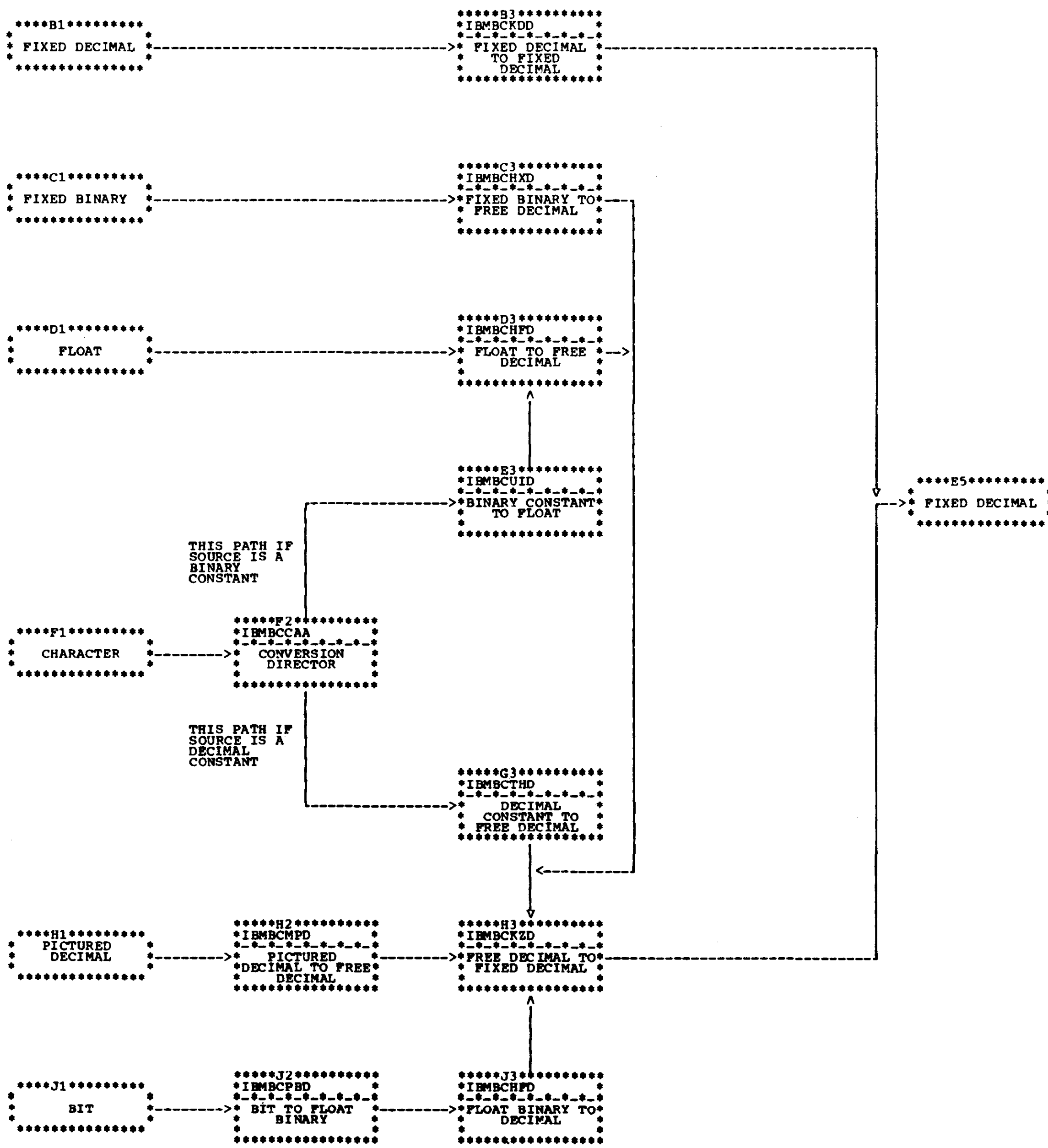


Figure 6.2: Conversion Paths; fixed decimal target

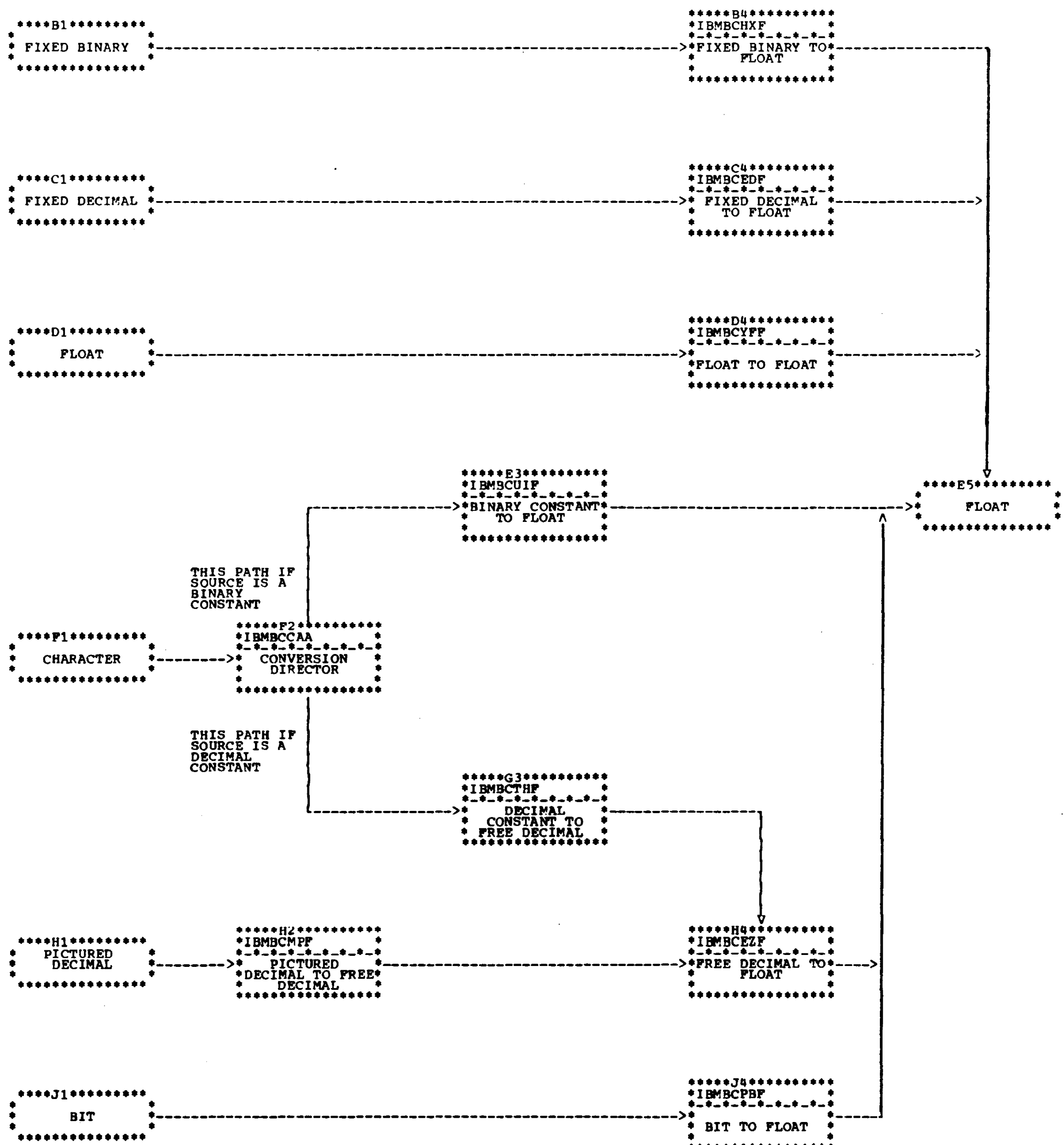


Figure 6.3: Conversion Paths; float target

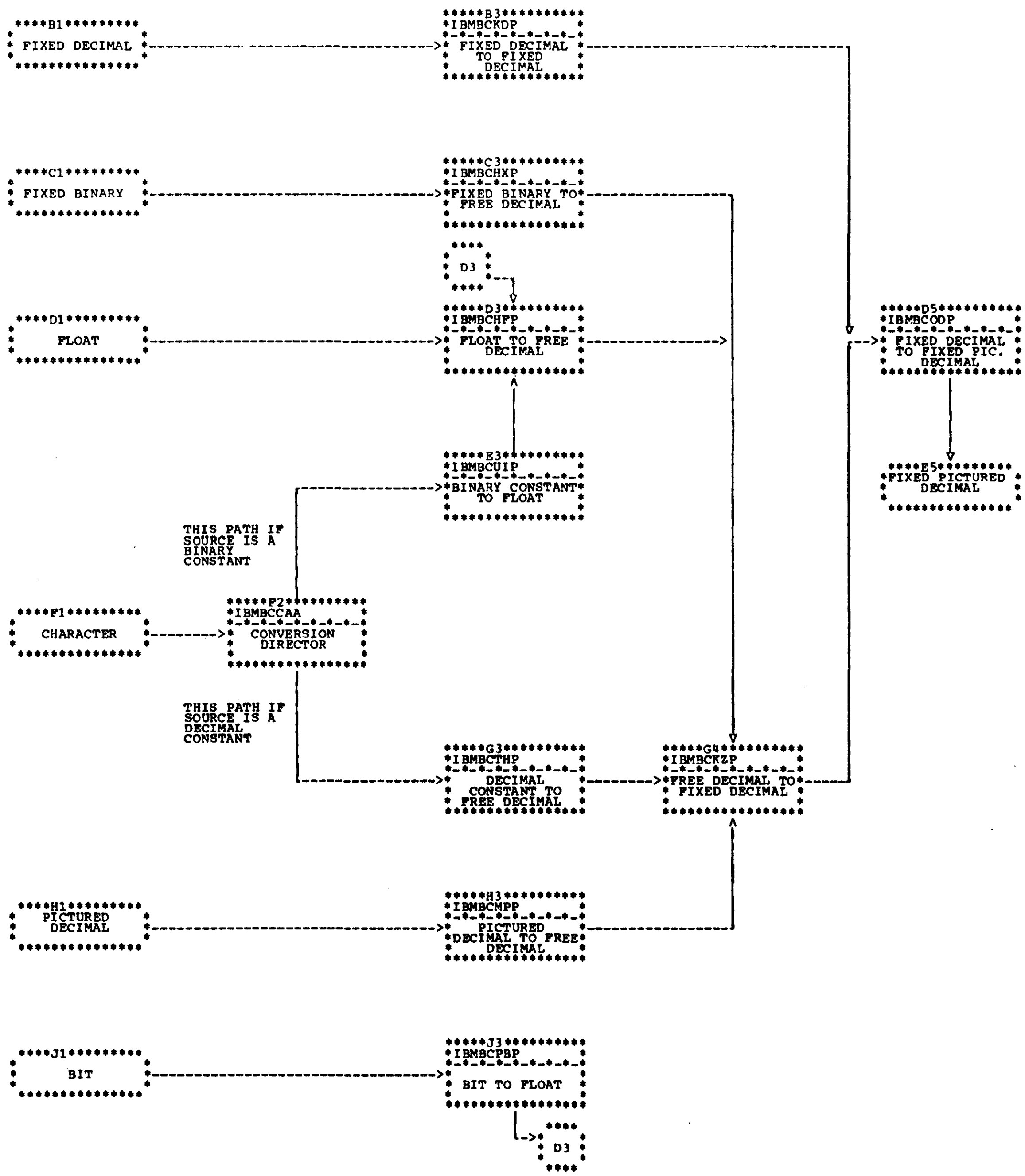


Figure 6.4: Conversion Paths; fixed pictured decimal target

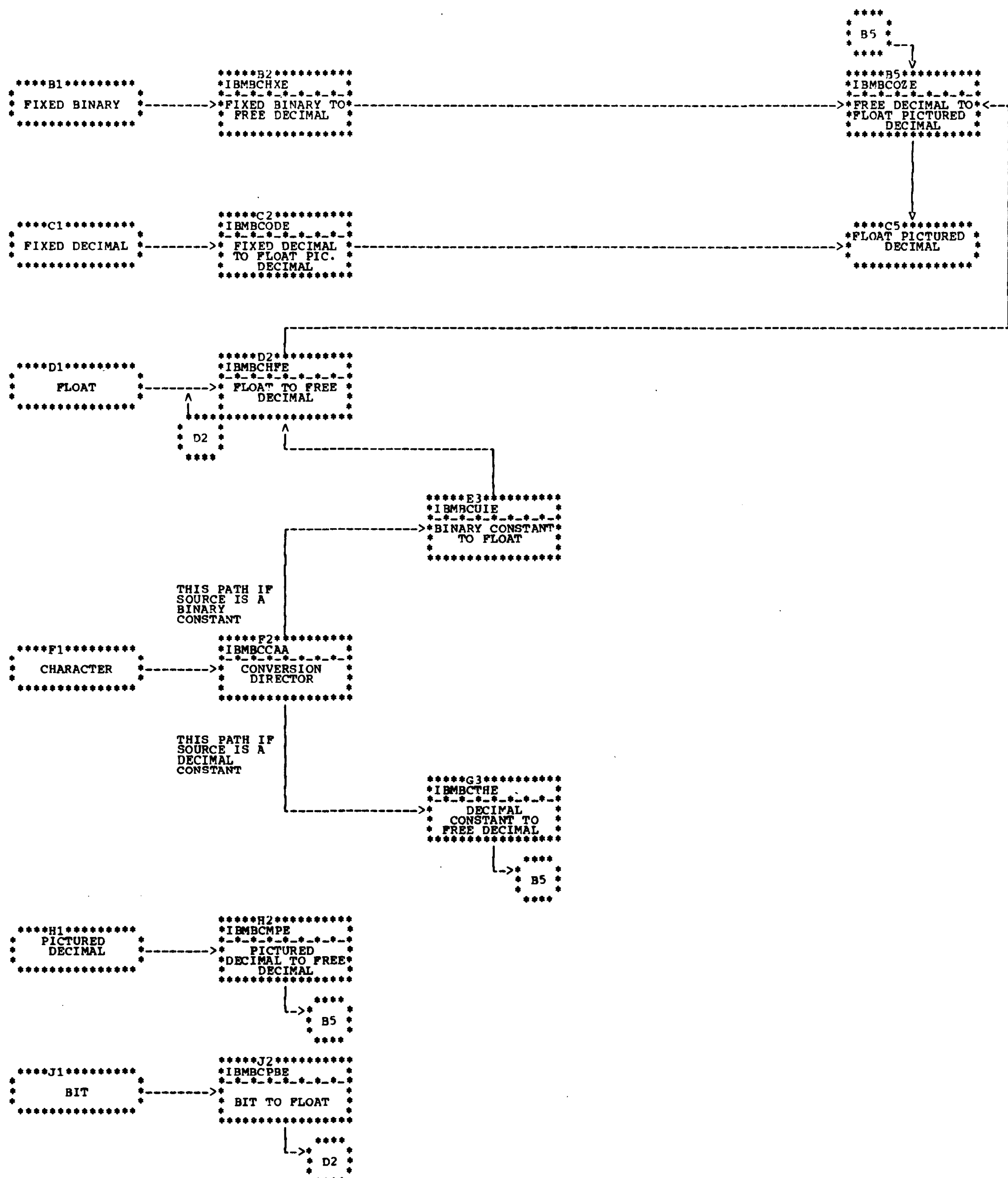


Figure 6.5: Conversion Paths; float pictured decimal target

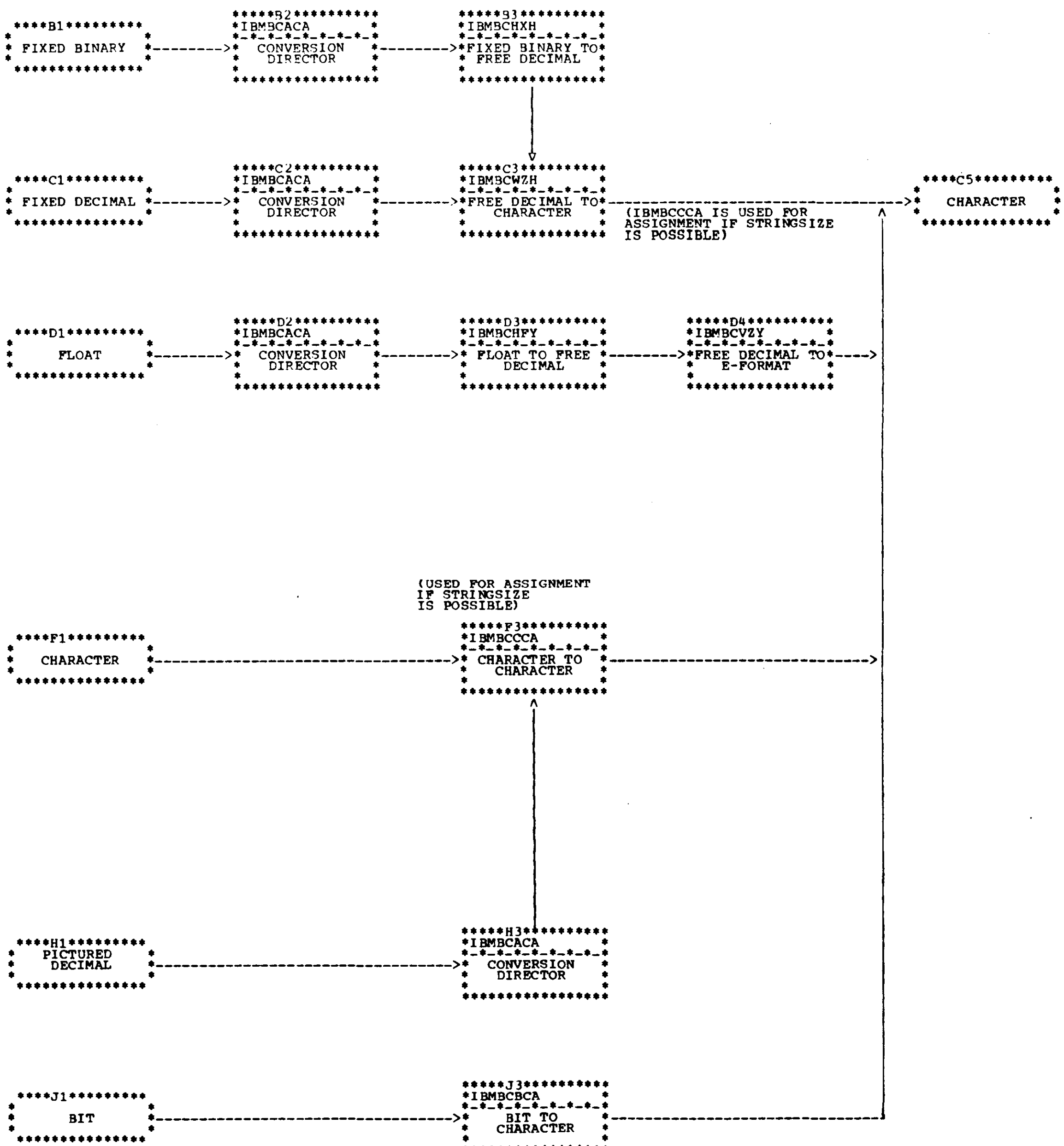


Figure 6.6: Conversion Paths; character target

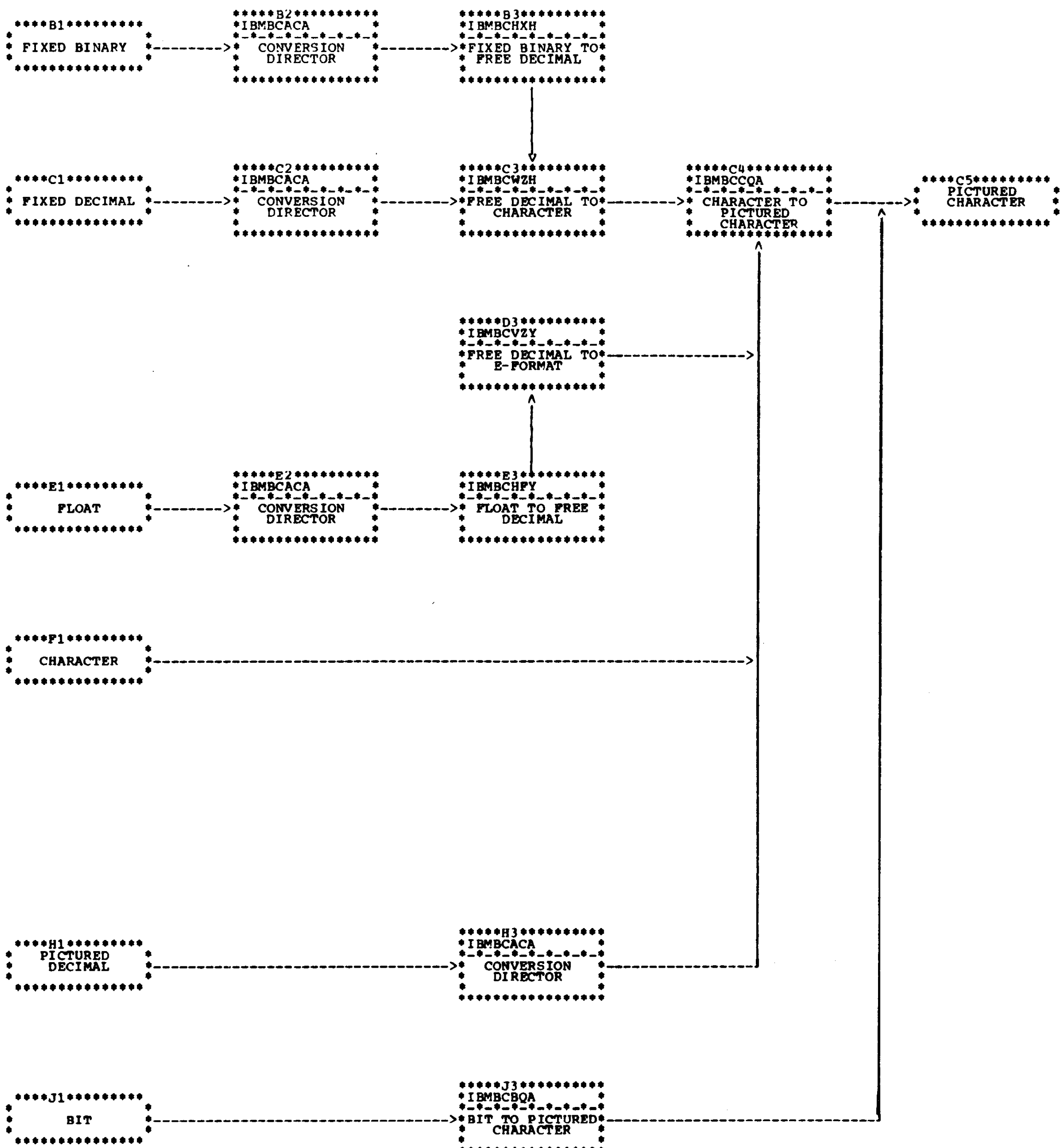


Figure 6.7: Conversion Paths; pictured character target

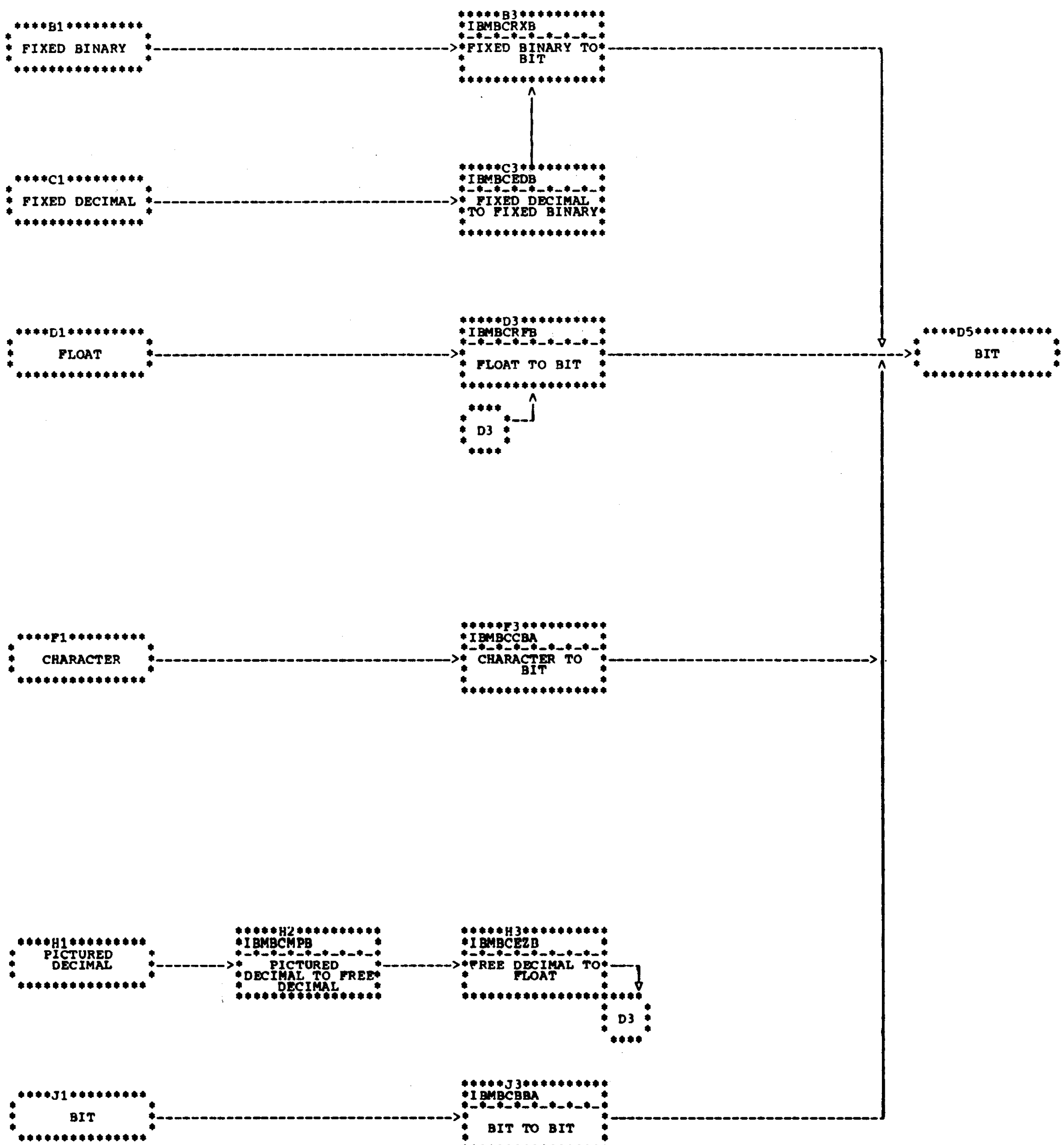


Figure 6.8: Conversion Paths; bit target

MODULE DESCRIPTIONS

IBMBCAC - Arithmetic to Character Conversion Director

Function

To direct arithmetic to character conversions.

Method

A flowchart of the module is given in chart BCAC.

Linkage

```
R1    = A(PLIST)
PLIST = A(source)
        A(source DED)
        A(target)
        A(target DED)
```

Calls

IBMBCCC - Assign (character strings).
IBMBCCQ - Conversion (character to pictured character).
IBMBCGZ - Set a subfield of a Complex number to zero.
IBMBCH - Conversion (fixed binary - float - free decimal).
IBMBCW - Conversion (packed decimal to F-format).
IBMBCV - Conversion (packed decimal to E-format).

Called By

Compiled code.
IBMBSAO - Output conversion director.
IBMBSBO - Output conversion director.
IBMBCCL - Conversion director - complex strings (transient)
IBMDCCR - Conversion director - non complex strings (transient).
IBMDSLC - List-directed output.
IBMDSEE - Edit-directed combination module.
IBMDSEH - Edit-directed combination subset module.

IBMBCBB - Conversion (Bit to Bit)

Function

To assign a fixed or varying bit string to a fixed or varying bit string target.

Error and Exceptional Conditions

STRINGSIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source locator descriptor)
        4-byte gap
        A(target locator descriptor)
```

Called By

Compiled code.
IBMBCR - Conversion (fixed or float binary to bit).
IBMBAPM - STRING psuedovariable.

IBMBCBC - Conversion (Bit to Character)

Function

To convert a fixed or varying bit string to a fixed or varying character string.

Error and Exceptional Conditions

STRINGSIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source locator descriptor)
        4-byte gap
        A(target locator descriptor)
```

Called By

Compiled code.
IBMBSAO - Output conversion director.
IBMBSBO - Output conversion director.
IBMDSLO - List-directed output.
IBMDSEE - Edit-directed combination module.
IBMDSEH - Edit-directed combination subset module.

IBMBCBQ - Conversion (Bit to Pictured Character)

Function

To convert a fixed or varying bit string to a pictured character string.

Error and Exceptional Conditions

STRINGSIZE can occur in this module. The ERROR condition is raised if a bit matches a picture character other than X or 9.

Linkage

```
R1    = A(PLIST)
PLIST = A(source locator descriptor)
        4-byte gap
        A(target locator descriptor)
        A(target DED)
```

Called By

Compiled code.
IBMBSBO - Output conversion director.

IBMBCCA - Character to Arithmetic Conversion Director

Function

To direct conversions from fixed or varying character string to any arithmetic data type, real or complex.

Method

A flowchart of the module is given in chart BCCA.

Error and Exceptional Conditions

CONVERSION can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source locator descriptor)
        A(source DED)
        A(target)
        A(target DED)
```

Calls

IBMBCT - Conversion (decimal constant to free decimal).
IBMBCU - Conversion (binary constant to float).
IBMDSCV - Conversion fix-up bootstrap.
IBMBCGZ - Set a subfield of a Complex number to zero.

Called By

Compiled code.
IBMBSAI - Input conversion director.
IBMBSFO - Output conversion director.
IBMBSPO - Output conversion director.
IBMDSLI - List-directed input.
IBMDSEE - Edit-directed combination module.

IBMBCCB - Conversion (Character to Bit)

Function

To convert a fixed or varying character string to a fixed or varying bit string.

Error and Exceptional Conditions

CONVERSION and STRINGSIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source locator descriptor)
        4-byte gap
        A(target locator descriptor)
```

Calls

IBMDSCV - Conversion fix-up bootstrap.

Called By

Compiled code.
IBMBSAI - Input conversion director.
IBMDSLI - List-directed input.
IBMDSEE - Edit-directed combination module.

IBMBCCC - HIGH, LOW, Assign (Character Strings)

Function

To assign a character string to a target and to implement the built-in functions HIGH and LOW. The module has three entry points:

IBMBCCCA: Assign a fixed or varying character string to a fixed or varying target.

IBMBCCCB: Form the character string HIGH(length) in a target field.

IBMBCCCC: Form the character string LOW(length) in a target field.

Error and Exceptional Conditions

STRINGSIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source locator descriptor)
             or A(length)
        4-byte gap
        A(target locator descriptor)
```

Called By

Compiled code.
IBMBSAI - Input conversion director.
IBMBSPI - Input conversion director.
IBMBSAO - Output conversion director.
IBMBSPO - Output conversion director.
IBMBCAC - Conversion director (arithmetic to character).
IBMDSEE - Edit-directed combination module.
IBMDSEH - Edit-directed combination subset module.

IBMDCCS - String Conversion Director Bootstrap

Function

To load and link to the transient string conversion director module IBMBCCL or IBMDCCR.

Method

The address of the transient conversion module is held in field TCCL in the TCA appendage. If the module has not been loaded, this field contains zero.

Module IBMDCCS tests field TCCL. If this field is zero, the module tests the VCON for the module IBMBCGZ (Set a subfield of a Complex number to zero). If this VCON is resolved, IBMDCCS obtains the necessary storage and loads the complex string conversion director IBMBCCL, by means of a LOAD macro. If the VCON is not resolved, IBMDCCS obtains storage and loads IBMDCCR, the non-complex string conversion director. The address of the director loaded is placed in TCCL.

The module then restores register R1, so that it addresses the parameter list passed by the caller, and points register R4 to the list of conversion package entry points contained in IBMDCCS. It then branches and links to the transient module.

On return from the transient module, control is returned to the caller.

Linkage

Input:

```
R1    = A(PLIST)
PLIST = A(source or source locator)
        A(source DED)
        A(target or target locator)
        A(target DED)
```

Output to IBMBCCL or IBMDCCR:

```
R1    = A(PLIST)  (as above)
R4    = A(list of conversion package entry points)
```

Called By

Compiled code.
IBMDSLI - List-directed input.
IBMBSFI - Input conversion director.
IBMDSLJ - List-directed input (restricted conversions).
IBMBSFO - Output conversion director.
IBMBSPO - Output conversion director.
IBMBSCI - Input conversion director.
IBMBSCO - Output conversion director.
IBMDSEE - Edit-directed combination module.
IBMDSEH - Edit-directed combination subset module.

IBMBCCQ - Conversion (Character to Pictured Character)

Function

To convert a fixed or varying character string to a pictured character string. The module has two entry points:

IBMBCCQA: Conversion, character to pictured character.

IBMBCCQB: As above, but accepts double quotes as a single quote.

Error and Exceptional Conditions

CONVERSION and STRINGSIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source locator descriptor)
        4-byte gap
        A(target locator descriptor)
        A(target DED)
```

Calls

IBMDSCV - Conversion fix-up bootstrap.

Called By

Compiled code.
IBMBSAI - Input conversion director.
IBMBSPI - Input conversion director.
IBMBSBO - Output conversion director.
IBMBCCL - Conversion director - complex strings (transient).
IBMDCCR - Conversion director - non complex strings (transient).
IBMDSLI - List-directed input (Entry point IBMBCCQB only).
IBMDSLJ - List-directed input (restricted conversions).
IBMDSEE - Edit-directed combination module.

IBMBCE - Conversion (Fixed Decimal - Free Decimal - Float - Fixed Binary)

Function

To perform conversions on the route: fixed decimal free decimal - float - fixed binary. The module has seven entry points:

IBMBCEDB: Fixed decimal to bit. This module converts to fixed binary and then branches to entry point IBMBCRXB.

IBMBCEDF: Fixed decimal to float.

IBMBCEDX: Fixed decimal to fixed binary.

IBMBCEFX: Float to fixed binary.

IBMBCEZB: Free decimal to bit. This module converts to float and then branches to entry point IBMBCRFB.

IBMBCEZF: Free decimal to float.

IBMBCEZX: Free decimal to fixed binary.

Error and Exceptional Conditions

SIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source)
        A(source DED)
        A(target)
        A(target DED)
```

Calls

IBMBCR - Conversion (fixed or float binary to bit).
IBMBCGT - Table of powers of ten.

Called By

Compiled code
IBMBCM - Conversion (pictured decimal to packed decimal).
IBMBCT - Conversion (decimal constant to packed decimal).
IBMBCU - Conversion (binary constant to float).
IBMBCAC - Conversion director (character to arithmetic).
IBMBCCL - Conversion director - complex strings (transient).
IBMDCCR - Conversion director - non complex strings (transient).
IBMBSBO - Output conversion director.
IBMBSPO - Output conversion director.

IBMBCGP - Check Input (Pictured Decimal)

Function

To check the validity of pictured decimal input.

Error and Exceptional Conditions

CONVERSION can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source)
        A(source DED)
```

Calls

IBMDSCV - Conversion fix-up bootstrap.

Called By

Compiled code.
IBMBSPI - Input conversion director.
IBMBCCL - Conversion director - complex strings (transient).
IBMDCCR - Conversion director - non complex strings (transient).

IBMBCGC - Check Input (Pictured Character)

Function

Tc check the validity of pictured character input.

Error and Exceptional Conditions

CONVERSION can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source locator descriptor)
        A(source DED)
```

Calls

IBMDSCV - Conversion fix-up bootstrap.

Called By

Compiled code.
IBMBSAI - Input conversion director.

IBMBCGT - Table of Powers of Ten

Function

The module consists of a table of powers of ten for use by the conversion modules. It does not contain any executable code.

Called By

```
IBMBCE  - Conversion (fixed decimal - free decimal - float
                      - fixed binary).
IBMBCH  - Conversion (fixed binary - float - free decimal).
```

IBMBCGZ - Set a Subfield of a Complex Number to Zero

Function

To set to zero the real or the imaginary part of a complex number, and to return the address of the imaginary part.

Method

A flowchart of the module is given in chart BCGZ.

Linkage

```
R1    = A(PLIST)
```

```
PLIST = A(target)
        A(target DED)
        A(imaginary part)
        A(request code)
```

Request codes:	'01'	Zero the real part of the number and return the address of the imaginary part.
	'10'	Zero the imaginary part of the number.
	'00'	Return the address of the imaginary part of the number.

Calls

IBMBCO - Conversion (packed decimal to pictured decimal).
IBMBCK - Conversion (fixed decimal - free decimal - fixed decimal).

Called By

Compiled Code.
IBMBSFI - Input conversion director.
IBMBCCL - Conversion director - complex strings (transient).
IBMBSPI - Input conversion director.
IBMBCAC - Conversion director (arithmetic to character).
IBMBCCA - Conversion director (character to arithmetic).
IBMBSCI - Input conversion director.
IBMBSCO - Output conversion director.

IBMBCH - Conversion (Fixed Binary - Float - Free Decimal)

Function

To perform conversions on the route: fixed binary - float - free decimal. The module has 11 entry points:

IBMBCHFD: Float to fixed decimal. This module converts to free decimal and then branches to entry point IBMBKZD.

IBMBCHFE: Float to float pictured decimal. This module converts to free decimal and then branches to entry point IBMBCOZE.

IBMBCHFH: Float to F-format. This module converts to free decimal and then branches to entry point IBMBCWZH.

IBMBCHFP: Float to fixed pictured decimal. This module converts to free decimal and then branches to entry point IBMBCKZP.

IBMBCHFY: Float to E-format. This module converts to free decimal and then branches to entry point IBMBCVZY.

IBMBCHXD: Fixed binary to fixed decimal. This module converts to free decimal and then branches to entry point IBMBCVZY.

IBMBCHXE: Fixed binary to float pictured decimal. This module converts to free decimal and then branches to entry point IBMBCOZE.

IBMBCHXF: Fixed binary to float.

IBMBCHXH: Fixed binary to F-format. This module converts to fixed decimal and then branches to entry point IBMBCWZH.

IBMBCHXP: Fixed binary to fixed pictured decimal. This module converts to free decimal and then branches to entry point IBMBCKZP.

IBMBCHXY: Fixed binary to E-format. This module converts to free decimal and then branches to entry point IBMBCVZY.

Error and Exceptional Conditions

SIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(scurce)
        A(source DED)
        A(target)
        A(target DED)
```

Calls

```
IBMBCV  - Conversion (packed decimal to E-format).
IBMBCO  - Conversion (packed decimal to pictured decimal).
IBMBCW  - Conversion (packed decimal to F-format).
IBMBCK  - Conversion (fixed decimal - free decimal - fixed decimal).
```

Called By

```
Compiled code.
IBMBCP  - Conversion (bit to fixed or float binary).
IBMBCU  - Conversion (binary constant to float).
IBMBCAC - Conversion director (arithmetic tc character).
IBMBCCL - Conversion director - complex strings (transient)
IBMDCCR - Conversion director - non complex strings (transient).
IBMBSFO - Output conversion director.
IBMBSPO - Cutput conversion director.
IBMDSEE - Edit-directed combination module.
IBMDSEH - Edit-directed combination subset module.
```

IBMBCK - Conversion (Fixed Decimal - Free Decimal - Fixed Decimal)

Function

To perform conversions on the route: fixed decimal - free decimal - fixed decimal. The module has four entry points:

IBMBCKDD: Fixed decimal to fixed decimal.

IBMBCKDP: Fixed decimal to fixed pictured decimal. This module converts to fixed decimal and then branches to entry point IBMBCODP.

IBMBCKZD: Free decimal to fixed decimal.

IBMBCKZP: Free decimal to fixed pictured decimal. This module converts tc fixed decimal and then branches to entry point IBMBCODP.

Error and Exceptional Conditions

SIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source)
        A(source DED)
        A(target)
        A(target DED)
```

Calls

IBMBCO - Conversion (packed decimal tc pictured decimal).

Called By

Compiled code.
IBMBCH - Conversion (fixed binary - float - free decimal).
IBMBCT - Conversion (decimal constant to packed decimal).
IBMBCM - Conversion (pictured decimal to packed decimal).
IBMBCCI - Conversion director - complex strings (transient)
IBMDCCR - Conversion director - non complex strings (transient).
IBMBSPC - Output conversion director.

IBMBCM - Conversion (Pictured Decimal to Packed Decimal) to Packed Decimal)

Function

To convert from pictured decimal to free or fixed decimal. The module has six entry points:

IBMBCMPB: Pictured decimal to bit. This module converts to free decimal and then branches to entry point IBMBCEZB.

IBMBCMPD: Pictured decimal to fixed decimal. This module converts to free decimal and then branches to entry point IBMBCKZD.

IBMBCMPE: Pictured decimal to float pictured decimal. This module converts to free decimal and then branches to entry point IBMBCOZE.

IBMBCMPF: Pictured decimal to float. This module converts to free decimal and then branches to entry point IBMBCEZF.

IBMBCMPP: Pictured decimal to float pictured decimal. This module converts to fixed decimal and then branches to entry point IBMBCKZP.

IBMBCMPX: Pictured decimal to fixed binary. This module converts to free decimal and then branches to IBMBCEZX.

Error and Exceptional Conditions

SIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source)
        A(source DED)
        A(target)
        A(target DED)
```

Calls

```
IBMBCE  - Conversion (fixed decimal - free decimal - float
                       - fixed binary).
IBMBCH  - Conversion (fixed binary - float - free decimal).
IBMBCK  - Conversion (fixed decimal - free decimal - fixed decimal).
IBMBCO  - Conversion (packed decimal to pictured decimal).
```

Called By

```
Compiled code.
IBMBCCL - Conversion director - complex strings (transient).
IBMDCCR - Conversion director - non complex strings (transient).
IBMBSPI - Input conversion director.
IBMBSBC - Output conversion director.
IBMBSPO - Output conversion director.
IBMBSFO - Output conversion director.
IBMDSEE - Edit-directed combination module.
IBMDSEH - Edit-directed combination subset module.
```

IBMBCO - Conversion (Packed Decimal to Pictured Decimal)

Function

To convert from fixed or free decimal to pictured decimal. The module has three entry points:

IBMBCODE: Fixed decimal to float pictured decimal.

IBMBCODP: Fixed decimal to fixed pictured decimal.

IBMBCOZE: Free decimal to float pictured decimal.

Error and Exceptional Conditions

SIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source)
        A(source DED)
        A(target)
        A(target DED)
```

Called By

Compiled code.
IBMBCH - Conversion (fixed binary - float - free decimal).
IBMBCK - Conversion (fixed decimal - free decimal - fixed decimal).
IBMBCM - Conversion (pictured decimal to packed decimal).
IBMBCT - Conversion (decimal constant to packed decimal).
IBMBCGZ - Set a subfield of a complex number to zero.
IBMBCCL - Conversion director - complex strings (transient).
IBMDCCR - Conversion director - non complex strings (transient).
IBMBSPO - Output conversion director.

IBMBCP - Conversion (Bit to Fixed Binary or Float)

Function

To convert from bit to fixed binary or to float. The module has five entry points:

IBMBCPBD: Bit to fixed decimal. This module converts to float binary and then branches to entry point IBMBCHFD.

IBMBCPBE: Bit to float pictured decimal. This module converts to float and then branches to entry point IBMBCHFE.

IBMBCPBF: Bit to float.

IBMBCPBP: Bit to fixed pictured decimal. This module converts to float binary and then branches to entry point IBMBCHFP.

IBMBCPBX: Bit to binary.

Error and Exceptional Conditions

SIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source locator descriptor)
        A(source DED)
        A(target)
        A(target DED)
```

Calls

IBMBCH - Conversion (fixed binary - float - free decimal).

Called By

Compiled code.
IBMBSAI - Input conversion director.
IBMBSPO - Output conversion director.
IBMBSFO - Output conversion director.
IBMBCCL - Conversion director - complex strings (transient).
IBMDCCR - Conversion director - non complex strings (transient).
IBMDSLI - List-directed input.

IBMBCR - Conversion (Fixed Binary or Float to Bit)

Function

To convert fixed binary or float to bit. The module has two entry points:

IBMBCRFB: Float to bit.

IBMBCRXB: Fixed binary to bit.

Linkage

```
R1    = A(PLIST)
PLIST = A(source)
        A(source DED)
        A(target locator descriptor)
        A(target DED)
```

Calls

IBMBCBB - Conversion (bit to bit).

Called By

Compiled code
IBMBCE - Conversion (fixed decimal - free decimal - float - fixed binary).
IBMBCCL - Conversion director - complex strings (transient).
IBMDCCR - Conversion director - non complex strings (transient).
IBMBSBO - Output conversion director.

IBMBCT - Conversion (Decimal Constant to Packed Decimal)

Function

To convert from a decimal constant to fixed or free decimal. The module has seven entry points:

IBMBCTHD: Decimal constant to fixed decimal. This module converts to free decimal and then branches to entry point IBMBCKZD.

IBMBCTHE: Decimal constant to float pictured decimal. This module converts to free decimal and then branches to entry point IBMBCOZE.

IBMBCTHF: Decimal constant to float. This module converts to free decimal and then branches to entry point IBMBCEZF.

IBMBCTHI: Decimal constant to integer.

IBMBCTHP: Decimal constant to fixed pictured decimal. This module converts to free decimal and then branches to entry point IBMBCKZP.

IBMBCTHX: Decimal constant to fixed binary. This module converts to free decimal and then branches to entry point IBMBCEZX.

IBMBCTHZ: Decimal constant to free decimal.

Error and Exceptional Conditions

CONVERSION can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source)
        A(source DED)
        A(target)
        A(target DED)
```

Calls

```
IBMBCE  - Conversion (fixed decimal - free decimal - float
                      - fixed binary).
IBMBCK  - Conversion (fixed decimal - free decimal - fixed decimal).
IBMBCO  - Conversion (packed decimal to pictured decimal).
IBMDSCV - Conversion fix-up bootstrap.
```

Called By

Compiled code.
IBMBCCL - Conversion director - complex strings (transient).
IBMDCCR - Conversion director - non complex strings (transient).
IBMBCCA - Conversion director (character to arithmetic).
IBMBSFI - Input conversion director.
IBMDSEE - Edit-directed combination module.

IBMBCU - Conversion (Binary Constant to Float)

Function

To convert a binary constant to float. The module has five entry points:

IBMBCUID: Binary constant to fixed decimal. This module converts to float and then branches to entry point IBMBCHFD.

IBMBCUIE: Binary constant to float pictured decimal. This module converts to float and then branches to entry point IBMBCHFE.

IBMBCUIF: Binary constant to float.

IBMBCUIX: Binary constant to fixed binary. This module converts to float and then calls entry point IBMBCEFX.

IBMBCUIP: Binary constant to fixed pictured decimal. This module converts to float and then branches to entry point IBMBCHFP.

Error and Exceptional Conditions

CONVERSION and SIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source)
        A(source DED)
        A(target)
        A(target DED)
```

Calls

IBMBCH - Conversion (fixed binary - float - free decimal
IBMBCE - Conversion (fixed decimal - free decimal - float - fixed binary).
IBMDSCV - Conversion fix-up bootstrap.

Called By

Compiled code.
IBMBCCL - Conversion director - complex strings (transient).
IBMDCCR - Conversion director - non complex strings (transient).
IBMBCCA - Conversion director (character to arithmetic).
IBMBSFI - Input conversion director.

IBMBCV - Conversion (Packed Decimal to E-Format)

Function

To convert from fixed or free decimal to E-format. The module has two entry points:

IBMBCVDY: Fixed decimal to E-format.

IBMBCVZY: Free decimal to E-format.

Error and Exceptional Conditions

SIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source)
        A(source DED)
        A(target)
        A(target DED)
```

Called By

Compiled code.
IBMBCH - Conversion (fixed binary - float - free decimal
IBMBCAC - Conversion director (arithmetic to character).
IBMBCCL - Conversion director - complex strings (transient).
IBMDCCR - Conversion director - non complex strings (transient).
IBMBSFO - Output conversion director.
IBMDSEE - Edit-directed combination module.
IBMDSEH - Edit-directed combination subset module.

IBMBCW - Conversion (Packed Decimal to F-format)

Function

To convert from fixed or free decimal to F-format. The module has two entry points:

IBMBCWDH: Fixed decimal to F-format.

IBMBCWZH: Free decimal to F-format.

Error and Exceptional Conditions

SIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(source)
        A(source DED)
        A(target)
        A(target DED)
```

Called By

Compiled code.
IBMBCH - Conversion (fixed binary - float - free decimal).
IBMBCAC - Conversion director (arithmetic to character).
IBMBCCL - Conversion director - complex strings (transient).
IBMDCCR - Conversion director - non complex strings (transient).
IBMBSFO - Output conversion director.
IBMDSEE - Edit-directed combination module.
IBMDSEH - Edit-directed combination subset module.

IBMBCY - Conversion (Fixed Binary to Fixed Binary and Float to Float)

Function

The module has two entry points:

IBMBCYFF: Float to float.

IBMBCYXX: Fixed binary to fixed binary.

Linkage

```
R1    = A(PLIST)
PLIST = A(source)
        A(source DED)
        A(target)
        A(target DED)
```

Called By

Compiled code.
IBMBCCI - Conversion director - complex strings (transient).
IBMDCCR - Conversion director - non complex strings (transient).

CHAPTER 7: COMPUTATIONAL ROUTINES

The computational subroutine package of the resident library can be divided into three main sections:

1. Aggregate handling routines. These all have module names of the form IBMBA**.

2. String handling routines. These all have module names of the form IBMBB**.

3. Arithmetic and mathematical routines. These all have module names of the form IBMBM**.

The routines are described below under these main section headings.

AGGREGATE HANDLING ROUTINES

The aggregate handling modules support the PL/I built-in functions ALL, ANY, STRING, PROD, SUM, and POLY. The modules are listed in figure 7.1.

PL/I Function	Module
ALL, ANY	IBMBAAH
STRING (built-in function)	IBMBANM
STRING (pseudovariable)	IBMBAPM
PROD (fixed integer)	IBMBAPC
PROD (float)	IBMBAPF
SUM (fixed)	IBMBASC
SUM (float)	IBMBASF
POLY (float)	IBMBAYF

Figure 7.1: Aggregate Handling Modules

Two further modules are included in the aggregate handling package: a structure mapping module IBMBAMM and an array indexing module IBMBAIH.

The array indexing module is used by library modules for indexing interleaved arrays. It provides an indexing table for the array which allows the caller to step quickly through the array by the addition of address increments.

MODULE DESCRIPTIONS

IBMBAAH - ALL, ANY (Simple/Interleaved Arrays)

Function

To implement the PL/I built-in functions ALL(x) and ANY(x). The argument (x) of the function is a bit string array. The result is a scalar bit string of length equal to the greatest of the current lengths of the elements of x. The module has three entry points:

IBMBAAHA: ALL(x) and ANY(x), for byte aligned x.

IBMBAAHB: ALL(x) for unaligned x.

IBMBAAHC: ANY(x) for unaligned x.

Method

For byte-aligned string arrays, the AND and OR operations are used for ALL and ANY respectively. For unaligned string arrays, the BOOL function is used with the appropriate parameter bits.

For the first call, the first element of the array serves as both the first and the second argument, thus generating the first element in the target field. For subsequent calls, the target field is the first argument and the next element of the array is the second argument.

Linkage

Entry point IBMBAAHA:

```
R1    = A(PLIST)
PLIST = A(aggregate locator)
        A(halfword number of dimensions)
        A(halfword 0 for ALL, halfword 4 for ANY)
        A(string locator descriptor of target field)
        A(array-type) (non-zero for multidimensional
           interleaved arrays; zero for simple and one-
           dimensional interleaved arrays)
```

Entry points IBMBAAHB and IBMBAAHC

```
R1    = A(PLIST)
PLIST = A(aggregate locator)
        A(halfword number of dimensions)
        A(halfword 8)
        A(string locator descriptor of target field)
        A(array-type) (non-zero for multidimensional
           interleaved arrays; zero for simple and one-
           dimensional interleaved arrays)
```

Calls

IBMBBBA - AND, OR (byte-aligned strings).
IBMBBGB - BOOL.
IBMBAIH - Indexer for interleaved arrays.

Called By

Compiled code.

IBMBAIH - Indexer for Interleaved arrays

Function

To calculate the extent of the nth dimension of an n-dimensional array and to construct a table which is used by the calling routine for indexing the array. The extent of the nth dimension is returned in Register R1; the table is constructed in a work area provided by the caller.

Method

The format of the indexing table constructed by the module is shown in figure 7.2. The table contains three fullwords for each of the dimensions 1 thru (n-1) of the array, where n is the number of dimensions; it is thus 12*(n-1) bytes long.

$SC_{n-1} = E_{n-1}$ (the initial value of the subscript count)
$E_{n-1} = U_{n-1} - L_{n-1} + 1$ (extent)
$INC_{n-1} = M_{n-1} - E_n*M_n$ (increment)
$SC_i = E_i$
$E_i = U_i - L_i + 1$
$INC_i = INC_{i+1} + M_i - E_{i+1}*M_{i+1}$
$SC_1 = E_1$
$E_1 = U_1 - L_1 + 1$
$INC_1 = INC_2 + M_1 - E_2*M_2$

Figure 7.2: Array-Indexing Table

The module constructs the table from the low-address end, using information in the Array Descriptor and in previously completed entries in the table. The three fields for each dimension contain the following information:

<u>Field 1 - Subscript Count (SC)</u>:

The subscript count is a counting field used by the calling routine when indexing the array. It is initially set equal to the extent of the appropriate dimension of the array.

<u>Field 2 - Extent (E)</u>:

The extent field is set equal to

U - L + 1

where U and L are the upper and lower bounds of the appropriate dimension of the array. The extent is thus the number of subscript values that the particular dimension can take.

Field 3 - Increment (INC):

The increment for a particular dimension is used by the calling routine to locate the next element of the array when the subscript value for that dimension changes (and those of the lower dimensions, if any, remain unchanged) during array indexing. The increment for the ith dimension is computed as

$$INC_i = INC_{i+1} + Mi - E_{i+1} * M_{i+1}$$

where Mi is the multiplier for the ith dimension of the array. The value of INC_n is zero.

Linkage

On entry: R1 = A(work area in which the indexing table is to be constructed)
R5 = A(array descriptor)
R6 = A(halfword number of dimensions)

On return: R1 = Extent of nth dimension (E_n)

Called By

IBMBAAH - ALL/ANY.
IBMBASC - SUM (fixed point).
IBMBASF - SUM (floating point).
IBMBAPC - PRCD (fixed point integer).
IBMBAPF - PROD (floating point).
IBMBANM - STRING built-in function.
IBMBAPM - STRING pseudovariable.
IBMBJWI - WAIT (array events).

IBMBAMM - Structure Mapping

Function

The module has four entry points:

IBMBAMMA: To compute the total length of a structure, given a complete Structure Descriptor (SD) and an Aggregate Descriptor Descriptor (ADD).

IBMBAMMB: To map a structure completely using the PL/I mapping algorithm, given an ADD and an SD with only the length and bound fields complete.

IBMBAMMC: To map a structure completely using the COBOL mapping algorithm, given an ADD and an SD with only the length and bound fields complete.

IBMBAMMD: To map a structure up to a given point and subsequently to map it further to another given point. This entry point is used when the REFER option is present in the structure declaration.

Method (chart BAMM)

Entry Point IBMBAMMA:

The offset of the last base element of the structure from the structure origin is found from the Structure Descriptor. If the last base element is an array, the offset of the array's actual origin from its virtual origin is obtained from the Array Descriptor and added to the base element offset previously found. The offset of the last element of the array from the structure origin is then found by adding the multiplier-upperbound products for each dimension.

The length of the last element of the array is obtained, and the length of the structure is finally calculated as the byte offset of the last element from the structure origin, plus the number of bytes occupied by the last element and its bit offset.

Entry Points IBMBAMMB and IBMBAMMC:

A Variable Data Area (VDA) is obtained in which the length accumulator, offset, dimensionality, and SD pointer are stored for each minor structure in PL/I mapping or each dimensioned minor structure in COBOL mapping.

The length of a base element is found (in bits), and the element is aligned to the appropriate boundary.

If the base element is an array, the offset field (AO-VO) of the Array Descriptor in the Structure Descriptor is cleared. If the uninherited dimensionality of the array is not zero, the multipliers for each uninherited dimension are stored in the Array Descriptor.

The Virtual Origin sum is calculated for each of the uninherited dimensions, if any, and the amount by which the padding can be reduced is then calculated and added to the offset. The offset and the actual length of the packed structure are calculated, the element length added to the length accumulator, and the maximum alignment updated if necessary. On the end of a structure for COBOL, each successive containing structure is tested until a minor dimensioned structure or the major structure is found. If the containing structure is minor with dimensions of its own, then the length so far calculated is padded to a multiple of the maximum alignment boundary appearing in it. This pad is stored so that it can be subtracted after the multipliers have been calculated.

If a dimensioned major or minor structure contains more than one base element, the bounds for the dimensions inherited fron that structure are set up by the compiler only for the first base element, and are copied by this module for each of the remaining base elements.

Adjacent structures are then aligned. The information at the top of the stack is unstacked, and the amount of padding which must be placed between the preceding and present structures calculated so that they are both aligned on the correct boundaries. The adjacent structure offset is then added. The offset from the start of the preceding minor structure to the start of the present minor structure is added into the partial offset sum of each base element of the present minor structure.

Entry Point IBMBAMMD:

This entry point uses the same mapping method as that described for IBMBAMMB, but maps the structure only as far as the point specified in the parameter list. The structure may already be partially mapped.

For this reason this entry point uses a VDA which is passed to it by the calling routine.

Error and Exceptional Conditions

The ERROR condition is raised if:

1. The length of the structure is greater than or equal to 2**24 bytes.

2. The value of (actual origin minus virtual origin) for the structure is greater than the one word field reserved for it in the aggregate descriptor descriptor.

Linkage

Entry Points IBMBAMMA, B, and C:

```
R1    = A(PLIST)
PLIST = A(target)
        A(structure descriptor)
        A(aggregate descriptor descriptor)
```

Entry Point IBMBAMMD:

```
R1    = A(PLIST)
PLIST = A(target)
        A(structure descriptor field for the first element to
          be mapped)
        A(aggregate descriptor descriptor)
        A(halfword containing the offset in the aggregate descriptor
          descriptor for the first element to be mapped)
        A(halfword containing the offset in the aggregate descriptor
          descriptor for the last element to be mapped)
        A(VDA)
```

For all entry points, the third parameter on entry contains a fullword binary integer. This integer is equal to (POS-value minus one) for DEFINED structures, and zero for non-DEFINED structures. On exit, the parameter contains the length of the structure and its offset from a doubleword boundary.

Called By

Compiled code

IBMBANM - STRING Built-in Function

Function

To implement the PL/I built-in function STRING. The module has two entry points:

IBMBANMA: Return a fullword containing the string length that would result from the concatenation of all the elements of an argument structure or array.

IBMBANMB: Concatenate all the elements of an argument structure or array into a target field.

Method (chart BANM)

Entry Point IBMBANMA:

The routine consists essentially of finding the current length of each base element and adding it to a length accumulator. Base elements are found by searching the Aggregate Descriptor Descriptor (ADD). If a base element is an array of fixed-length elements, the length is computed as the product of the number of elements and the element length. For one-dimensional varying-string arrays, the current length of each string is found; for multi-dimensional varying-string arrays, use is made of the interleaved array indexing module IBMBAIH, which returns a table used in locating each element of the array. The indexing module is called for simple as well as for interleaved arrays, to avoid using additional code to determine whether an array is simple or interleaved.

Entry Point IBMBANMB:

If the argument is a non-dimensioned structure that does not contain any dimensioned minor structure, the current length and address of each element of every base element is found and the elements are assigned in turn to the target field after a test is made for STRINGSIZE.

For array base elements, the current length and address of each element of the array is found, using module IBMBAIH for multi-dimensioned arrays, before assigning the array elements to the target in row major order.

If dimensioned structures were treated in the same way as non-dimensioned structures, the elements of the dimensioned structure would be assigned to the target as simple arrays in row major order. For example, the structure:

```
 1  A(2),
  2  B(2),
  2  C;
```

would be assigned thus:

B(1,1),B(1,2),B(2,1),B(2,2),C(1),C(2).

To prevent this, each successive minor structure, or in the first instance the major structure itself, is tested for dimensionality. If the structure is dimensioned then the address and current length of each element in the structure is stored in a variable data area(VDA). When the end of the outermost containing structure is reached, these elements are sorted into core address order before being assigned to the target.

Error and Exceptional Conditions

STRINGSIZE can occur in this module.

Linkage

Entry Point IBMBANMA:

```
R1    = A(PLIST)
PLIST = A(aggregate locator)
        A(aggregate descriptor descriptor)
        A(target field)
```

Entry Point IBMBANMB:

```
R1    = A(PLIST)
PLIST = A(aggregate locator)
        A(aggregate descriptor descriptor)
        A(target locator descriptor)
```

Calls

IBMBAIH - Indexer for interleaved arrays.
IBMBBGK - Concatenate (bit strings).
IBMBBGK - General assign (bit strings).
IBMBBGF - Fill (bit strings).

Called By

Compiled code.

IBMBAPC - PROD (Simple or Interleaved Arrays with Fixed Point Integer Elements)

Function

To produce a scalar whose value is the product of all the elements in the array argument. The elements of the array argument may be fixed point binary or decimal, real or complex. Real elements and both the real and the imaginary parts of complex elements must be integers.

Method

The elements of the array are used in row major order to multiply the current product. For multidimensional interleaved arrays the module calls the indexer module IBMBAIH to produce an indexing table for the array.

For fixed point binary integers the product is computed at each step to 63 bits and then reduced to 31 bits by a left shift operation. Each left shift operation incorporates a test for FIXEDOVERFLOW. The result is stored as FIXED (31,0). For fixed point decimal integers the product is calculated throughout as FIXED (31,0) and is finally assigned to the target field as FIXED(15,0). A test is made for FIXEDOVERFLOW.

Error and Exceptional Conditions

FIXEDOVERFLOW can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(aggregate locator)
        A(halfword number of dimensions)
        A(data element descriptor of array)
        A(target)
        A(array-type) (non-zero for multidimensional
          interleaved arrays; zero for simple and one-
          dimensional interleaved arrays)
```

Calls

IBMBAIH - Indexer for interleaved arrays.

Called By

Compiled code.

IBMBAPF - PROD (Simple or Interleaved Arrays with Floating Point Elements)

Function

To produce a scalar whose value is the product of all the elements in the array argument. The elements of the array argument may be short or long floating point, real or complex. The module has four entry points:

IBMBAPFA: Short float real elements

IBMBAPFB: Short float complex elements

IBMBAPFC: Long float real elements

IBMBAPFD: Long float complex elements

Method

The elements of the array are used in row major order to multiply the current product. For multidimensional interleaved arrays the module calls the indexer module IBMBAIH to produce an indexing table for the array.

Error and Exceptional Conditions

OVERFLOW or UNDERFLOW can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(aggregate locator)
        A(halfword number of dimensions)
        A(target)
        A(array-type) (non-zero for multidimensional
          interleaved arrays; zero for simple and one-
          dimensional interleaved arrays)
```

Calls

IBMBAIH - Indexer for interleaved arrays.

Called By

Compiled code.

IBMBAPM - STRING Pseudovariable

Function

To assign a bit or character string to the elements of an array or structure variable.

Method (chart BAPM)

The pseudovariable must have as argument a structure variable composed entirely of:

1. bit strings and/or binary numeric pictured data.

or

2. character strings and/or decimal numeric pictured data.

The total number of elements of all base elements of the structure is found and a variable data area (VDA) is obtained of length 8 bytes for each element. Each 8-byte field is made into a string locator/descriptor for an element as the allocated length and address of each element is determined. For multi-dimensioned array base elements the interleaved array indexing module IBMBAIH is used to locate each element. The indexing module is called for simple as well as for interleaved arrays to avoid using additional code to determine whether the array is simple or interleaved.

Since each base element of a dimensioned structure is an interleaved array, and SLDs (string locator descriptors) are set up in the VDA for all elements of one interleaved array at a time, the order in which the SLDs are set up does not necessarily correspond to the order in which the elements are encountered in the structure. For example, elements of the structure:

```
 1 A(2),
   2 B(2),
   2 C   ;
```

appear in storage in the order:

B(1,1),B(1,2),C(1),B(2,1),B(2,2),C(2)

but the SLDs are set up in the VDA in the order:

B(1,1),B(1,2),B(2,1),B(2,2),C(1),C(2)

Therefore after the last entry in the VDA is completed, a simple exchange sort is carried out to arrange the SLDs in the order of the core addresses of the corresponding elements. (For non-dimensioned structures the SLDs will already be in the required order). The source string is then assigned to the elements in core address order and any padding which may be necessary is performed. A check for STRINGSIZE is made when assigning to the last element of the structure.

Error and Exceptional Conditions

STRINGSIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(structure descriptor)
        A(Aggregate descriptor descriptor)
        A(string locator descriptor of source string)
```

Calls

IBMBAIH - Indexer for interleaved arrays.
IBMBCBB - Conversion (bit to bit).

Called By

Compiled code.

IBMBASC - SUM (Simple or Interleaved Arrays with Fixed Point Elements)

Function

To produce a scalar whose value is the sum of all the elements in the array argument. The elements of the array argument may be fixed point binary or decimal, real or complex.

Method

The elements of the array are added to the current sum in row major order. For multidimensional interleaved arrays the module calls the indexer module IBMBAIH to produce an indexing table for the array.

For fixed point binary elements of precision (p,q), the sum is both accumulated and stored to precision (31,q). For decimal elements, however, the sum is accumulated to precision (31,q) but stored to precision (15,q); a test is therefore made for FIXEDOVERFLOW.

Error and Exceptional Conditions

FIXEDOVERFLOW can occur in this module

Linkage

```
R1    = A(PLIST)
PLIST = A(aggregate locator)
        A(halfword number of dimensions)
        A(data element descriptor of array)
```

```
        A(target)
        A(array-type) (non-zero for multidimensional
          interleaved arrays; zero for simple and one-
          dimensional interleaved arrays)
```

Calls

IBMBAIH - Indexer for interleaved arrays.

Called By

Compiled code.

IBMBASF - SUM (Simple or Interleaved Arrays with Floating Point Elements)

Function

To produce a scalar whose value is the sum of all the elements in the array argument. The elements of the array argument may be short or long floating point, real or complex. The module has four entry points:

IBMBASFA: Short float real elements

IBMBASFB: Short float complex elements

IBMBASFC: Long float real elements

IBMBASFD: Long float complex elements

Method

The elements of the array are added to the current sum in row major order. For multidimensional interleaved arrays the module calls the indexer module IBMBAIH to produce an indexing table for the array.

Error and Exceptional Conditions

OVERFLOW or UNDERFLOW can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(aggregate locator)
        A(halfword number of dimensions)
        A(data element descriptor of array)
        A(target)
        A(array-type) (non-zero for multidimensional
          interleaved arrays; zero for simple and
          one-dimensional interleaved arrays)
```

Calls

IBMBAIH - Indexer for interleaved arrays.

Called By

Compiled code.

IBMBAYF - POLY Built-In Function

Function

To implement the PL/I built-in function POLY(A,X). The module has four entry points:

IBMBAYFA: Short float real arguments.

IBMBAYFB: Short float complex arguments.

IBMBAYFC: Long float real arguments.

IBMBAYFD: Long float complex arguments.

The function is defined as follows:

Let A and X be vectors, that is, one-dimensional arrays, declared as A(m:n) and X(p:q). Then:

$$\mathrm{POLY}(A,X) = A(m) + \sum_{j=1}^{(n-m)} A(m+j) * \prod_{i=0}^{(j-1)} X(p+i)$$

unless n=m, when the result is A(m).

If (q-p)<(n-m-1) then for (p+i)>q ,

X(p+i) = X(q)

If X is scalar it is treated as the vector X(1). The function is then computed as:

$$\mathrm{POLY}(A,X) = \sum_{j=0}^{(n-m)} A(m+j) * X ** j$$

Method

Case 1 - Vector X, (q-p) ≥ (n-m-1):

POLY(A,X) is evaluated by nested multiplication and addition, i.e.

(...(A(n)*X(k) + A(n-1))*X(k-1) + A(n-2))*....+A(m+1))*X(p) +A(m)

where k = p+n-m-1

Case 2 - Vector X, (q-p)<(n-m-1):

In the expression given in Case 1 above, the terms in X with subscripts ranging from k down to q are all made equal to X(q).

Case 3 - Scalar X:

In the expression given in Case 1, all terms in X are made equal to X.

Error and Exceptional Conditions

OVERFLOW or UNDERFLOW can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(aggregate locator for A)
        A(X) (zero for vector X)
        A(aggregate locator for X) (zero for scalar X)
        A(target)
```

Called By

Compiled code

STRING HANDLING ROUTINES

The string handling modules of the resident library are listed in figure 7.3.

PL/I Operations	PL/I Functions	Character Strings	Bit Strings Byte-aligned	Bit Strings General
AND, OR		-	IBMBBBA	-
NOT		-	IBMBBBN	-
Compare		-	-	IBMBBGC
Concatenate	REPEAT	IBMBBCK	-	IBMBBGK
	INDEX	IBMBBCI	-	IBMBBGI
	TRANSLATE	IBMBBCT	-	-
	VERIFY	IBMBBCV	-	IBMBBGV
	BCOL	-	-	IBMBBGB
	SUBSTR	IBMBBGS	-	IBMBBGS
Fill, Assign		-	IBMBBGF	-

Figure 7.3: String Handling Modules

IBMBBBA - AND and OR Operations (Byte-aligned Bit Strings)

Function

To implement the logical AND and OR operations between two byte-aligned bit strings. The module has two entry points:

IBMBBBAA: AND operation.

IBMBBBAB: OR operation.

Method

Entry Point IBMBBBAA: The current length of a varying target is set to the length of the longer operand, or to the maximum length of the target field if this is smaller. The two strings are then ANDed together for the length of the shorter operand. The remainder of the target field is filled with zeros.

Entry Point IBMBBBAB: The current length of a varying target is set to the length of the longer operand, or to the maximum length of the target field if this is smaller. The two strings are then ORed together for the length of the shorter operand. The remainder of the longer string is moved unchanged to the target field.

Error and Exceptional Conditions

STRINGSIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of first operand)
        A(string locator descriptor of second operand)
        A(string locator descriptor of target)
```

Called By

Compiled code.
IBMBAAH - ALL/ANY.

IBMBBBN - NOT operation (Byte-aligned Bit Strings)

Function

To implement the NOT operation on a byte-aligned bit string and place the result in a byte-aligned target field.

Method

The current length of a varying target is set to the current length of the operand, or to the maximum length of the target field if this is smaller. The current length of the target field is then set to a string of ones, and the result is obtained by an EXCLUSIVE-OR operation with the operand.

Error and Exceptional Conditions

STRINGSIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of operand)
        A(string locator descriptor of target)
```

Calls

IBMBBGFC - Fill (bit strings)

Called By

Compiled code

IBMBBCI - INDEX (Character Strings)

Function

To implement the PL/I built-in function INDEX(bit string, configuration string)

Method

The index is found by shifting and comparing portions of the two strings in even-odd pairs of registers. The index is stored in the target field as a binary integer of default precision.

Linkage

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of source string)
        A(string locator descriptor of configuration string)
        A(halfword target)
```

Called By

Compiled code.

IBMBBCK - Concatenate, REPEAT (Character Strings)

Function

To concatenate character strings. The module has two entry points:

IBMBBCKA: To concatenate two character strings into a target field.

IBMBBCKB: To implement the PL/I built-in function REPEAT, that is, to concatenate a single character string with itself n times and to form the resulting n+1 instances of the string in a target field.

Method

Both entry points of the module use an internal subroutine which obtains data from a source, aligns it correctly and moves it to the target field.

Entry Point IBMBBCKA:

The current length of a varying target is made equal to the sum of the lengths of the two source strings, or to the maximum length of the target field if this is smaller. The internal subroutine is then used twice to move the source strings to the target field.

Entry Point IBMBBCKB:

The current length of a varying target is made equal to (n+1) times the length of the source string, or to the maximum length of the target field if this is smaller. The internal subroutine is then used repeatedly to assemble the concatenated string in the target field. To reduce the number of times the subroutine is used, the target field is concatenated with itself whenever possible.

Error and Exceptional Conditions

STRINGSIZE can occur in this module.

Linkage

Entry Point IBMBBCKA:

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of first string)
        A(string locator descriptor of second string)
        A(string locator descriptor of target)
```

Entry Point IBMBBCKB:

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of source string)
        A(n)
        A(string locator descriptor of target)
```

Called By

Compiled code.

IBMBBCT - TRANSLATE (Character String)

Function

To implement the PL/I built-in function TRANSLATE for character string arguments.

Method

The module uses a translate table consisting of the 256 elements of the EBCDIC code arranged in ascending order and modified in such a way that any character appearing in the position string is replaced in the table by the corresponding replacement string character. The translate table may either be passed to the the routine by compiled code or constructed by the module itself. The necessary substitution of characters in the source string is then accomplished using the Translate (TR) instruction. Finally, the translated string is assigned to the target.

Error and Exceptional Conditions

The STRINGSIZE condition can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of source string)
        A(string locator descriptor of replacement string) or
           zero if the translate table has been built
        A(string locator descriptor of position string) or
           zero if not specified
        A(target string locator)
        A(Translate table)  or zero if both 2nd 3rd arguments
          are non-zero
```

Called By

Compiled code

IBMBBCV - VERIFY (Character Strings)

Function

To implement the PL/I built-in function VERIFY for character string arguments.

Method

The address of a translate table consisting of the 256 elements of the EBCDIC code arranged in ascending binary order is passed to the module by compiled code. The module modifies the table by replacing any character that appears in the verification table by zero and setting the remaining elements of the table to X'FF'. The Translate and Test instruction TRT is then used to check the characters of the source string from left to right. If a non-zero byte is referenced in the source string, its position is returned as a binary integer of default precision. Otherwise the result field is set to zero.

Linkage

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of source string)
        A(string locator descriptor of verification string)
        A(translate table)
        A(halfword result field)
```

Called By

Compiled code.

IBMBBGB - BOOL (Bit Strings)

Function

To take two source strings and perform one of the sixteen possible logical operations between corresponding bits. The particular operation performed is defined by inserting the bit pattern n1,n2,n3,n4 yielded by the third argument into the following table.

First field	0	0	1	1
Second field	0	1	0	1
Target field	n1	n2	n3	n4

Method

The current length of the target string is set equal either to the maximum of the current lengths of the source strings or to the maximum length of the target field if this is smaller. The specified operation is then performed on the strings and the result is stored in the target field. If one string is shorter than the other it is regarded as being extended on the right with zeros up to the length of the longer.

Error and Exceptional Conditions

STRINGSIZE can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of first string)
        A(string locator descriptor of second string)
        A(string locator descriptor of target)
        A(halfword containing n1,n2,n3,n4 in the last four bits)
```

Calls

IBMBBGFC - Fill (Byte-aligned strings)

Called by

Compiled code.
IBMBAAH - ALL/ANY.

IBMBBGC - Compare (General Bit Strings)

Function

To compare two bit strings and to return a condition code as bits 2 and 3 of a fullword target as follows:

00 if the two strings are equal.

01 if the first string compares low (i.e. has a '0'B) at the first inequality.

10 if the first string compares high (i.e. has a '1'B) at the first inequality.

The shorter string is treated as though extended with zeros to the length of the longer.

Method

Corresponding portions of the two strings, up to 32 bits long, are aligned in even-odd register pairs and then compared using the CLR instruction.

Linkage

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of first operand)
        A(string locator descriptor of second operand)
        A(target)
```

Called By

Compiled code.

IBMBBGF - Assign (Byte-Aligned Bit Strings) and Fill (General Bit Strings)

Function

To assign a fixed or varying byte-aligned bit string to a fixed or varying byte-aligned bit string target, and to fill a general bit string to its maximum length with zeros. The module has three entry points:

IBMBBGFA: Assign a byte-aligned bit string to a byte-aligned target, filling out with zero bits if necessary.

IBMBBGFB: Fill out a byte-aligned bit string with zero bits, for a specified length. This entry point is used only by compiled code.

IBMBBGFC: Fill out a byte-aligned bit string with zero bits, for a specified length. This entry point is used only by other library modules.

Error and Exceptional Conditions

STRINGSIZE can occur in this module.

Linkage

Entry Point IBMBBGFA:

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of source string)
        A(string locator descriptor of target string)
```

Entry Point IBMBBGFB:

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of target to be filled)
```

Entry Point IBMBBGFC:

R1 = Complement of number of bits in the bit overlap.
R6 = Length to be filled.
R8 = Byte origin of target to be filled.

Called By

Compiled code.
IBMBANM - STRING built-in function.
IBMBBGK - Concatenate (general bit strings).
IBMBBGB - BOOL (general bit strings).

IBMBBGI - INDEX (Bit Strings)

Function

To implement the PL/I built-in function INDEX(bit string, configuration string)

Method

The index is found by shifting and comparing portions of the two strings in even-odd pairs of registers. The index is stored in the target field as a binary integer of default precision.

Linkage

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of source string)
        A(string locator descriptor of configuration string)
        A(halfword target)
```

Called By

Compiled code.

IBMBBGK - Concatenate, REPEAT, and General Assign (Bit Strings)

Function

To concatenate bit strings and to assign them to a target. The module has three entry points:

IBMBBGKA: To concatenate two bit strings into a target field.

IBMBBGKA: To implement the PL/I built-in function REPEAT, that is, to concatenate a single bit string with itself n times and to form the resulting n+1 instances of the string in a target field.

IBMBBGKC: To assign a bit string to a target field.

Method

All three entry points of the module use an internal subroutine which obtains data from a source, aligns it correctly and moves it to the target field.

Entry Point IBMBBGKA:

The current length of a varying target is made equal to the sum of the lengths of the two source strings, or to the maximum length of the the target field if this is smaller. The internal subroutine is then used twice to move the source strings to the target field.

Entry Point IBMBBGKB:

The current length of a varying target is made equal to (n+1) times the length of the source string, or to the maximum length of the target field if this is smaller. The internal subroutine is then used repeatedly to assemble the concatenated string in the target field. To reduce the number of times the subroutine is used, the target field is concatenated with itself whenever possible.

Entry Point IBMBBGKC:

The current length of a varying target is made equal to the current length of the source string. The internal subroutine is then used once to move the contents of the source string to the target field.

Error and Exceptional Conditions

STRINGSIZE can occur in this module.

Linkage

Entry Point IBMBBGKA:

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of first string)
        A(string locator descriptor of second string)
        A(string locator descriptor of target)
```

Entry Point IBMBBGKB:

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of source string)
        A(n)
        A(string locator descriptor of target)
```

Entry Point IBMBBGKC:

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of source string)
        A(string locator descriptor of target)
```

Called By

Compiled code.
IBMBNMA - STRING built-in function.

IBMBBGS - SUBSTR SLD

Function

To produce a string locator descriptor (SLD) describing the substring specified by the SUBSTR built-in function or pseudovariable. The module has four entry points:

IBMBBGSA: SUBSTR(bit string, i)

IBMBBGSB: SUBSTR(character string, i)

IBMBBGSC: SUBSTR(bit string, i, j)

IBMBBGSD: SUBSTR(character string,i,j)

Method

Arithmetic is performed according to the function definition, using the current length of the argument string. The result described by the SLD is a fixed-length string.

Error and Exceptional Conditions

STRINGRANGE can occur in this module.

Linkage

Entry Points IBMBBGSA and B:

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of source string)
        A(i)
        A(field for target SLD)
```

Entry Points IBMBBGSC and D:

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of source string)
        A(i)
        A(j)
        A(field for target SLD)
```

Called By

Compiled code.

IBMBBGV - VERIFY (Bit Strings)

Function

To implement the PL/I built-in function VERIFY for bit string arguments.

Method

The verification string is first checked to see whether it contains only '1'Bs, only '0'Bs, or both. If the verification string contains both '0'Bs and '1'Bs, a zero is returned in the result field. If the verification string contains only '0'Bs or only '1'Bs, the source string is searched bit-by-bit for an offending bit. If such a bit is found, its position is returned in the result field; otherwise a zero is returned.

Linkage

```
R1    = A(PLIST)
PLIST = A(string locator descriptor of source string)
        A(string locator descriptor of verification string)
        A(halfword result field)
```

Called By

Compiled code.

ARITHMETIC AND MATHEMATICAL ROUTINES

The modules described in this section support the PL/I built-in arithmetic and mathematical functions and also provide support for several arithmetic operations that are too complicated or time-consuming to be done inline.

The arithmetic operations supported by the library are shown in figure 7.4. Routines are provided to handle exponentiation, complex multiplication and division, and the PL/I built-in functions ADD, MULTIPLY, DIVIDE, and ABS. It should be noted that these operations are not necessarily always handled by a call to the library; they may be done inline.

The mathematical built-in functions supported by the library are shown in figure 7.5.

Operation or Function	Real Operands Fixed binary	Fixed decimal	Short float	Long float
Integer exponentiation: x**n	-	-	IBMBMXS	IBMBMXL
General exponentiation: x**y	-	-	IBMBMYS	IBMBMYL
Shift-and-assign, shift-and-load	-	IBMBMUD	-	-
ADD	-	IBMBMOD	-	-

	Complex Operands Fixed binary	Fixed decimal	Short float	Long float
Integer exponentiation: z**n	-	-	IBMBMXW	IBMBMXW
General exponentiation: z1**z2	-	-	IBMBMYX	IBMBMYY
Division: z1/z2	-	-	IBMBMWX	IBMBMWY
Multiplication: z1*z2	-	-	IBMBMVW	IBMBMVW
Multiplication/division: z1*z2; z1/z2	IBMBMVU	IBMBMVV	-	-
ADD	-	IBMBMOD	-	-
MULTIPLY	IBMBMPU	IBMBMPV	-	-
DIVIDE	IBMBMQU	IBMBMQV	-	-
ABS	IBMBMRU	IBMBMRV	IBMBMRX	IBMBMRY

Figure 7.4: Arithmetic Operations

Real Arguments

Function	Short float	Long float
SQRT	IBMBMAS	IBMBMAL
EXP	IBMBMBS	IBMBMBL
ERF, ERFC	IBMBMCS	IBMBMCL
LOG, LOG2, LOG10	IBMBMDS	IBMBMDL
SIN, SIND, COS, COSD	IBMBMGS	IBMBMGL
TAN, TAND	IBMBMHS	IBMBMHL
SINH, COSH	IBMBMIS	IBMBMIL
TANH	IBMBMJS	IBMBMJL
ATAN, ATAND	IBMBMKS	IBMBMKL
ATANH	IBMBMLS	IBMBMLL
ASIN, ACOS	IBMBMMS	IBMBMML

Complex Arguments

Function	Short float	Long float
SQRT	IBMBMAX	IBMBMAY
EXP	IBMBMBX	IBMBMBY
LOG	IBMBMDX	IBMBMDY
SIN, SINH, COS, COSH	IBMBMGX	IBMBMGY
TAN, TANH	IBMBMHX	IBMBMHY
ATAN, ATANH	IBMBMKX	IBMBMKY

Figure 7.5: Mathematical Functions

MODULE DESCRIPTIONS

IBMBMAL - SQRT (Long Float Real)

Function

To calculate the square root of x.

Method

If x = 0, SQRT(x) = 0. Otherwise, let x = 16**(2*p - q)*f, where p is an integer, q = 0 or 1, and 1/16 ≤ f < 1. Then

SQRT(x) = 16**p*4**-q*SQRT(f)

The first approximation of SQRT(x) is computed as:

y_0= 16**p*4**(1-q)*0.2202(f+0.2587)

This approximation was chosen in order to permit the use of single precision instructions in the final iteration by making the quantity x/y_3-y_3 less than 16**(p-8).

Four Newton-Raphson iterations of the form $y_{n+1} = (y_n + x/y_n)/2$ are then applied, two in short precision and two in long precision, the last being computed as

$SQRT(x) = y_3 + (x/y_3 - y_3)/2$

with an appropriate truncation maneuver to obtain virtual rounding.

The maximum relative error of the final result is theoretically 2**-63.23.

Error and Exceptional Conditions

The ERROR condition is raised if x is negative.

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Called By

Compiled code.
IBMBMML - ASIN (long float real).
IBMBMRY - ABS (long float complex).
IBMBMAY - SQRT (long float complex)

IBMBMAS - SQRT (Short Float Real)

Function

To calculate the square root of x.

Method

If x = 0, then SQRT(x) = 0.

Otherwise, let

$x = 16^{**}(2^*p - q)^*f$

where p is an integer, q = 0 or 1, and $1/16 \le f < 1$.

Then

$SQRT(x) = 16^{**}p^*4^{**}-q^*SQRT(f)$

The first approximation of SQRT(x) is obtained by the hyperbolic fit

$y_0 = 16^{**}p^*4^{**}-q(1.681595 - 1.288973/(0.8408065 + f))$

This approximation attains the minimax relative error. The maximum relative error is 2**-5.748.

The Newton-Raphson iteration

$y_{n+1} = (y_n + x/y_n)/2$

is applied twice, the second iteration being performed as

$y_2 = (y_1 - x/y_1)/2 + x/y_1$

with partial rounding.

The maximum relative error of y_2 is theoretically 2**-25.9

Error and Exceptional Conditions

The ERROR condition is raised if x is negative.

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Called By

Compiled code.
IBMBMMS - ASIN, ACOS (short float real).
IBMBMRX - ABS (short float complex).
IBMBMAX - SQRT (short float complex).

IBMBMAX, IBMBMAY - SQRT (Float Complex)

Modules

Argument	Module
short float	IBMBMAX
long float	IBMBMAY

Function

To calculate the principal value of the square root of z, where z = x + yI. The principal value is that result satisfying the condition -pi/2 < argument of result ≤ pi/2.

Method

1. Let SQRT(x+yI) = a+bI.

2. Let SQRT((ABS(x) + ABS(x+yI))/2) = k*SQRT(w_1+w_2) = c

   v_1 = MAX(ABS(x),ABS(y)) and

   v_2 = MIN(ABS(x),ABS(y))

3. In the special case when either v_2 = 0 or v_1 >> v_2 let w_1 = v_2 and w_2 = v_1

   Let k = 1 if v_1 = ABS(x) or k = 1/SQRT(2) if v_1 = ABS(y)

4. In the general case compute:

$F = SQRT(1/4+(1/4)*(v_1/v_2)**2)$

If ABS(x) is near the underflow threshold, then take

$w_1 = ABS(x)$, $w_2 = v_1*2*F$, and $k = 1/SQRT(2)$

If v_1*F is near the overflow threshold, then take

$w_1 = ABS(x)/4$, $w_2 = v_1*F/2$, and $k = SQRT(2)$

In all other cases take

$w_1 = ABS(x)/2$, $w_2 = v_1*F$, and $k = 1$

5. Compute c. The appropriate real square root routine is called to evaluate $SQRT(w_1 + w_2)$; see "Calls" below.

6. If c = 0 then a = b = 0.

If $c \neq 0$ and $x \geq 0$, then a=c and b=y/(2*c).

If $c \neq 0$ and $x < 0$, then a=ABS(y/(2*c)) and b=SIGN(y)*c.

Linkage

```
R1    = A(PLIST)
PLIST = A(z)
        A(target)
```

Calls

IBMBMAS - SQRT (short float real).
IBMBMAL - SQRT (long float real).

Called By

Compiled code.

IBMBMBL - EXP (Long Float Real)

Function

To calculate e to the power of x.

Method

If x < -180.2187, return zero result. Otherwise EXP(x) is calculated as follows:

1. Divide x by LOG(2) and write

$x = (4*a-b-c/16)*LOG(2)-d$

where a, b and c are integers, $0 \le b \le 3$, $0 \le c \le 15$, and the remainder d is in the range $0 \le d <$ LOG(2)/16. This reduction is carried out in extra precision. Then EXP(x) = 16**a*2**-b*2**(-c/16)*EXP(-d).

2. Compute EXP(-d) by using a minimax polynomial approximation of degree 6 over the range $0 \le d <$ LOG(2)/16. The coefficients of this approximation were obtained by taking the minimax of relative errors under the constraint that the constant term shall be exactly one. The relative error is less than -2**56.87.

3. Multiply EXP(-d) by 2**(-c/16), then halve the result b times.

4. Finally, add the hexadecimal exponent a to the characteristic of the answer.

Error and Exceptional Conditions

The OVERFLOW condition occurs if x > 174.673.

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Called By

Compiled code.

IBMBMYL - General Exponentiation (long float real).
IBMBMCL - ERF,ERFC (long float real).
IBMBMBY - EXP (long float complex).
IBMBMIL - SINH,COSH (long float real).
IBMBMGY - SIN,SINH,COS,COSH (long float complex).
IBMBMJL - TANH (long float real).

IBMBMBS - EXP (Short Float Real)

Function

To calculate e to the power of x.

Method

If x < -180.2187, return zero result. Otherwise EXP(x) is calculated as follows:

1. Divide x by LOG(2) and write

   y = x/LOG(2) = 4*a-b-d

   where a and b are integers, $0 \le b \le 3$, and $0 \le d \le 1$.

Then EXP(x) = 2**y = 16**a*2**-b*2**-d

2. Compute 2**-d by the following fractional approximation

```
2**-d =  1-2*d/(0.024657359*d**d+d+9.9545946-617.97227/
                              (d**2+87.417497))
```

This formula can be obtained by the transformation of the Gaussian continued fraction

```
EXP(-z) = 1-z/(1+z/(2-z/(3+z/(2-z/(5+z/(2-z/(7+z/2)))))))
```

The maximum relative error of this approximation is 2**-29.

3. Multiply 2**-d by 2**-b

4. Finally, add the hexadecimal exponent a to the characteristic of the answer.

Error and Exceptional Conditions

The OVERFLOW condition can occur if x >174.673.

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Called By

Compiled code.
IBMBMYS - General Exponentiation (short float real).
IBMBMCS - ERF,ERFC (short float real).
IBMBMBX - EXP (short float complex).
IBMBMIS - SINH,COSH (short float real).
IBMBMGX - SIN,SINH,COS,COSH (short float complex).
IBMBMJS - TANH (short float real).

IBMBMBX, IBMBMBY - EXP (Float Complex)

Modules

Argument	Module
Short float	IBMBMBX
Long float	IBMBMBY

Function

To calculate e to the power z.

Method

Let z = x + yI.

Then REAL(EXP(z))=EXP(x)*COS(y)

and IMAG(EXP(z))=EXP(x)*SIN(y)

Error and Exceptional Conditions

The ERROR condition is raised in the called real SIN routine IBMBMGS (short) or IBMBMGL (long) if:

```
  ABS(y) ≥  2**18*pi  -  short
            2**50*pi  -  long
```

The OVERFLOW condition can occur in the real EXP routine IBMBMBS (short) or IBMBMBL (long).

Linkage

```
R1    = A(PLIST)
PLIST = A(z)
        A(target)
```

Calls

IBMBMBX calls:

IBMBMBS - EXP (short float real).
IBMBMGS - SIN, COS (short float real).

IBMBMBY calls:

IBMBMBL - EXP (long float real).
IBMBMGL - SIN, COS (long float real).

Called By

Compiled code.

Routines IBMBMBX and IBMBMBY are called by the complex general exponentiation routines IBMBMYX and IBMBMYY respectively.

<u>IBMBMCL - ERF, ERFC (Long Float Real)</u>

Function

To calculate the error function of x or the complement of this function. The module has two entry points:

<u>IBMBMCLA</u>: ERF

<u>IBMBMCLB</u>: ERFC

Method

Case 1: $0 \le x < 1$

Compute ERF(x) by the following approximation:

ERF(x) = x*(a_0+a_1*x**2+a_2*x**4+....+a_{11}*x**22)

The coefficients were obtained by the minimax approximation in relative error of ERF(x)/x as a function of x**2 over the range 0 ≤ x**2 ≤ 1. The relative error of this approximation is less than 2**-56.9.

ERFC(x) = 1-ERF(x)

Case 2: $1 \le x < 2.040452$

Compute ERFC(x) by the following approximation:

ERFC(x) = b +b_1*z+b_2*z**2+....+b_{18}*z**18

where z = x-T and T = 1.709472. The coefficients were obtained by the minimax approximation in absolute error of the function f(z) = ERFC(z+T) over the range -0.709472 ≤ z ≤ 0.330948. The absolute error of this approximation is less than 2**-60.3. The limits of this range and the value of the origin T were chosen to minimize the hexadecimal rounding errors.

ERF(x) = 1-ERFC(x),   1/256 ≤ ERFC(x) ≤ 0.1573

Case 3: $2.040452 \le x < 13.306$

Compute ERFC(x) by the following approximation:

ERFC(x) = EXP(-z)*F/x

where z = x**2

and   F = c +d_1/(z+c_1)+d_2/(z+c_2)+....+d_7/(z+c_7)

The coefficients of F were obtained by transforming a minimax rational approximation of the function f(w) = ERFC(x)*x*EXP(x**2) in absolute error over the range 13.306**-2 ≤ w ≤ 2.040452**-2 of the form

f(w) = (a_0+a_1*w+a_2*w**2+....+a_7*w**7)/
          (b_0+b_1*w+b_2*w**2+....+b_6*w**6+w**7)

where w = x**-2. The absolute error of this approximation is less than 2**-57.9.

If 2.040452 ≤ x < 6.092368 then ERF(x) = 1-ERFC(x)

If 13.306 > x ≥ 6.092368 then ERF(x) = 1

Case 4: $x \ge 13.306$

Results 1 and 0 are returned for ERF(x) and ERFC(x) respectively.

Case 5: $x < 0$

ERF(x) = -ERF(-x)

and ERFC(x) = 2-ERFC(-x).

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Calls

IBMBMBL - EXP (long float real).

Called By

Compiled code.

IBMBMCS - ERF, ERFC (Short Float Real)

Function

To calculate the error function of x or the complement of this function. The module has two entry points:

IBMBMCSA: ERF

IBMBMCSB: ERFC

Method

Case 1: $0 \le x \le 1$

Compute ERF(x) by the following approximation:

ERF(x) = x*(a_0+a_1*x**2+a_2*x**4+.....+a_5*x**10)

The coefficients were obtained by the minimax approximation in relative error of ERF(x)/x as a function of x**2 over the range 0 ≤ x**2 ≤ 1. The relative error of this approximation is less than 2**-24.6.

ERFC(x) = 1-ERF(x)

Case 2: $1 < x < 2.040452$

Compute ERFC(x) by the following approximation:

ERFC(x) = b_0+b_1*z+b_2*z**2+.....+b_9*z**9

where z = x-T and T = 1.709472. The coefficients were obtained by the minimax approximation in absolute error of the function f(z) = ERFC(z+T) over the range, -0.709472 ≤ z ≤ 0.33098. The absolute error of this approximation is less than 2**-31.5. The limits of this range and the value of the origin T were chosen to minimize the hexadecimal rounding errors.

ERF(x) = 1-ERFC(x),   1/256 < ERFC(x) ≤ 0.1573

Case 3: $2.040452 \le x < 13.306$

Compute ERFC(x) by the following approximation:

ERFC(x) = EXP(z)*F/x

where z = x**2

and f = c_0 +(c_1+c_2*z+c_3*z**2)/(d_1*z+d_2*z**2+z**3)

The coefficients of F were obtained by transforming a minimax rational approximation of the function f(w) = ERFC(x)*x*EXP(x**2) in absolute error over the range 13.306**-2 ≤ w ≤ 2.040452**-2 of the form

f(w) = (a_0 +a_1*w+a_2*w**2+a_3*w**3)/(b_0+b_1*w+w**2)

where w = x**2. The absolute error of this approximation is less than 2**-26.1.

If 2.040452 ≤ x < 3.919206, ERF(x) = 1 - ERFC(x)

If 13.306 > x ≥ 3.919206, ERF(x) = 1

Case 4: x ≥ 13.306

Results 1 and 0 are returned for ERF(x) and ERFC(x) respectively.

Case 5: x < 0

ERF(x) = - ERF(-x)

and ERFC(x) = 2 - ERFC(-x).

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Calls

IBMBMBS - EXP (short float real).

Called By

Compiled code.

IBMBMDL - LOG, LOG2, LOG10 (Long Float Real)

Function

To calculate the logarithm of x to the base e, 2, or 10. The module has three entry points:

IBMBMDLA: LOG

IBMBMDLB: LOG2

IBMBMDLC: LOG10

Method

Let x = 16**p*2**(-q)*m where p is the exponent, q is an integer such that 0 ≤ q ≤ 3, and 1/2 ≤ m < 1.

Two constants, a (= base point) and b (=-LOG2(a)), are defined as follows:

1/2 ≤ m ≤ 1/SQRT(2): a = 1/2, b = 1

1/SQRT(2) ≤ m < 1: a = 1, b = 0

Let y = (m - a)/(m + a).

Then m = a*(1 + y)/(1 - y) and ABS(y) < 0.1716.

Now x = 2**(4*p - q - b)*(1 + y)/(1 - y). Therefore:

LOG(x) = (4*p - q - b)*LOG(2) + LOG((1+y)/(1-y))

To obtain LOG((1+y)/(1-y)), w = 2*y = (m-a)/(0.5m+0.5a) is computed and the following approximation is then performed:

LOG((1+y)/(1-y)) = w*(c + c_1*w**2*(w**2 +c_2+c_3/(w**2+c_4+c_5/(w**2+c_6))))

The coefficients were obtained by the minimax rational approximation of LOG((1+y)/(1-y))/(2*y), in relative error, over the range y**2 ≤ 0.02944 under the constraint that the first term shall be 1. The maximum relative error of this approximation is less than 2**-60.55.

LOG2(x) or LOG10(x) is calculated by multiplying the result by LOG2(e) or LOG10(e) respectively.

Error and Exceptional Conditions

The ERROR condition is raised if x ≤ 0.

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Called By

Compiled code. Entry point IBMBMDLA(log to base e) is also called by:

IBMBMLL - ATANH (long float real).
IBMBMYL - General exponentiation (long float real).
IBMBMYY - General exponentiation (long float complex).
IBMBMDY - LOG (long float complex).

IBMBMDS - LOG, LOG2, LOG10 (Short Float Real)

Function

To calculate the logarithm of x to the base e, 2, or 10. The module has three entry points:

IBMBMDSA: LOG

IBMBMDSB: LOG2

IBMBMDSC: LOG10

Method

Let x = 16**p*2**(-q)*m where p is the exponent, q is an integer such that $0 \le q \le 3$, and $1/2 \le m < 1$.

Two constants, a (= base point) and b (= -LOG2(a)), are defined as follows:

$1/2 \le m <$ 1/SQRT(2): a = 1/2, b = 1

1/SQRT(2) $\le m < 1$: a = 1, b = 0

Let y = (m-a)/(m+a).

Then m = a*(1+y)/(1-y) and ABS(y) < 0.1716.

Now x = (2**(4*p-q-b))*((1+y)/(1-y)). Therefore:

    LOG(x) = (4*p-q-b)*LOG(2) + LOG((1+y)/(1-y))

To obtain LOG((1+y)/(1-y)) first w = 2*y (m-a)/(0.5m+0.5a) is computed. The following approximation is performed:

    LOG((1+y)/(1-y)) = w*(c_0 + c_1*w**2/(c_2 - w**2))

The coefficients were obtained by the minimax rational approximation of LOG((1+y)/(1-y))/(2*y), in relative error, under the constraint that the first term (c_0) shall be one. The maximum relative error of this approximation is less than 2**-25.33.

LOG2(x) or LOG10(x) is calculated by multiplying the result by LOG2(e) or LOG10(e) respectively.

Error and Exceptional Conditions

The ERROR condition is raised if $x \le 0$.

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Called By

Compiled code. Entry point IBMBMDSA(log to base e) is also called by:

IBMBMLS - ATANH (short float real).
IBMBMYS - General exponentiation (short float real).
IBMBMYX - General exponentiation (short float complex).
IBMBMDX - LOG (short float complex).

IBMBMDX - LOG (Short Float Complex)

Function

To calculate the principal value of the natural logarithm of z. The principal value is that result satisfying the condition:

-pi < imaginary part of result ≤ pi

Method

Let LOG(x+yI) = a+bI.

Then a = LOG(ABS(x+yI)) and b = ATAN(y,x)

LOG(ABS(x+yI)) is computed as follows:

Let v_1 = MAX(ABS(x),ABS(y))

and v_2 = MIN(ABS(x),ABS(y))

Let t be the exponent of v_1 (i.e., v_1 = m*16**t, 1/16 ≤ m < 1)

Let t_1 = t if t ≤ 0

or t_1 = t-1 if t> 0

and s = 16**t_1

Then LOG(ABS(x+yI)) = 4*t_1*LOG(2)+ LOG((v_1/s)**2+(v_2/s)**2)/2

Computation of v_1/s and v_2/s are carried out by suitable adjustment of the characteristics of v_1 and v_2 .If v_2/s << 1, it is taken to be 0.

Error and Exceptional Conditions

The ERROR condition is raised in the called real ATAN routine IBMBMKS if x = y = 0.

Linkage

```
R1    = A(PLIST)
PLIST = A(z)
        A(target)
```

Calls

IBMBMDS - LOG (short float real).
IBMBMKS - ATAN (short float real).

Called By

Compiled code.
IBMBMYX - General exponentiation (short float complex).

IBMBMDY - LOG (Long Float Complex)

Function

To calculate the principal value of the natural logarithm of z. The principal value is that result satisfying the condition

-pi < imaginary part of result ≤ pi

Method

Let LOG(x+yI) = a+bI.

Then, a = LOG(ABS(x+yI)) and b = ATAN(y,x)

LOG(ABS(x+yI)) is computed as follows:

Let v_1 = MAX(ABS(x),ABS(y)) and

v_2 = MIN(ABS(x),ABS(y))

Let t be the exponent of v_1 (i.e., v_1 = m*16**t, 1/16 ≤ m < 1)

Let t_1 = t if t ≤ 0

or t_1 = t-1 if t > 0

and s = 16**t_1

Then LOG(ABS(x+yI)) = 4*t_1*LOG(2)+ LOG((v_1/s)**2+(v_2/s)**2)/2

Computation of v_1/s and v_2/s are carried out by suitable adjustment of the characteristics of v_1 and v_2 in particular, if v_2/s << 1, it is taken to be 0.

Error and Exceptional Conditions

The ERROR condition is raised in the called real ATAN routine IBMBMKL if x = y = 0.

Linkage

```
R1    = A(PLIST)
PLIST = A(z)
        A(target)
```

Calls

IBMBMDL - LOG (long float real).
IBMBMKL - ATAN (long float real).

Called By

Compiled code.
IBMBMYY - General exponentiation (long float complex).

IBMBMGL - SIN, SIND, COS, COSD (Long Float Real)

Function

To calculate sin x or cos x where x is in radians or degrees. The module has four entry points:

IBMBMGLA: SIN

IBMBMGLB: SIND

IBMBMGLC: COS

IBMBMGLD: COSD

Method

Let y = ABS(x)/(pi/4) for x in radians,
or y = ABS(x)/45 for x in degrees.
and y = q + r, q integral, $0 \le r < 1$.

Take q_1 = q for SIN or SIND with positive or zero argument,
q_1 = q + 2 for COS or COSD,
q_1 = q + 4 for SIN or SIND with negative argument,
and q_2 = MOD(q_1,8).

Since COS(x) = SIN(ABS(x) + pi/2)
and SIN(-x) = SIN(ABS(x) + pi),

it is only necessary to find

SIN(pi/4*(q_2 + r)), for $0 \le q_2 \le 7$.

Therefore compute:

SIN(pi/4*r),	if q_2 = 0 or 4,
COS(pi/4*(1 - r)),	if q_2 = 1 or 5,
COS(pi/4*r),	if q_2 = 2 or 6,
SIN(pi/4*(1 - r))	if q_2 = 3 or 7.

SIN(pi/4*r_1)/r_1, where r_1 is r or (1 - r), is computed by using the Chebyshev interpolation polynomial of degree 6 in r_1**2, in the range $0 \le r_1$**2 ≤ 1, with maximum relative error 2**(-58).

COS(pi/4*r_1) is computed by using the Chebyshev interpolation polynomial of degree 7 in r_1**2, in the range $0 \le r_1$**2 ≤ 1, with maximum relative error 2**(-64.3).

Finally, if $q_2 \ge 4$ a negative sign is given to the result.

Error and Exceptional Conditions

The ERROR condition is raised if

ABS(x) ≥ 2**50*K, where K = pi if x is in radians
or K = 180 if x is in degrees

Linkage

R1 = A(PLIST)
PLIST = A(x)
A(target)

Called By

Compiled code. Entry points IBMBMGLA(SIN) and IBMBMGLC (COS) are also called by:

IBMBMBY - EXP (long float complex).
IBMBMGY - SIN, SINH, COS, COSH (long float complex).
IBMBMHY - TAN, TANH (long float complex).

IBMBMGS - SIN, SIND, COS, COSD (Short Float Real)

Function

To calculate sin x or cos x, where x is in radians or degrees. The module has four entry points:

IBMBMGSA: SIN

IBMBMGSB: SIND

IBMBMGSC: COS

IBMBMGSD: COSD

Method

Let k = pi/4

Evaluate p = ABS(x)*(1/k) if x is in radians
      or p = ABS(x)*(1/45) if x is in degrees
using long-precision multiplication to safeguard accuracy.

Separate p into integer part q and fractional part r, i.e., p = q + r where $0 \le r < 1$.

Define q_1 = q if SIN or SIND is required and $x \ge 0$;
       q_1 = q + 2 if COS or COSD is required;
       q_1 = q + 4 if SIN or SIND is required and $x < 0$.

Then for all values of x each case has been reduced to the computation of SIN(k*(q_1+r)) = SIN(t) say, where $t \ge 0$.

Let q_2 = Mod(q_1,8).

If q_2 = 0, SIN(t) = SIN(k*r)
if q_2 = 1, SIN(t) = COS(k*(1-r))
If q_2 = 2, SIN(t) = COS(k*r)
If q_2 = 3, SIN(t) = SIN(k*(1-r))
If q_2 = 4, SIN(t) = -SIN(k*r)
If q_2 = 5, SIN(t) = -COS(k*(1-r))
if q_2 = 6, SIN(t) = -COS(k*r)
if q_2 = 7, SIN(t) = -SIN(k*(1-r)).

Thus it is necessary to compute only SIN(k*r_1) or COS(k*r_1); where r_1 = r or 1 - r and $0 \le r_1 \le 1$; as follows:

1. SIN(k*r_1) = r_1*(a_0 +a_1r_1**2+a_2r_1**4+a_3r_1**6)

   The coefficients were obtained by the Chebyshev interpolation. The maximum relative error is less than 2**-28.1.

2. $COS(k*r_1) = 1+b_1r_1**2+b_2r_1**4+b_3r_1**6$

The coefficients were obtained by a variation of the minimax approximation which provides partial rounding for the short precision computation. The maximum absolute error is 2**-24.57.

Error and Exceptional Conditions

The ERROR condition is raised if

```
ABS(x) ≥ 2**18*K,     where K = pi if x is in radians
                         or K = 180 if x is in degrees
```

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Called By

Compiled code. Entry points IBMBMGSA(SIN) and IBMBMGSC (COS) are also called by:

IBMBMBX - SIN (short float complex).
IBMBMGX - SIN, SINH, COS, COSH (short float complex).
IBMBMHX - TAN, TANH (short float complex).

IBMBMGX - SIN, SINH, COS, COSH (Short Float Complex)

Function

To calculate sin z, hyperbolic sin z, cos z, or hyperbolic cos z. The module has four entry points:

IBMBMGXA: SIN

IBMBMGXB: SINH

IBMBMGXC: COS

IBMBMGXD: COSH

Method

Let z = x + YI.

```
Then REAL(SIN(z))   =  SIN(x)*COSH(y)
and  IMAG(SIN(z))   =  COS(x)*SINH(y);

     REAL(COS(z))   =  COS(x)*COSH(y)
and  IMAG(COS(z))   = -SIN(x)*SINH(y);

     REAL(SINH(z))  = COS(y)*SINH(x)
and  IMAG(SINH(z))  =  SIN(y)*COSH(x);

     REAL(COSH(z))  =  COS(y)*COSH(x)
and  IMAG(COSH(z))  =  SIN(y)*SINH(x).
```

To avoid making calls to evaluate SINH and COSH separately, and thus frequently having to evaluate EXP twice for the same argument, SINH(u) is computed as follows:

Case 1: u > 0.3465736

SINH(u) = (EXP(u) - 1/EXP(u))/2.

Case 2: 0 ≤ u ≤ 0.3465736

SINH(u)/u is approximated by a polynomial of the form a0 + a_1*u**2 + a_1*u**4 (which has a relative error of less than 2**-26.18). The coefficients were obtained by the minimax approximation in relative error of SINH(x)/x over the range 0 ≤ x**2 ≤ 0.12011 under the constraint that the first term shall be exactly 1.

Case 3: u ≤ 0

SINH(u) = -SINH(-u). Then
COSH(u) = SINH(ABS(u)) + 1/EXP(ABS(u)).

Error and Exceptional Conditions

The ERROR condition is raised in the called real SIN routine IBMBMGS if

ABS(v) ≥ 2**18*pi

where v = x for entry points IBMBMGXA and C (SIN and COS)
and v = y for entry points IBMBMGXB and D (SINH and COSH).

The OVERFLOW condition can occur in the called EXP routine IBMBMBS.

Linkage

```
R1    = A(PLIST)
PLIST = A(z)
        A(target)
```

Calls

IBMBMGS - SIN,COS (short float real).
IBMBMBS - EXP (short float real).

Called By

Compiled code.

IBMBMGY - SIN, SINH, COS, COSH (Long Float Complex)

Function

To calculate sin z, hyperbolic sin z, cos z, or hyperbolic cos z. The module has four entry points:

IBMBMGYA: SIN

IBMBMGYB: SINH

IBMBMGYC: COS

IBMBMGYD: COSH

Method

Let z = x + yI.

```
Then REAL(SIN(z)) = SIN(x)*COSH(y)
and  IMAG(SIN(z)) = COS(x)*SINH(y);

     REAL(COS(z)) = COS(x)*COSH(y)
and  IMAG(COS(z)) = -SIN(x)*SINH(y);

     REAL(SINH(z)) = COS(y)*SINH(x)
and  IMAG(SINH(z)) = SIN(y)*COSH(x);

     REAL(COSH(z)) = COS(y)*COSH(x)
and  IMAG(COSH(z)) = SIN(y)*SINH(x).
```

To avoid making calls to evaluate SINH and COSH separately, and thus frequently having to evaluate EXP twice for the same argument, SINH(u) is computed as follows:

Case 1: u ≥ 0.481212

SINH(u) = (EXP(u) - 1/EXP(u))/2

Case 2: 0 ≤ u < 0.481212

SINH(u)/u is approximated by a polynomial of the fifth degree in u**2 which has a relative error of less than 2**-56.07. The coefficients were obtained by the minimax approximation in relative error of SINH(x)/x over the range 0 ≤ x**2 ≤ 0.23156 under the constraint that the first term shall be exactly 1.

Case 3: u < 0

SINH(u) = -SINH(-u). Then
COSH(u) = SINH(ABS(u)) + 1/EXP(ABS(u)).

Error and Exceptional Conditions

The ERROR condition is raised in the called real SIN routine IBMBMGL if

ABS(v) ≥ 2**50*pi

where v = x for entry points IBMBMGYA and C (SIN and COS)
and v = y for entry points IBMBMGYB and D (SINH and COSH).

The OVERFLOW condition can occur in the called EXP routine IBMBMBL.

Linkage

R1 = A(PLIST)

PLIST = A(z)
A(target)

Calls

IBMBMGL - SIN, COS (long float real).
IBMBMEL - EXP (long float real).

Called By

Compiled code.

IBMBMHL - TAN, TAND (Long Float Real)

Function

To calculate tan x, where x is in radians or degrees. The module has two entry points:

IBMBMHLA: TAN

IBMBMHLB: TAND

Method

Evaluate p = (4/pi)*ABS(x) if x is in radians
or p = (1/45)*ABS(x) if x is in degrees.

Let q and r be respectively the integral and fractional parts of p.

If q is even, put s = r;
if q is odd, put s = 1 - r.

Let q_1 = MOD(q,4). Then

If q_1 = 0, TAN(ABS(x)) = TAN(pi*s/4)
If q_1 = 1, TAN(ABS(x)) = COT(pi*s/4)
If q_1 = 2, TAN(ABS(x)) = -COT(pi*s/4)
If q_1 = 3, TAN(ABS(x)) = -TAN(pi*s/4)

Compute TAN(pi*s/4) and COT(pi*s/4) as the ratio of two polynomials:

TAN(pi*s/4) = s*P(s**2)/Q(s**2))
COT(pi*s/4) = Q(s**2)/(s*P(s**2))

where both P and Q are polynomials of degree 3 in s**2. The coefficients of P and Q were obtained by the minimax rational approximation (in relative error) of TAN(pi*s/4) of the indicated form. The maximum relative error of this approximation is 2**-55.6.

Finally, if x < 0, TAN(x) = -TAN(ABS(x)).

Error and Exceptional Conditions

The OVERFLOW condition can occur in this module.

The ERROR condition is raised if

ABS(x) ≥ 2**50*K, where K = pi if x is in radians
or K = 180 if x is in degrees.

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Called By

Compiled code.

<u>IBMBMHS - TAN, TAND (Short Float Real)</u>

Function

To calculate tan x, where x is in radians or degrees. The module has two entry points:

<u>IBMBMHSA</u>: TAN

<u>IBMBMHSB</u>: TAND

Method

Evaluate p = (4/pi)*ABS(x) if x is in radians

or p = (1/45)*ABS(x) if x is in degrees,

using long-precision multiplication to safeguard accuracy.

Let q and r be respectively the integral and fractional parts of p.

If q is even, put s = r;
if q is odd, put s = 1-r.

Let q_1 = MOD(q,4). Then

If q_1 = 0, TAN(ABS(x)) = TAN(pi*s/4)
If q_1 = 1, TAN(ABS(x)) = COT(pi*s/4)
If q_1 = 2, TAN(ABS(x)) = -COT(pi*s/4)
If q_1 = 3, TAN(ABS(x)) = -TAN(pi*s/4)

Compute TAN(pi*s/4) and COT(pi*s/4) as the ratio of two polynomials:

TAN(pi*s/4) = s*P(u)/Q(u)

COT(pi*s/4) = Q(u)/(s*P(u))

where u = s**2/2

P(u) = -8.460901 + u

and Q(u) = -10.772754 + 5.703366*u - 0.159321*u**2

These coefficients were obtained by the minimax rational approximation in relative error of the above form. The maximum relative error of this approximation is 2**-26. The variable u, rather than s**2, was chosen for P and Q in order to improve the rounding effect of the coefficients.

Finally, if x < 0, put

TAN(x) = -TAN(ABS(x)).

Error and Exceptional Conditions

The OVERFLOW condition can occur in this module.

The ERROR condition is raised if

```
ABS(x) ≥ 2**18*K,      where K = pi  if x is in radians
                          or K = 180 if x is in degrees.
```

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Called By

Compiled code.

IBMBMHX - TAN, TANH (Short Float Complex)

Function

To calculate tan z or hyperbolic tan z. The module has two entry points:

IBMBMHXA: TAN

IBMBMHXB: TANH

Method

Let z = x + yI.

```
Then REAL(TAN(z))   =   SIN(2*x)/(COS(2*x) + COSH(2*y)).

     IMAG(TAN(z))   =   SINH(2*y)/(COS(2*x) + COSH(2*y)).

and  TANH(z) = -(TAN(zI))I
```

Error and Exceptional Conditions

The OVERFLOW condition can occur in this module.

The ERROR condition is raised in the called module IBMBMGS if:

ABS(u) ≥ 2**18*pi

```
where u = 2*x for entry point IBMBMHXA,
      u = 2*y for entry point IBMBMHXB.
```

OVERFLOW can occur in module IBMBMBS.

Linkage

```
R1    = A(PLIST)
PLIST = A(z)
        A(target)
```

Calls

IBMBMGS - SIN, SIND, COS, COSD (short float real).
IBMBMIS - SINH, COSH (short float real).

Called By

Compiled code.

IBMBMHY - TAN, TANH (Long Float Complex)

Function

To calculate tan z or hyperbolic tan z. The module has two entry points:

IBMBMHYA: TAN

IBMBMHYB: TANH

Method

Let z = x + yI.

Then REAL(TAN(z)) = SIN(2*x)/(COS(2*x) + COSH(2*y)).

IMAG(TAN(z)) = SINH(2*y)/(COS(2*x) + COSH(2*y)).

and TANH(z) = -(TAN(zI))I

Error and Exceptional Conditions

The OVERFLOW condition can occur in this module.

The ERROR condition is raised in the called module IBMBMGL if:

ABS(u) ≥ 2**50*pi

where u = 2*x for entry point IBMBMHYA,
u = 2*y for entry point IBMBMHYB.

OVERFLOW can occur in module IBMBMBL.

Linkage

```
R1    = A(PLIST)
PLIST = A(z)
        A(target)
```

Calls

IBMBMGL - SIN, SIND, COS, COSD (long float real).
IBMBMIL - SINH, COSH (long float real).

Called By

Compiled code.

IBMBMIL - SINH, COSH (Long Float Real)

Function

To calculate hyperbolic sin x or hyperbolic cos x. The module has two entry points:

IBMBMILA: SINH

IBMBMILB: COSH

Method

Case 1: ABS(x) < 0.881374

$$SINH(x) = c_0*x + c_1*x**3 + \ldots.. + c_{16}*x**13$$

$$COSH(x) = EXP(x)/2 + 0.5/EXP(x)$$

The coefficients C were obtained by the minimax approximation in relative error of SINH(x)/x as the function of x**2. The maximum relative error of this approximation is 2**-55.7.

Case 2: x ≥ 0.881374

SINH(x) = EXP(x)/2 - 0.5/EXP(x),

COSH(x) = EXP(x)/2 + 0.5/EXP(x).

These two versions of EXP(x)/2 ± 0.5/EXP(x) are preferable to the equivalent versions of (EXP(x) - 1/EXP(x))/2 because, in floating-point, 0.5 has three more significant bits than 1.0.

Case 3: x ≤ -0.881374

The computation is reduced to case 2 by

SINH(x) = -SINH(ABS(x)).

COSH(x) = EXP(x)/2+0.5/EXP(x)

Error and Exceptional Conditions

The OVERFLOW condition can occur in the called routine IBMBMBL.

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Calls

IBMBMBL - EXP (long float real).

Called By

IBMBMHY - TAN, TANH (long float complex).
Compiled code.

IBMBMIS - SINH, COSH (Short Float Real)

Function

To calculate hyperbolic sin x or hyperbolic cos x. The module has two entry points:

IBMBMISA: SINH

IBMBMISB: COSH

Method

Case 1: ABS(x) < 1

SINH(x) = x+c_1*x**3+c_2*x**5+c_3*x**7

COSH(x) = EXP(x)/2+0.5/EXP(x)

The coefficients c were obtained by the minimax approximation in relative error of SINH(x)/x as the function of x**2. The maximum relative error of this approximation is 2**-25.6.

Case 2: x ≥ 1

SINH(x) = EXP(x)/2 - 0.5/EXP(x).

COSH(x) = EXP(x)/2 + 0.5/EXP(x).

These two versions of EXP(x)/2 ± 0.5/EXP(x) are preferable to the equivalent versions of (EXP(x) - 1/EXP(x))/2 because, in floating-point, 0.5 has three more significant bits than 1.0.

Case 3: x ≤ -1

The computation is reduced to case 2 by

SINH(x) = -SINH(ABS(x))

COSH(x) = EXP(x)/2+0.5/EXP(x)

Error and Exceptional Conditions

The OVERFLOW condition can occur in the called routine IBMBMBS.

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Calls

IBMBMBS - EXP (short float real).

Called By

IBMBMHX - TAN, TANH (short float complex)
Compiled code.

IBMBMJL - TANH (Long Float Real)

Function

To calculate the byperbolic tangent of x.

Method

Case 1: ABS(x) ≤ 2**-28

Return x as result.

Case 2: 2**-28 < ABS(x) < 0.54931

Use a transformed minimax approximation of the form:

$$TANH(x)/x = c + d_1*x**2/(x**2+c_1) + d_2/(x**2+c_2)+d_3/(x**2+c_3)$$

The minimax of relative error was taken over the range x**2 ≤ 0.30174 under the constraint that the first term is 1. The maximum relative error is 2**-63.

Case 3: 0.54931 ≤ x < 20.101

TANH(x) = 1 - 2/(EXP(2*x)+1)

Case 4: x ≥ 20.101

Return result 1.

Case 5: x ≤ -0.54931

TANH(x) = -TANH(-x)

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Calls

IBMBMBL - EXP (long float real).

Called By

Compiled code.

IBMBMJS - TANH (Short Float Real)

Function

To calculate the hyperbolic tangent of x.

Method

Case 1: ABS(x) ≤ 2**-12

Return x as result.

Case 2: 2**-12 < ABS(x) ≤ 0.7

Use a transformed continued fraction of the form:

TANH(x)/x = 1 - x**2*(.0037828+.8145651/(x**2+2.471749))

The coefficients of this approximation were obtained by taking the minimax of relative error, over the range x**2 < 0.49, of approximations of this form under the constraint that the first term shall be 1. The maximum relative error of this approximation is 2**-26.4.

Case 3: 0.7 < x < 9.011

Use TANH(x) = 1 - 2/(EXP(2*x) + 1).

Case 4: x ≥ 9.011

Return result 1.

Case 5: x < -0.7

TANH(x) = -TANH(-x).

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Calls

IBMBMBS - EXP (short float real).

Called By

Compiled code.

IBMBMKL - ATAN, ATAND (Long Float Real)

Function

To calculate arctan x or arctan(y/x). The result ranges are:

```
Arctan x(radians)      - ± pi/2
Arctan(y/x)(radians)   - ± pi
Arctan x(degrees)      - ± 90°
Arctan(y/x)(degrees)   - ± 180°
```

The module has four entry points:

IBMBMKLA: ATAN(x)

IBMBMKLB: ATAND(x)

IBMBMKLC: ATAN(y,x)

IBMBMKLD: ATAND(y,x)

Method

1. ATAN(y,x)

   If x = 0 or ABS(y/x) ≥ 2**56, the answer SIGN(y)*pi/2 is returned except for the error case x = y = 0. Otherwise

   ATAN(y,x) = ATAN(y/x) if x > 0

   or ATAN(y,x) = ATAN(y/x) + SIGN(y)*pi if x < 0

   Hence the computation is now reduced to the single argument case.

2. ATAN(x)

   The general case may be reduced to the range 0 ≤ x ≤ 1 since

   ATAN(-x) = - ATAN(x)

and ATAN(1/ABS(x)) = pi/2 - ATAN(ABS(x)).

A further reduction to the range ABS(x) ≤ TAN(pi/12) is made by using

ATAN(x) = pi/6 + ATAN((SQRT(3)*x - 1)/(x + SQRT(3)))

Care is taken to avoid the loss of significant digits in computing SQRT(3)*x - 1.

For the basic range ABS(x) ≤ TAN(pi/12), use a continued fraction of the form

$$\mathrm{ATAN}(x)/x = 1+u*(b - a_1/(b_1+u-a_2/(b_2+u-a_3/(b_3+u))))$$

where u = x**2.

The relative error of this approximation is less than 2**-60.7.

The coefficients of this formula were derived by transforming a minimax rational approximation in relative error over the range 0 ≤ u ≤ 0.071797 for ATAN(x)/x of the following form:

$$\mathrm{ATAN}(x)/x = a + u*((c + c_1*u + c_2*u*u + c_3*u*u*u)/(d + d_1*u + d_2*u*u + u*u*u)).$$

under the constraint that a = 1.

3. ATAND(x) and ATAND(y,x)

The treatment is as above with the addition of a final conversion of the result to degrees.

Error and Exceptional Conditions

Entry Points IBMBMKLC and D:

The ERROR condition is raised if x = y = 0.

Linkage

Entry Points IBMBMKLA and B:

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Entry Points IBMBMKLC and D:

```
R1    = A(PLIST)
PLIST = A(y)
        A(x)
        A(target)
```

Called By

Compiled code. In addition, entry point IBMBMKLC is called by:

IBMBMDY - LOG (long float complex).
IBMBMKY - ATAN, ATANH (long float complex).

IBMBMKS - ATAN, ATAND (Short Float Real)

Function

To calculate arctan x or arctan(y/x). The result ranges are:

Arctan x(radians)	- ± pi/2
Arctan(y/x)(radians)	- ± pi
Arctan x(degrees)	- ± 90°
Arctan(y/x)(degrees)	- ± 180°

The module has four entry points:

IBMBMKSA: ATAN(x)

IBMBMKSB: ATAND(x)

IBMBMKSC: ATAN(y,x)

IBMBMKSD: ATAND(y,x)

Method

1. ATAN(y,x)

   If x = 0 or ABS(y/x) ≥ 2**24, the answer SIGN(y)*pi/2 is returned except for the error case x = y = 0. Otherwise

   ATAN(y,x) = ATAN(y/x) if x > 0

   or ATAN(y,x) = ATAN(y/x) + SIGN(y)*pi if x < 0.

   Hence the computation is now reduced to the single argument case.

2. ATAN(x)

   The general case may be reduced to the range 0 ≤ x ≤ 1 since

   ATAN(-x) = -ATAN(x), and

   and ATAN(1/ABS(x)) = pi/2 - ATAN(ABS(x)).

   A further reduction to the range ABS(x) ≤ TAN(pi/12) is made by using

   ATAN(x) = pi/6 + ATAN((SQRT(3)*x - 1)/(x + SQRT(3)))

   Care is taken to avoid the loss of significant digits in computing SQRT(3)*x - 1.

   For the basic range ABS(x) ≤ TAN(pi/12), use an approximation formula of the form

   ATAN(x)/x = a + b*x**2 + c/(d + x**2)

   with relative error less than 2**-27.1.

3. ATAND(x) and ATAND(y,x)

   The treatment is as above with the addition of a final conversion of the result to degrees.

Error and Exceptional Conditions

Entry Points IBMBMKSC and D: The ERROR condition is raised if x=y=0.

Linkage

Entry Points IBMBMKSA, IBMBMKSB:

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Entry Points IBMBMKSC, IBMBMKSD:

```
R1    = A(PLIST)
PLIST = A(y)
        A(x)
        A(target)
```

Called By

Compiled code. In addition, entry point IBMBMKSC is called by:

IBMBMDX - LOG (short float complex).
IBMBMKX - ATAN, ATANH (short float complex).

IBMBMKX - ATAN, ATANH (Short Float Complex)

Function

To calculate arctan z or hyperbolic arctan z. The module has two entry points:

IBMBMKXA: ATAN

IBMBMKXB: ATANH

Method

Let z = x + yI.

Then

```
        REAL(ATANH(z)) = (ATANH(2*x/(1 + x*x + y*y)))/2

        IMAG(ATANH(z)) = (ATAN((2*y),(1 - x*x - y*y)))/2

 and    ATAN(z) = -(ATANH(zI))I
```

Error and Exceptional Conditions

ATAN: The ERROR condition is raised if z = ±1I

ATANH: The ERROR condition is raised if z = ±1

Linkage

```
R1    = A(PLIST)
PLIST = A(z)
        A(target)
```

Calls

IBMBMKS - ATAN (short float real).
IBMBMLS - ATANH (short float real).

Called By

Compiled code.

IBMBMKY - ATAN, ATANH (Long Float Complex)

Function

To calculate arctan z or hyperbolic arctan z. The module has two entry points:

IBMBMKYA: ATAN

IBMBMKYB: ATANH

Method

Let z = x + yI.

Then

```
        REAL(ATANH(z)) = (ATANH(2*x/(1 + x*x + y*y)))/2

        IMAG(ATANH(z)) = (ATAN((2*y),(1 - x*x - y*y)))/2

 and    ATAN(z) = -(ATANH(zI))I
```

Error and Exceptional Conditions

ATAN: The ERROR condition is raised if z = ±1I.

ATANH: The ERROR condition is raised if z = ±1.

Linkage

```
R1    = A(PLIST)
PLIST = A(z)
        A(target)
```

Calls

IBMBMKL - ATAN (long float real).
IBMBMIL - ATANH (long float real).

Called By

Compiled code.

IBMBMLL - ATANH (Long Float Real)

Function

To calculate hyperbolic arctan x.

Method

Case 1: ABS(x)< 0.25

Use a Chebyshev polynomial of degree 8 in x**2 to compute ATANH(x)/x.

Case 2: 0.25 ≤ ABS(x)<1

Compute ATANH(x) as:

```
ATANH(x) = -SIGN(x)*0.5*LOG((0.5 - ABS(x/2))/(0.5 + ABS(x/2)))
```

Error and Exceptional Conditions

The ERROR condition is raised if if ABS(x) ≥ 1.

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Calls

IBMBMDL - LOG (long float real).

Called By

Compiled code.
IBMBMKY - ATAN, ATANH (long float complex).

IBMBMLS - ATANH (Short Float Real)

Function

To calculate hyperbolic arctan x.

Method

Case 1: ABS(x) ≤ 0.2

Use a rational approximation of the form:

```
ATANH(x) = x + x**3/(a + b*x**2)
```

<u>Case 2</u>: 0.2 < ABS(x) < 1

Compute as:

```
ATANH(x) = -SIGN(x)*0.5*LOG((0.5 - ABS(x/2))/(0.5 + ABS(x/2)))
```

Error and Exceptional Conditions

The ERROR condition is raised if ABS(x) ≥ 1.

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Calls

IBMBMDS - LOG (short float real).

Called By

Compiled code.
IBMBMKX - ATAN, ATANH (short float complex).

<u>IBMBMML - ASIN, ACOS (Long Float Real)</u>

Function

To calculate ASIN x or ACOS x. The module has two entry points:

<u>IBMBMMLA</u>: ASIN

<u>IBMBMMLB</u>: ACOS

Method

<u>Case 1</u>: $0 \le x \le 1/2$

Compute ASIN(x) by a continued fraction of the form:

ASIN(x) = x + F*x**3

Where F $= c_1 + d_1/(x{**}2 + c_2 + d_2/(x{**}2 + c_3 + d_3/(x{**}2 + c_4 + d_4/(x{**}2 + c_5))))$

The coefficients of this formula were derived by transforming the minimax rational approximation (in relative error, over the range $0 \le x{**}2 \le 1/4$) for ASIN(x)/x of the following form:

$$ASIN(x)/x = a_0 + x^{**}2^*(a_1 + {}_2{}^*x^{**}2 + a_3{}^*x^{**}4 + a_4{}^*x^{**}6 + a_5{}^*x^{**}8)/ (b_0 + b_1{}^*x^{**}2 + b_2{}^*x^{**}4 + b_3{}^*x^{**}6 + x^{**}8) + \ldots$$

Minimax was taken under the constraint that a = 1 exactly. The relative error of this approximation is less than 2**-57.2.

ACOS(x) is computed as:

ACOS(x) = pi/2 - ASIN(x)

Case 2: 1/2 < x ≤ 1

Compute ACOS(x) as:

ACOS(x) = 2*ASIN(SQRT((1 - x)/2))

Case 2 is thus reduced to case 1, because within these limits

0 ≤ SQRT((1 - x)/2) ≤ 1/2

ASIN(x) is computed as:

ASIN(x) = pi/2 - ACOS(x)

Case 3: -1 ≤ x < 0

For negative x, ASIN(x) is computed as:

ASIN(x) = -ASIN(ABS(x))

and ACOS(x) is computed as:

ACOS(x) = pi - ACOS(ABS(x))

Case 3 thus reduces to case 1 or case 2.

Error and Exceptional Conditions

The ERROR condition is raised if ABS(x) > 1.

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(target)
```

Calls

IBMBMAI - SQRT (long float real).

Called By

Compiled code.

IBMBMMS - ASIN, ACOS (Short Float Real)

Function

To calculate ASIN(x) or ACOS(x). The module has two entry points:

IBMBMMSA: ASIN

IBMBMMSB: ACOS

Method

Case 1: $0 \le x \le 1/2$

Compute ASIN(x) by a continued fraction of the form:

ASIN(x) = x + F*x**3

where F = d_1/(x**2 + c_1 + d_2/(x**2 + c_2))

The coefficients of this formula were derived by transforming the minimax rational approximation (in relative error, over the range (0 ≤ x**2 ≤ 1/4) for ASIN(x)/x of the following form:

ASIN(x)/x = a + x**2*(a_1 + a_2*x**2)/ (b + b_1*x**2 + x**4) + ...

Minimax was taken under the constraint that a = 1 exactly. The relative error of this approximation is less than 2**-28.3.

ACOS(x) is computed as:

ACOS(x) = pi/2 - ASIN(x)

Case 2: $1/2 < x \le 1$

Compute ACOS(x) as

ACOS(x) = 2*ASIN(SQRT((1 - x)/2))

Case 2 is thus reduced to case 1, because within these limits 0 ≤ SQRT((1 - x)/2) ≤ 1/2.

ASIN(x) is computed as:

ASIN(x) = pi/2 - ACOS(x)

Case 3: $-1 \le x < 0$

For negative x, ASIN(x) is computed as:

ASIN(x) = -ASIN(ABS(x))

and ACOS(x) is computed as:

ACOS(x) = pi - ACOS(ABS(x))

Case 3 thus reduces to case 1 or case 2.

Error and Exceptional Conditions

The ERROR condition is raised if ABS(x)>1.

Linkage

R1   = A(PLIST)
PLIST = A(x)
        A(target)

Calls

IBMBMAS - SQRT (short float real).

Called By

Compiled code.

IBMBMOD - ADD (Fixed Decimal)

Function

Entry point IBMBMODA: ADD(x,y,p,q), where x and y are fixed decimal real numbers and (p,q) is the required precision of the result.

Entry point IBMBMODB: ADD(z_1,z_2,p,q) where z_1 and z_2 are fixed decimal complex numbers and (p,q) is the required precision of the result.

Method

Real Arguments (IBMBMODA): If both arguments are non-zero, a call to module IBMBMUDA is used to shift the argument with the larger scale factor to give it the scale factor of the other, and to convert it to precision 31. The arguments are then added together, and module IBMBMUDA is again called to convert the sum to the specified precision and assign it to the target field. If one of the arguments is zero, the other is treated as the sum above.

Complex Arguments (IBMBMODB): The real parts of each argument are added and the sum is assigned to the target field by using entry point IBMBMODA. The imaginary parts are treated similarly.

Error and Exceptional Conditions

The FIXEDOVERFLOW or the SIZE condition can occur in the called module IBMBMUDA.

Linkage

R1   = A(PLIST)
PLIST = A(x or z_1)
        A(DED for x or z_1)
        A(y or z_2)
        A(DED for y or z_2)
        A(target)
        A(DED for target)

Calls

IBMBMUD - decimal shift-and-load, shift-and-assign.

Called By

Compiled code.

IBMBMPU - MULTIPLY (Fixed Binary Complex)

Function

MULTIPLY(z_1,z_2,p,q) where z_1 and z_2 are compled fixed-point binary numbers, and (p,q) is the required precision of the result.

Method

Let the arguments be z_1 = a + bI and z_2 = c + dI. Then:

REAL(z_1*z_2) = a*c - b*d

and IMAG(z_1*z_2) = b*c + a*d

The real and imaginary parts of the product are computed. These numbers are then shifted to give them the required scale factor(q).

The results of the shifts are tested for FIXEDOVERFLOW and truncated by left shifts.

Error and Exceptional Conditions

The FIXEDOVERFLOW condition can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(z1)
        A(DED for z1)
        A(z2)
        A(DED for z2)
        A(target)
        A(DED for target)
```

Called By

Compiled code.

IBMBMPV - MULTIPLY (Fixed Decimal Complex)

Function

MULTIPLY(z_1,z_2,p,q) where z_1 and z_2 are complex fixed-point decimal numbers, and (p,q) is the required precision of the result.

Method

Let z_1 = a + bI and z_2 = c + dI, then:

REAL(z_1*z_2) = a*c - b*d.

and IMAG(z_1*z_2) = b*c + a*d.

The real and imaginary parts are calculated and then each is assigned to the target with precision (p,q) by separate calls to entry point IBMBMUDA of the decimal shift and assign module IBMBMUD.

Error and Exceptional Conditions

The FIXEDOVERFLOW or the SIZE condition can occur in the called module IBMBMUD.

Linkage

```
R1    = A(PLIST)
PLIST = A(z1)
        A(DED for z1)
        A(z2)
        A(DED for z2)
        A(target)
        A(DED for target)
```

Calls

IBMBMUDA - decimal shift-and-assign

Called By

Compiled code.

IBMBMQU - DIVIDE (Fixed Binary Complex)

Function

DIVIDE(z_1,z_2,p,q) where z_1 and z_2 are complex fixed-point binary numbers, and (p,q) is the required precision of the result.

Method

Let z_1 = a + bI, and z_2 = c + dI, then:

REAL(z_1/z_2) = (a*c + b*d)/(c**2 + d**2)

and IMAG(z_1/z_2) = (b*c - a*d)/(c**2 + d**2)

The expressions a*c + b*d, b*c -a*d, and c**2 + d**2 are computed with a precision of 63. The denominator, c**2 + d**2, is shifted to precision 31 by either a right or left shift.

Two calls are then made to an incorporated subroutine which accepts a numerator and shifts it so that it has two insignificant leading digits. It then divides by c**2 + d**2 and shifts the quotient to the required scale factor (q).

Error and Exceptional Condition

The FIXEDOVERFLOW and the ZERODIVIDE conditions can occur in this module.

Linkage

```
R1     = A(PLIST)
PLIST  = A(z1)
         A(DED for z1)
         A(z2)
         A(DED for z2)
         A(target)
         A(DED for target)
```

Called By

Compiled code.

IBMBMQV - DIVIDE (Fixed Decimal Complex)

Function

DIVIDE(z_1,z_2,p,q) where z_1 and z_2 are complex fixed-point decimal numbers, and (p,q) is the required precision of the result.

Method

Let z_1 = a + bI, and z_2 = c + dI, then

REAL(z_1/z_2) = (a*c + b*d)/(c**2 + d**2)

and IMAG(z_1/z_2) + (b*c - a*d)/(c**2 + d**2)

The expressions a*c + b*d, b*c - a*d, and c**2 + d**2 are computed. Leading zeros are removed from the denominator (c**2 + d**2) by truncation on the left and a left shift if necessary. If the denominator is still more than 15 digits long it is truncated on the right to 15 digits.

Two calls are then made to an incorporated subroutine which accepts a numerator and shifts it to precision 31 with 2 leading zeros by calling IBMBMUD (via entry point IBMBMUDB). It then divides by c**2 + d**2 and calls IBMBMUD (via entry point IBMBMUDA) to assign the quotient to the target field with the required precision (p,q).

Error and Exceptional Conditions

The ZERODIVIDE condition can occur in this module.

The FIXEDOVERFLOW or the SIZE condition can occur in the called module IBMBMUD.

Linkage

```
R1    = A(PLIST)
PLIST = A(z1)
        A(DED for z1)
        A(z2)
        A(DED for z2)
        A(target)
        A(DED for target)
```

Calls

IBMBMUD - Decimal shift-and-assign.

Called By

Compiled code.

IBMBMRU - ABS (Fixed Binary Complex)

Function

To calculate ABS(z) = SQRT(x**2 + y**2), where z = x + yI.

Method

If x = y, result is x*SQRT(2). Otherwise,

let X1 = MAX(ABS(x),ABS(y)

Y1 = MIN(ABS(x),ABS(y)).

Then ABS(z) is computed as

X1*SQRT(1 + (Y1/X1)**2),

where the fixed binary calculation of SQRT(g) for 1 ≤ g < 2 is included within the module.

The first approximation to the square root is taken as

g/(1+g) + (1+g)/4,

with maximum relative error 1.8*2**-10.

One Newton-Raphson iteration gives maximum relative error 1.6**-20, and suffices if X1 < 2**(15-q) where q is the scale factor of z.

Otherwise a second iteration is used, with theoretical maximum relative error of 1.3*2**-40.

Error and Exceptional Conditions

The FIXEDOVERFLOW condition can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(z)
        A(DED for z)
        A(target)
        A(DED for target)
```

Called By

Compiled code

IBMBMRV - ABS (Fixed Decimal Complex)

Function

To calculate ABS(z) = SQRT(x**2 + y**2), where z = x + yI.

Method

x and y are converted to binary, with appropriate scaling if either exceeds 9 significant decimal digits.

Let X1 be the maximum, and Y1 the minimum, of the absolute values of the two binary numbers thus obtained.

Then if X1 = Y1 = 0, result 0 is returned. Otherwise, an approximation to ABS(z) is computed as

    X1*SQRT(1 + (Y1/X1)**2),

where the fixed binary calculation of SQRT(g) for $1 \le g \le 2$ is included within the module.

The first approximation to the square root is taken in the form

    A + B*(1 + g) - A/(1 + g)

with maximum relative error 2.17*10**-4. One Newton-Raphson iteration then gives a value with maximum relative error 2.35*10**8.

Multiplication by X1 produces a value for ABS(z) which is rounded and converted to decimal, and this suffices if it has not more than 7 significant decimal digits. Otherwise, this approximation is scaled if necessary and used in a final Newton-Raphson iteration for SQRT(x**2 + y**2) in decimal, with theoretical maximum relative error 2.76*10**-16.

Error and Exceptional Conditions

The FIXEDOVERFLOW condition can occur in this module.

Linkage

```
R1    = A(PLIST)
PLIST = A(z)
        A(DED for z)
        A(target)
        A(DED for target)
```

Called By

Compiled code.

IBMBMRX, IBMBMRY - ABS (Float Complex)

Modules

Argument	Module
short float	IBMBMRX
long float	IBMBMRY

Function

To calculate ABS(z) = SQRT(x**2 + y**2), where z = x + yI.

Method

Let z = x + yI.

If x = y = 0, answer is 0.

Otherwise let Z1 = MAX(ABS(x),ABS(y))

and Z2 = MIN(ABS(x),ABS(y)).

Then the answer is computed as:

Z1*SQRT(1 + (Z2/Z1)**2).

Error and Exceptional Conditions

The OVERFLOW condition can occur in this module

Linkage

```
R1    = A(PLIST)
PLIST = A(z)
        A(target)
```

Calls

IBMBMRX calls:
IBMBMAS - SQRT (short float real).

IBMBMRY calls:

IBMBMAL - SQRT (long float real).

Called By

Compiled code.

IBMBMUD - Shift-and-Assign, Shift-and-Load (Fixed Decimal)

Function

Shift-and-assign (Entry point IBMBMUDA): To convert a real fixed decimal number from precision (p_1, q_1) to precision (p_2, q_2), where $p_1 \leq 31$ and $p_2 \leq 15$.

Shift-and-load (Entry point IBMBMUDB): To convert a real fixed decimal number from precision (p_1, q_1) to precision (31, q_2), where $p_1 \leq 31$.

Method

The scale factor of the argument is subtracted from the scale factor of the target. The argument is then converted to precision 31 in a field with a shift equal to the magnitude of the difference between the scale factors, the shift being to the left if the difference is positive and tc the right if it is negative.

For shift-and-load, the field is moved unchanged to the target. For shift-and-assign the result is checked for FIXEDOVERFLOW and then assigned to the target with the specified precision. The assignment may cause the SIZE condition to be raised.

Error and Exceptional Conditions

The FIXEDOVERFLOW or the SIZE condition can occur in this module.

Linkage

R1 = A(argument)
R5 = A(DED for argument)
R6 = A(target)
R7 = A(DED for target)

Called By

IBMBMUDA:

IBMBMOD - ADD (fixed decimal).
IBMBMPV - MULTIPLY (complex fixed decimal).
IBMBMQV - DIVIDE (complex fixed decimal).

IBMBMUDB:

IBMBMOD - ADD (fixed decimal).

IBMBMVU - Multiplication and Division (Fixed Binary Complex)

Function

To calculate z_1*z_2 or z_1/z_2.
The module has two entry points:

IBMBMVUA: Multiplication

IBMBMVUB: Division

Method

An incorporated subroutine computes the expressions u*x + v*y and v*x - u*y to 63 bits in register pairs, where u, v, x, and y are given in registers to 31 bits.

Let z_1 = a + bI and z_2 = c + kI, z_1 having precision (p,q).

For multiplication,

$$REAL(z_1*z_2) = a*c - b*d$$

and

$$IMAG(z_1*z_2) = b*c + a*d$$

These expressions are computed by the subroutine; the results are then tested for fixed overflow and reduced to 31 bits each by a left shift.

For division,

$$REAL(z_1/z_2) = (a*c + b*d)/(c**2 + d**2)$$

and

$$IMAG(z_1/z_2) = (b*c - a*d)/(c**2 + d**2)$$

The subroutine is used to compute a*c + b*d and b*c - a*d. Each of these is then shifted left by (31 - p) bits. The expression c**2 + d**2 is next computed and, if truncation is necessary to reduce it to 31 significant bits, all three expressions are shifted right. Finally, the real and imaginary parts of the result are obtained by division.

Error and Exceptional Conditions

The FIXEDOVERFLOW condition can occur in either multiplication or division; the ZERODIVIDE condition can occur in division.

Linkage

R1 = A(z_1)
R5 = A(DED for z_1)
R6 = A(z_2)
R7 = A(DED for z_2)
R8 = A(DED for target)

Called By

Compiled code.

IBMBMVV - Multiplication and Division (Fixed Decimal Complex)

Function

To calculate z_1*z_2 or z_1/z_2.
The module has two entry points:

IBMBMVVA: Multiplication

IBMBMVVB: Division

Method

Let $z_1 = a + bI$ and $z_2 = c + dI$, with precisions (p,q) and (r,s) respectively. The results are computed as follows.

Multiplication:

$$\text{REAL}(z_1*z_2) = a*c - b*d$$

and $$\text{IMAG}(z_1*z_2) = b*c + a*d$$

Division:

$$\text{REAL}(z_1/z_2) = (a*c + b*d)/(c**2 + d**2)$$

and $$\text{IMAG}(z_1/z_2) = (b*c - a*d)/(c**2 + d**2)$$

For division, if the denominator has 15 significant digits or less, it is right-justified in an 8-byte field and the numerators are right-justified in 16-byte fields. If the denominator has more than 15 significant digits it is truncated on the right (t low order digits lost) to leave 15 significant digits. To obtain the required scale factor (15 - p + q - s) on division the numerators are shifted left by (15 - p - t) digits. The divisions are then performed. The real and imaginary parts of the result are each 8 bytes long (15 digits and a sign).

Error and Exceptional Conditions

The FIXEDOVERFLOW condition can occur in either multiplication or division; the ZERODIVIDE condition can occur in division.

Linkage

R1 = A(z_1)
R5 = A(DED for z_1)
R6 = A(z_2)
R7 = A(DED for z_2)
R8 = A(target)
R9 = A(DED for target)

Called By

Compiled code.

IBMBMVW - Multiplication (Float Complex)

Function

To calculate z_1*z_2, where $z_1 = a + bI$ and $z_2 = c + dI$. The module has two entry points:

IBMBMVWA: Short float complex arguments.

IBMBMVWB: Long float complex arguments.

Method

The real and imaginary parts of the result are computed as a*c - b*d and b*c + a*d respectively.

Linkage

R1 = $A(z_2)$
R5 = $A(z_1)$
R6 = A(target)

Called By

Compiled code.
IBMBMXW - Integer exponentiation (float complex).

IBMBMWX, IBMBMWY - Division (Float Complex)

Modules

Argument	Module
Short float	IBMBMWX
Long float	IBMBMWY

Function

To calculate z_1/z_2 in floating point, where $z_1 = a + bI$ and $z_2 = c + dI$.

Method

Case 1: ABS(c) ≥ ABS(d)

Compute q = d/c

Then $REAL(z_1/z_2) = (a + b*q)/(c + d*q)$

and $IMAG(z_1/z_2) = (b - a*q)/(c + d*q)$

Case 2: ABS(c) < ABS(d)

$(a + bI)/(c + dI) = (b - aI)/(d - cI)$, which reduces to case 1.

Error and Exceptional Conditions

The OVERFLOW, UNDERFLOW, and ZERODIVIDE conditions can occur in this module.

Linkage

R1 = $A(z_1)$
R5 = $A(z_2)$
R6 = A(target)

Called By

Compiled code.

IBMBMXS, IBMBMXL - Integer Exponentiation (Float Real)

Modules

Argument	Module
Short float	IBMBMXS
Long float	IBMBMXL

Function

To calculate x**n, where n is an integer between -2**31 and +2**31 - 1 inclusive.

Method

If the exponent is zero and the argument is non-zero, the result 1 is returned immediately. Otherwise, the result is set initially to the value of the argument, the exponent is made positive, and each significant bit, after the first, is tested in turn, from left to right. For each 1 bit, the contents of the target field are squared and then multiplied by x. For each significant 0 bit, the contents are squared only.

Finally if the exponent was originally negative the reciprocal of the result is taken; otherwise it is left unchanged.

Error and Exceptional Conditions

The ERROR condition is raised if $x = 0$ and $n \leq 0$. Since x**(-m), where m is a positive integer, is evaluated as 1/(x**m), the OVERFLOW condition may occur when m is large and the UNDERFLOW condition may occur when x is very small.

Linkage

```
R1    = A(PLIST)
PLIST = A(x)
        A(n)
        A(target)
```

Called By

Compiled code.

IBMBMXW - Integer Exponentiation (Float Complex)

Function

To calculate z**n, where n is an integer between -2**31 and 2**31 - 1 inclusive. IBMBMXW has two entry points:

IBMBMXWA: short float complex arguments.

IBMBMXWB: long float complex arguments.

Method

If the exponent is zero and the argument is non-zero, the result 1 is returned immediately.

If the exponent is non-zero, the contents of the target field are set to the argument value, the exponent is made positive, and each significant bit, after the first is tested in turn, from left to right. For each significant 0 bit, the target field contents are squared. For each 1 bit the contents are squared and then multiplied by z.

Multiplication is performed by a branch to the complex multiplication subroutine. Finally, if the exponent was originally negative the reciprocal of the result is taken; otherwise it is left unchanged.

Error and Exceptional Conditions

The ERROR condition is raised if

   z = 0 with n ≤ 0

Since x**(-m), where m is a positive integer, is evaluated as 1/(x**m), the OVERFLOW condition may occur when m is large and the UNDERFLOW condition may occur when x is very small.

The OVERFLOW or UNDERFLOW condition can occur in the called complex multiplication routine IBMBMVW.

Linkage

```
R1    = A(PLIST)
PLIST = A(z)
        A(n)
        A(target)
```

Calls

IBMBMXW calls:

IBMBMVW - Multiplication (short and long float complex).

Called By

Compiled code.

IBMBMYS, IBMBMYL - General Exponentiation (Float Real)

Modules

Argument	Module
Short float	IBMBMYS
Long float	IBMBMYL

Function

To calculate x**y, where x and y are real floating point numbers.

Method

When x = 0, the result x**y = 0 is given if y > 0, and an error message is given if y ≤ 0.

When x ≠ 0 and y = 0, the result x**y = 1 is given.

In all other cases x**y is computed as EXP(y*LOG(x)).

Error and Exceptional Conditions

The ERROR condition is raised in this module if

x = 0 with y ≤ 0

The OVERFLOW condition occurs in the called real EXP routine IBMBMBS (short) or IBMBMBL (long) if

y*LOG(x) > 174.673

The ERROR condition is raised in the called real LOG routine IBMBMDS (short or IBMBMDL (long) if

x < 0 with y ≠ 0

Linkage

```
R1    = A(PLIST)
PLIST = A(y)
        A(x)
        A(target)
```

Calls

IBMBMYS calls:

IBMBMDS - LOG (short float real).
IBMBMBS - EXP (short float real).

IBMBMYL calls:

IBMBMDL - LOG (long float real).
IBMBMBL - EXP (long float real).

Called By

Compiled code.

IBMBMYX, IBMBMYY - General Exponentiation (Float Complex)

Modules

Argument	Module
short float	IBMBMYX
long float	IBMBMYY

Function

To calculate z_1**z_2, where z_1 and z_2 are complex numbers of the same precision.

Method

When $z_1 = 0$, the result 0 is returned if REAL(z_2)>0 and IMAG(z_2) = 0, otherwise invalid exponentiation of zero. Otherwise, z_1**z_2 is computed as

EXP(z_2*LOG(z_1))

with the proviso that if IMAG(z_1) = 0 then LOG(ABS(z_1)) is calculated by a call to the real LOG routine, not to the complex LOG routine.

Error and Exceptional Conditions

The ERROR condition is raised in this module if $z_1 = 0$ with either REAL(z_2) $\le$ 0 or IMAG(z_2) $\ne$ 0.

The OVERFLOW condition occurs in the called EXP routine if REAL(z_2*LOG(z_1)) > 174.673

The ERROR condition is raised in the called LOG routine if

ABS(IMAG(z_2*LOG(z_1))) $\ge$ 2**18*pi (short float)
2**50*pi (long float)

Linkage

```
R1    = A(PLIST)
PLIST = A(z2)
        A(z1)
        A(target)
```

Calls

IBMBMYX calls:

IBMBMBX - EXP (short float complex).
IBMBMDX - LOG (short float complex).

IBMBMYY calls:

IBMBMBY - EXP (long float complex).
IBMBMDY - LOG (long float complex).

In the special case where IMAG(z_1) = 0, then calls are made to the real floating point LOG routines IBMBMDS (short) and IBMBMDL (long), not to the complex LOG routines shown above.

Called By

Compiled code.

CHAPTER 8: INTERLANGUAGE COMMUNICATION MODULES

The interlanguage communication modules of the resident library support |communication between routines written in PL/I and routines written in |COBOL or FORTRAN, and from RPGII to PL/I. Communication between COBOL, FORTRAN, and RPGII is not supported.

The resident library contains three interlanguage communication modules which handle the environmental changes necessary when going from PL/I to |non-PL/I routines and vice versa. These modules are IBMDIEC (PL/I calls COBOL), IBMDIEF (PL/I calls FORTRAN), and IBMDIEP (COBOL, FORTRAN, or RPGII calls PL/I). The key to environmental switching is a CSECT (control section) called IBMBILC1. This CSECT is included in each of the modules, and one copy of it is processed by the linkage editor whenever interlanguage communication is required.

IBMBILC1 contains two fullwords, both of which are initially zero. The first word is used to point to ZCTL, a control block which is set up on the first occasion that one of the interlanguage routines is called. |The second word contains flags that are set to indicate COBOL, FORTRAN, |or RPGII on the first occasion that calls are made to or from these languages.

Full details of the interlanguage communication control blocks are given in DOS PL/I Optimizing Compiler: Execution Logic.

MODULE DESCRIPTIONS

IBMDIEC - Interlanguage Housekeeping

Function

To provide the housekeeping necessary when a PL/I procedure calls a COBOL subprogram. The module has three entry points:

IBMBIECA: Establish standard COBOL environment.

IBMBIECB: Establish COBOL environment with provision for PL/I interrupt handling.

IBMBIECC: Re-establish the PL/I environment after return from a COBOL subprogram.

Method (chart DIEC)

Entry point IBMBIECA:

This entry point is used to establish the COBOL environment when PL/I calls a COBOL entry point for which the INTER option has not been specified.

If the interlanguage control block ZCTL has not already been set up, that is, if this is the first interlanguage call, IBMDIEC acquires non-LIFO storage by a call to IBMDPGR and initializes ZCTL. The address of

ZCTL is stored in the first word of IBMBILC1 and the contents of register R12 are stored in ZCTL.

The module then acquires an interlanguage VDA(in LIFO storage) for this particular call to COBOL. The new VDA is inserted in the interlanguage VDA chain between the previous VDA and ZCTL. The contents of register 13, which were saved on entry to the module, are then stored in the new VDA.

To enable any interrupts which occur in the COBOL subprogram to be handled by the supervisor, the module issues a STXIT macro with null arguments to cancel the previously issued STXIT, and sets the program mask to zero.

Finally, control is returned to PL/I compiled code, which subsequently branches to the COBOL subprogram.

Entry Point IBMBIECB:

This entry point is used to establish the COBOL environment when PL/I calls a COBOL entry point for which the INTER option has been specified.

When entered at this entry point, the module sets up ZCTL if necessary and acquires a VDA in the same way as has been described for entry point IBMBIECA. Before returning to compiled code, however, IBMDIEC issues a STXIT macro specifying an address (IE007) within the module. Interrupts in the COBOL subprogram thus cause module IBMDIEC to be entered at address IE007.

Entry Address IE007:

If control reaches address IE007 the following action is taken.

The PL/I environment is re-established by restoring register 13 from the interlanguage VDA and register 12 from ZCTL. The "return address" part of the PSW passed by the supervisor is stored in library workspace and replaced by the address (IE015) of a further point in IBMDIEC. Also, if the interrupt is due to UNDERFLOW, the last four bytes of the COBOL program, up to the point of interrupt, are copied to IE013 so that the error-handling module IBMDERR can take the correct action. This discrimination is made because, if the interrupt is due to a bad value in the IAR, copying might cause a further addressing interrupt.

A STXIT macro is issued to restore normal PL/I interrupt handling, and a branch is made, via the bootstrap in the TCA appendage, to entry point IBMBERRA of the error handling module. The interrupt is subsequently handled in the normal way, except that any return to the point of interrupt is made to IE015 in IBMDIECA.

Entry Addresses IE015 and IE016:

The module is entered at address IE015 when the PL/I interrupt handler IBMDERR returns control to the point that it takes to be the point of interrupt, or when a GOTO out of an on-unit for the interrupt is executed.

For the GOTO case, the module dechains and frees the latest language VDA, and calls IBMDERR to print out a warning message. Control is then returned to the caller.

Otherwise a STXIT macro is issued specifying an address (IE016) in the module. An interrupt is then forced, and the module is re-entered at IE016.

At IE016, the address from the PSW of the original interrupt is moved from library workspace into the PSW passed by the supervisor for this interrupt, and the program check exit address is reset to IE007 by a STXIT macro. The COBOL registers are then restored and an EXIT macro is issued. The supervisor subsequently returns control to the point at which the original interrupt occured.

Entry Point IBMBIECC:

This entry point is used to re-establish the PL/I environment after control has been returned to PL/I from a COBOL program.

A STXIT macro is issued to restore normal PL/I interrupt handling, and the program mask is restored to its PL/I setting (hexadecimal E). The VDA is then dechained and control is returned to compiled code.

Linkage

No parameters are passed to this module.

Calls

IBMDPGR - Storage management.
IBMDERR - Error handler.

Called By

Compiled code.

IBMDIEF - Interlanguage Housekeeping

Function

To provide the housekeeping necessary when a PL/I program calls a FORTRAN subprogram. The module has four entry points:

IBMBIEFA: Establish standard FORTRAN environment.

IBMBIEFB: Establish FORTRAN environment and enable interrupts not handled by FORTRAN to be handled by PL/I.

IBMBIEFC: Re-establish the PL/I environment after return from a FORTRAN subprogram.

IBMBIEFD: Re-establish the PL/I environment and store the returned value after returning from the invocation of a FORTRAN function.

Method (chart DIEF)

Entry point IBMBIEFA:

This entry point is used to establish the FORTRAN environment when PL/I calls a FORTRAN entry point for which the INTER option has not been specified.

If the interlanguage control block ZCTL has not already been set up,i.e. if the COBOL and FORTRAN flags are both off, IBMBIEF acquires non-LIFO storage by a call to IBMDPGR and initializes ZCTL. The address of ZCTL is stored in the first word of IBMBILC1 and the contents of register R12 are stored in ZCTL. If the FORTRAN flag is off, IBMDIEF calls the Fortran library to initialize the Fortran environment.

The module then acquires an interlanguage VDA(in LIFO storage) for this particular call to FORTRAN. The new VDA is inserted in the interlanguage VDA chain between the previous VDA and ZCTL. The contents of register 13, which were saved on entry to the module, are then stored in the new VDA. The FORTRAN environment is set up by issuing a STXIT macro specifying the FORTRAN interrupt information held in ZCTL.

Finally, control is returned to PL/I compiled code, which subsequently branches to the FORTRAN subprogram.

Entry Point IBMBIEFB:

This entry point is used to establish the FORTRAN environment when PL/I calls a FORTRAN entry point for which the INTER option has been specified.

When entered at this entry point, the module sets up ZCTL if necessary and acquires a VDA in the same way as has been described for entry point IBMBIEFA. Before returning to compiled code, however, IBMBIEF issues a STXIT macro specifying an address (IE023) within the module. Interrupts in the FORTRAN subprogram thus cause module IBMDIEF to be entered at address IE023.

Entry Address IE023:

If control reaches address IE023 the module determines the cause of the interrupt. If the interrupt condition is one that is not handled by FORTRAN, the following action is taken:

The PL/I environment is re-established by restoring register 12 from ZCTL and and register 13 from the interlanguage VDA. The "return address" part of the PSW passed by the supervisor is stored in library workspace and replaced by the address (IE034) of a further point in IBMDIEF.

A STXIT macro is issued to restore normal PL/I interrupt handling, and a branch is made to entry point IBMBERRA of the error handling module. The interrupt is subsequently handled in the normal way, except that any return to the point of interrupt is made to IE034 in IBMDIEF.

If the interrupt is one that is handled by FORTRAN, the FORTRAN environment is set up by copying the interrupted registers and the PSW into the FORTRAN interrupt savearea and issuing a STXIT macro instruction with the correct FORTRAN parameters. An interrupt is then forced, so that the FORTRAN interrupt handler is entered. The FORTRAN interrupt handler returns via a bootstrap in library workspace to address IE030 in IBMDIEF.

Entry Addresses IE034 and IE030:

The module is entered at address IE034 when the PL/I interrupt handler IBMDERR returns control to the point that it takes to be the point of interrupt, or at address IE030 when the FORTRAN interrupt handler returns control via the bootstrap in library workspace. In both cases, the general action taken is the same.

If a GOTO out of an on-unit has been taken, the module dechains and frees the latest language VDA and then returns to the caller.

Otherwise, a STXIT macro is issued specifying a further point (IE038) in the module. An interrupt is then forced, and the module is re-entered at IE038.

The interrupt address is now reset to IE023 by means of a STXIT macro. The address from the PSW of the original interrupt is moved from library workspace into the PSW passed by the supervisor for this interrupt. The FORTRAN registers are then restored and an EXIT macro is issued. The supervisor subsequently returns control to the point at which the original interrupt occured.

Entry Point IBMBIEFC:

This entry point is used to re-establish the PL/I environment after control has been returned to PL/I from a FORTRAN subprogram.

The program mask is reset and a STXIT macro is issued to restore normal PL/I interrupt handling. The current VDA is then dechained and control is returned to compiled code.

Entry Point IBMBIEFD:

This entry point is used to re-establish the PL/I environment after control has been returned from the invocation of a FORTRAN function. The processing is the same as that for entry point IBMBIEFC except that, having released the VDA, the module examines the parameter list and moves the returned value from the register, or registers, in which FORTRAN left it to the required location.

Linkage

```
R1    = A(PLIST)
PLIST = A(value to be returned or locator for value to be returned)
        A(DED for returned value)
```

Calls

IBMDPGR - Storage management.
IBMDERR - Error handler.
IBCOM# - FORTRAN initialisation and termination.

Called By

Compiled code.

IBMDIEP - Interlanguage Housekeeping

Function

To provide the housekeeping necessary when a PL/I procedure is called from another language. The module has four entry points:

IBMBIEPA: Establish the PL/I environment and get a DSA for a PL/I |procedure which has been entered at an entry point with OPTIONS(COBOL), |OPTIONS(FORTRAN), or OPTIONS(RPG).

IBMBIEPB: Set up a parameter list specifying the length and location of the PL/I ISA(initial storage area).

IBMBIEPC: Re-establish the environment of the non-PL/I calling program.

IBMBIEPD: Re-establish the environment of the non-PL/I calling program and move the value returned by the PL/I function into the appropriate register.

Method (chart DIEP)

Entry Point IBMBIEPA:

The module is entered at this entry point after a PL/I procedure has |been invoked at an entry point with OPTIONS(COBOL), OPTIONS(FORTRAN), or |OPTIONS(RPG).

|The module first tests the language flags in the interlanguage control |block IBMBILC1. If any flag is set, this means that a previous interlanguage call has been made and consequently that a PL/I ISA(initial storage area) has already been acquired and initialized. In these circumstances IBMBIEP has only to restore the PL/I TCA pointer (register 12), establish PL/I interrupt handling, set the appropriate flag in IBMBILC1, and obtain a DSA for the PL/I procedure.

Register 12 is restored from the interlanguage control block ZCTL. The interrupt handling data for the non-PL/I calling procedure is stored in the the latest interlanguge VDA, and a STXIT macro is issued to establish PL/I interrupt handling. A DSA is obtained, by a call to IBMDPGR if necessary, and its address is loaded into register 13. Finally, control is returned to PL/I compiled code.

|If none of the COBOL, FORTRAN, or RPGII flags in IBMBILC1 are set, the PL/I ISA must be initialized. The non-PL/I user may have already allocated an ISA by means of a call to IBMBIEPB (alias PLISA). If this is so, the stack flag in IBMBILC1 will be set. If the stack flag is not set IBMBIEP finds the length and the address of the available ISA from the information in the job's communication region, leaving space for ZCTL and a DTF and a buffer for FORTRAN error messages.

In either case, IBMDIEP sets up ZCTL. The interrupt handling data for the non-PL/I user is then stored in ZCTL.

The length and the address of the ISA are then passed as parameters to |entry point IBMBPJRB of the program initialization module IBMDPJR. Also passed to IBMBPJRB is an address in IBMDIEP. The initialization module takes this address to be that of the PL/I program; consequently, after the ISA has been initialized, control is returned to IBMDIEP.

|On return from IBMBPJRB, the DSA chain is changed so that the PL/I dummy DSA is above the DSA of the non-PL/I calling program. This is done so that PL/I program initialization does not take place every time a PL/I procedure is called from the same level of non-PL/I (see "Tail" Code

below).

IBMBIEP then acquires an interlanguage VDA and stores its address in the first word of ZCTL. The caller's error handling information and the contents of register 13 are saved in the new VDA.

Finally, a DSA is obtained for the PL/I procedure. Control is then returned to PL/I compiled code.

Entry point IBMBIEPB:

This entry point has an alias, PLISA, which is called by COBOL or FORTRAN to specify the area of storage that the PL/I procedure is to use as its ISA.

IBMBIEP takes part of the specified area for ZCTL and loads the address and length of the remainder into the parameter list for IBMBPIR. The stack flag in IBMBILC1 is set to indicate that an ISA allocation has | been made, and control is returned to the caller.

Note: If only one argument is passed to PLISA, the stack flag is reset to zero; the default ISA is thus obtained when the module is subsequently entered at IBMBIEPA.

Entry Point IBMBIEPC:

The module is entered at this entry point at the end of a PL/I procedure, immediately before control is returned to the calling program. The calling program's interrupt handling requirements are re-established.

Entry Point IBMBIEPD:

This entry has the same purpose as entry point IBMBIEPC, but also moves a value that is to be returned to FORTRAN into the appropriate registers.

Note: Floating point results may cover one, two, or four floating point registers. Fixed point and bit results are always returned in Register 0.

Tail Code:

When the non-PL/I program that called PL/I is returning to its own caller, it restores registers from the DSA prior to its own and returns to the address specified in this DSA. However, because the DSAs have been rechained (see above), this DSA is in fact the short save area in ZCTL. The address that control is returned to is thus that of point | IE014 in module IBMDIEP. This address is the start of the "tail" code.

The tail code passes the FORTRAN return code and value, if these exist, past the dummy DSA to the non-PL/I calling program's caller. It then restores registers from the dummy DSA and returns to module IBMDPIR. The return code field (TORC) in the TCA is set so that IBMDPIR will ignore any ON FINISH units and execute a normal return to the caller, i.e., to IBMDIEP.

| IBMDPIR returns control to point IE002 in the tail code. The tail code then tests the return code from IBMDPIR. If the return code is zero, the tail code clears the interlanguage flags (except the stack flag) and returns to the non-PL/I calling program's caller. If, however, the return code indicates that a PL/I STOP instruction has been executed, a test is made to determine whether or not there is a FORTRAN environment.

If there is, the equivalent of a FORTRAN library STOP command is used to close any FORTRAN files and to terminate the program. Otherwise, an EOJ macro is issued.

Entry point IE103 exists to intercept RPGII termination, since RPG issues its own EOJ macro without returning to its caller (since there normally is none). This re-routes control through the tail code, and ensures that, on return from IE002, control will return to the RPGII termination code.

Linkage

Entry Point IBMBIEPA:

```
R0 = length of required DSA
R1 = calling language code
     (0 for COBOL,
       4 for FORTRAN)
```

Entry Point IBMBIEPB:

```
R1    = A(PLIST)
PLIST = A(area for PL/I ISA)
        A(length of PL/I ISA) or X'80'
```

Entry Point IBMBIEPD:

```
R1    = A(PLIST)
PLIST = A(value to be returned or locator for value to be returned)
        A(DED for value to be returned)
```

No arguments are passed to entry point IBMBIEPC.

Calls

IBMDPGR - Storage management.
IBMDPIR - Program initialization.
IBCOM# - FORTRAN termination.

Called By

Compiled code.

CHAPTER 9: MISCELLANEOUS ROUTINES

The modules described in this chapter support the PL/I built-in functions DATE and TIME and the PL/I statements DISPLAY, WAIT, and DELAY, and provide interfaces with the system CHECKPOINT/RESTART and SORT utilities.

Two display modules are provided, IBMDJDS and IBMDJDZ. Except in the case of a program which contains a DISPLAY with EVENT statement, either module will suffice. In the case of the above exception however, module IBMDJDS is used. Three WAIT modules are described. IBMDJWT is the main WAIT module that handles PL/I WAIT statements in systems that support the WAITM macro. It is supported by module IBMBJWI, which is called whenever the WAIT statement specifies an array event variable. IBMBJWI converts any array into a list of scalar event variables, and then passes the list to IBMDJWT.

In systems that do not support the WAITM macro, neither of these modules is included. Their place is taken by the single WAIT module IBMGJWT, which permits waiting on one event only.

MODULE DESCRIPTIONS

IBMDJDS - DISPLAY

Function

To implement the PL/I DISPLAY statement. The module has two entry points:

IBMBJDSA: Normal entry from compiled code.

IBMBJDSB: Entry from the WAIT module (IBMDJWT or IBMGJWT).

Method (chart DJDS)

Entry Point IBMBJDSA:

Module IBMDJDS includes a channel control block (CCB) and a number of channel command words (CCWs). Four CCWs, occupying four contiguous double-word locations, are used to display the message, to put out a standard message "AWAITING REPLY", to read the reply, and to sense the result.

On entry to the module, surplus blanks are removed from the display message. The module then examines the parameter list to determine which of the three possible types of display statement (DISPLAY alone, DISPLAY with the REPLY option, and DISPLAY with the REPLY and EVENT options) is being processed. The action taken in each case is as described below:

DISPLAY: For display alone, only the first CCW is required. The module therefore dechains the display CCW from the "awaiting reply" CCW, moves the address of the display message, together with the required channel command code, into the CCW, and calls the internal "console I/O" subroutine (see below). On return from this subroutine, the module returns control to compiled code.

DISPLAY with REPLY option: If the REPLY option is specified, all four CCWs are required. The module therefore chains the display CCW to the "awaiting reply" CCW. (This CCW is permanently chained to the reply CCW, which, in turn, is permanently chained to the sense CCW). The module then moves the addresses of the display and reply strings, together with the required channel command codes, into the CCWs and calls the internal "console I/O" subroutine. On return from this subroutine, the module returns control to compiled code.

DISPLAY with REPLY and EVENT options: If the EVENT option is present, the module chains and initializes the CCWs as described in the preceding paragraph. It then flags the specified event variable as an active I/O display event and sets a "nowait" switch to indicate to the internal "console I/O" subroutine that the EVENT option is present. Because the EVENT option means that control will be returned to compiled code as soon as the channel program has been initiated, the copies of the CCB and the CCWs that are contained in IBMDJDS cannot be used. The module therefore acquires non-LIFC storage and copies the CCB and the CCWs into it. The display string is also copied into this storage, in case the contents of the original display string are altered by compiled code before the message has been displayed. The address of the new CCB is set into the event variable. The module then calls the internal "console I/O" subroutine (see below). On return from this subroutine, control is returned to compiled code.

Entry Point IBMBJDSB:

This entry point is called by the WAIT module (IBMDJWT or IBMGJWT) when the display event variable has been specified in a PL/I WAIT statement. The module is passed the address of the relevant CCB.

On entry, the module calls the internal "console I/O" subroutine at entry point JD600 (see below) to test the success or otherwise of the DISPLAY REPLY statement. On return from this subroutine, IBMDJDS frees the storage that was obtained for the CCB, the CCWs, and the display string and then returns control to the caller (IBMDJWT).

Console I/O Subroutine:

This subroutine issues an EXCP macro to execute the channel program defined by the CCWs that have been set up by the main code of the module.

It then tests the "nowait" switch to determine whether or not a wait is required. If the nowait switch is on, i.e., if the EVENT option was present in the display statement, the subroutine returns immediately to the main code of the module, which returns control to compiled code (see above). If the nowait switch is off, the subroutine issues a WAIT macro.

When channel end is received, the subroutine proceeds to check whether or not any errors have occurred by examining the sense byte. This point in the subroutine (JD600) is the entry point called by the main code of the module when it is entered at entry point IBMBJDSB.

If there have been no unit errors ("unit check" or "unit exception") the module returns to the caller.

If a unit error has occurred, the module blanks out the reply string and reissues the EXCP macro instruction for the display. However, if the error is a unit check that requires operator intervention (detected by examining the sense byte), the audible alarm is sounded by means of a further EXCP macro instruction before the display is retried.

Error and Exceptional Conditions

The ERROR condition is raised if an active event variable is specified.

Linkage

```
R1    = A(PLIST)
PLIST = A(string locator for message)
        A(string locator of space for reply) or zero
        A(event variable) or zero
```

Called By

```
Compiled code (entry point IBMDJDSA).
WAIT module   (entry point IBMDJDSB).
```

IBMDJDZ - DISPLAY without EVENT

Function

To support the PL/I DISPLAY statement and REPLY option.

Method

The display string is scanned and trailing blanks removed. If there is no REPLY option, an EXCP and WAIT macro are issued to transmit the message to the console. The CCB used contains the address and length of the message. Return is made to the caller.

If there is a REPLY option, the EXCP and WAIT are issued as for DISPLAY only. However, the EXCP specifies a chain of three CCBs. The first is to transmit the display string, the second to transmit a standard message to inform the operator that a reply is required, and the third to read the reply. When the reply is received, return is made to the caller.

If a unit check or exception occurs on the console, the reply string, if any, is blanked out, and the EXCP and WAIT for the DISPLAY/REPLY are reissued.

IBMDJDZ raises ERROR if there is a zero length DISPLAY with REPLY option, or if the length of the string to accept a reply is zero.

Linkage

```
R1 -> A(string locator of message)
      A(string locator of space for reply)(optional)
```

Called By

Compiled code.

IBMDJDT - DATE Built-In Function

Function

To implement the PL/I built-in function DATE, i.e. to return the date in the form yymmdd.

Method (chart DJDT)

The module accesses the communication region, which contains the date in bytes 0 thru 7. The format of the date (MM/DD/YY or DD/MM/YY) is found by testing the data configuration bit in the communication region, the result of this test being used to select the appropriate format translation table. The date is then translated to the target format YYMMDD.

Linkage

R1 = A(PLIST)
PLIST = A(6-byte field to receive answer)

Called By

Compiled code.

IBMDJDY - DELAY

Function

To implement the PL/I statement DELAY, i.e to suspend the execution of the task for a time period specified in milliseconds.

Method (chart DJDY)

If the system has no interval timer, an immediate return is made to the caller.

Otherwise, the required delay is divided by 1000 to convert it to seconds and is rounded to the nearest second. The result is given as a parameter to the SETIME macro specifying a Timer Event Control Block (TECB) and a WAIT macro is issued on the TECB. On completion of the WAIT a return is made to compiled code.

Note: Because the timer interval for DOS is one second, the DELAY statement is implemented only to the nearest second.

Linkage

R1 = A(required delay in milliseconds)

Called By

Compiled code.

IBMDJTT - TIME Built-In Function

Function

To implement the PL/I built-in function TIME, i.e. to return the time of day in the character format HHMMSSTTT.

Method (chart DJTT)

The module issues a GETIME macro with the TU option, which returns the time of day in units of 1/300 second. This is converted into hours, minutes, seconds, and thousandths of a second and assembled into the required character format.

If the system has no time-of-day facility, the module returns the character string '000000000'.

Linkage

```
R1    = A(PLIST)
PLIST = A(9-byte field to receive answer)
```

Called By

Compiled code.

IBMBJWI - WAIT (Array Events)

Function

To implement the PL/I WAIT statement when array events have been specified.

Method (chart BJWI)

The parameter list is scanned to compute the total number of event variables, and a VDA is obtained for the parameter list that will be passed to IBMDJWT.

The first word of the parameter list passed to IBMBJWI is copied into the VDA. Each pair of two words in the parameter list is then examined again. If the dimensionality is zero, the event variable address is copied into the VDA. If the dimensionality is one, a simple loop is executed to compute the address of each element in the single-dimensioned array and place it in the VDA. If the dimensionality (N) is greater than one, a VDA of length 12*(N-1) is obtained and passed to module IBMBAIH. This module sets up a table in the VDA which enables IBMBJWI to index through the array and place the address of each element in the VDA.

When the parameter list is complete, IBMDJWT is called to perform the wait. Return is then made to the caller.

Linkage

```
R1    = A(PLIST)
PLIST = A(number of events (N) to be waited on) or high order
          bit = 1 if N is not specified.
        A(event variable) or A(array locator)
        Dimensionality
        .
        .
        .
        .
        A(event variable) or A(array locator)
        Dimensionality
```

The end of the parameter list is denoted by a high order '1' bit in the dimensionality word.

Calls

IBMDPGR - Storage management.
IBMBAIH - Array indexer.
IBMDJWT - WAIT module.

Called By

Compiled code.

IBMDJWT - WAIT (Multiple Events)

Function

To implement the PL/I WAIT statement.

Method (chart DJWT)

The module first scans the list of Event Variables (EVs) and calculates the number of events which must be completed to satisy the WAIT. If this number is not positive, i.e. there are already enough completed events or the number of events to be waited on was less than one, control is returned to the caller.

Note: EWIP, which is tested at statement JW003, is a flag in the EV which indicates that the event is already being waited on and has caused entry to an ON-unit. This situation can arise if a wait on a particular event causes entry to an ON-unit which contains a wait on the same event.

The module then acquires a VDA for the EVTAB (event table) and the ECBLIST (list of event control block addresses). If all the events to be waited on are uncompleted active I/O events (other than DISPLAY events), an ECBLIST is not required; in these circumstances the length of the ECBLIST is set to zero before storage is acquired and flag F is set on.

The parameter list is then scanned again. Events which are complete or have the EWIP flag set are ignored. For other events an EVTAB element is created in the EVTAB and chained to the EV. If flag F is off and the EV is associated with an active I/O event, the address of the ECB is obtained from the EV and placed in the ECBLIST. The address of this ECBLIST element is placed in the EVTAB element for the EV.

Subsequent action depends on whether or not flag F is set.

Flag F Set:

Flag F indicates that all the events to be waited on are uncompleted active I/O events, all of which must be completed to satisfy the WAIT. In these circumstances, the events are not waited on in this module; instead, the module scans the EVTAB and passes the address of each active EV in turn to an internal "check event" subroutine (JW300), which calls IBMDRIC. The check event subroutine is described below.

Flag F Not Set:

If flag F is not set, the events are waited on in this module before being passed to IBMDRIC.

A check is first made to see whether or not an ECBLIST exists. If one does not, all the events being waited on are either incomplete non-I/O events or have the EWIP flag set. This means that no more events can be completed and the wait is still not satisfied. In these circumstances the module dechains the EVTAB elements and branches the the error handler (IBMDERR) to raise the ERROR condition.

If an ECBLIST does exist, a WAITM macro is issued with the ECBLIST as parameter. Control returns when any one of the ECBs addressed from the ECBLIST is complete. The EVTAB is then scanned to find any EV whose ECB is now complete, and the internal check event subroutine is called (see below). If control returns from the subroutine, the ECBLIST is checked again and a further WAIT macro is issued.

Check Event Subroutine (JW300)

The check event subroutine calls IBMDRIC to wait on I/O events, or IBMDJDS for DISPLAY events. When control returns, the EVTAB element is set inactive and complete.

Note: An EVTAB element is set inactive by setting the field that contains the address of the associated EV to zero.

If a chain of EVTAB elements exists for the EV, each EVTAB element in the chain is then set complete and inactive.

When the end of the chain of EVTAB elements is reached, the associated EV is set complete and inactive. The module then scans the EVTAB to see whether or not there are any other completed events. This is necessary because completion of the first event may have caused entry to an ON-unit in which other events were set complete, either by means of the the completion pseudovariable or by further WAIT statements.

The number of events to be waited on is decremented by one each time a complete active event is found. When this number is reduced to zero control is returned to the caller.

Error and Exceptional Conditions

The ERROR condition is raised if, at any time during the execution of and before the completion of the wait, the number of active incomplete events is reduced to zero. The type of error raised depends on the why the remaining events cannot be completed. If there are any inactive and incomplete events, the error code passed to IBMDERR indicates that the WAIT would cause a permanent wait. If there are no inactive, incomplete events, the error code indicates that the WAIT cannot be satisfied owing to EVs having the EWIP flag set.

Linkage

```
R1    = A(PLIST)
PLIST = A(number of events (N) to be waited on) or high order
          bit = 1 if N is not specified.
        A(event variable)
        .
        .
        .
        A(event variable)
```

The last A(EV) in the parameter list is indicated by a '1' in the high order bit.

Calls

IBMDRIO - Record I/O module.
IBMDERR - Error handler.
IBMDJDS - DISPLAY.

Called By

Compiled code.
IBMBJWI - WAIT (array events).

IBMGJWT - WAIT (Single Event)

Function

To implement the PL/I WAIT statement. This module is used in systems that do not support waits on multiple events.

Method (chart GJWT)

The module first checks the parameter list to confirm that only one event variable (EV) has been passed. If more than one EV has been passed, the module calls the error handler to raise the ERROR condition.

If the EV is already complete, or if the number of events to be waited on (specified in the PL/I WAIT statement) is less than or equal to zero, an immediate return is made to compiled code.

The module now tests to see whether or not the event can be waited on. If the EV is not active, or if flag EWIP in the EV is set, the module calls the error handler to raise the ERROR condition. Otherwise, the module goes ahead and processes the WAIT.

For I/O events, the module passes the EV to IBMDRIO, which in turn calls the relevant transmitter module to wait on the event and raise any errors (see DOS PL/I Transient Library: Program Logic).

For DISPLAY events, the module issues a WAIT macro on the CCB and, on completion of the wait, calls IBMDJDS to tidy up the DISPLAY environment.

If control returns from IBMDRIO or IBMDJDS, the module sets the EV complete and inactive and returns to the caller.

Error and Exceptional Conditions

The error handler is called to raise the ERROR condition in the following circumstances:

1. More than one EV is passed to the module.

2. The EV passed to the module is incomplete and inactive.

3. The EV passed to the module has its EWIP flag set.

Note: EWIP is a flag in the EV which indicates that the event is already being waited on and has caused entry to an ON-unit. This situation can arise if a wait on a particular event causes entry to an ON-unit which contains a wait on the same event.

Linkage

```
R1    = A(PLIST)
PLIST = A(number of events (N) to be waited on) or high order
          bit = 1 if N is not specified.
        A(event variable)
        .
        .
        .
        A(event variable)
```

The last A(EV) in the parameter list is indicated by a '1' in the high order bit.

Calls

IBMDRIO - RECORD I/O interface module.
IBMDJDS - DISPLAY module.
IBMDERR - Error handler.

Called By

Compiled code.

IBMDKCP - CHECKPOINT/RESTART Interface Module

Function

To implement the PL/I CHECKPOINT facility.

Method (chart DKCP)

The module establishes a checkpoint by issuing a CHKPT macro instruction. Operands for the macro instruction are set up as follows:

If the device number in the parameter list for IBMDKCP (see "Linkage" below) specifies a disk file, a DTFPH is generated and its address is set as an operand on the CHKPT macro instruction. The device number is contained in the "device characteristic" parameter, which has the form "[sysname],[device number]". The default values for the sysname and the device number are SYS001 and 2400, respectively.

The "restart address" set on the CHKPT macro instruction is an address within IBMDKCP. The purpose of this entry is described under the heading "Restart" below.

Two tables are set up and their addresses are set as operands on the CHKPT macro instruction. The first table contains the name of all tape files that are open when the checkpoint is taken. The second contains verification information for all disk files that are open when the checkpoint is taken. Information on the open files is obtained by scanning the open file chain.

The "checkid" parameter in the parameter list for IBMDKCP (see "Linkage" below) is a character string variable. The module places a unique checkpoint identification in this variable when the CHKPT macro instruction has been issued.

After issuing the CHKPT macro instruction, the module issues a STXIT macro instruction to restore PL/I interrupt handling, and then returns to the caller. The STXIT macro instruction is necessary because the CHKPT macro instruction cancels the previously issued STXIT.

Restart

When restart is invoked by means of a JCL statement, the restart routine passes control to the address in IBMDKCP that was specified in the CHKPT macro instruction. The module resets the program mask to the value it had when the checkpoint was established, and issues a STXIT macro instruction to restore PL/I error handling. The module then returns to the point in compiled code at which the checkpoint was taken.

Error and Exceptional Conditions

Errors are handled by passing a fullword binary return code to the caller in the address specified in the parameter list. The return codes have the following meanings:

0	Successful completion
4	Restart has occurred
8	Unsuccessful completion (program error)
12	Unsuccessful completion (I/O error or set-up error)

Linkage

```
RA     = A(PLIST)
PLIST = A(filename string locator)
        A(checkid return string locator)
        A(device characteristic string locator)
        A(return code)
```

Note: All the PLIST arguments are optional.

Called By

Compiled code.

IBMDKDM - Dump Bootstrap

Function

|IBMDKDM is the bootstrap routine for the transient dump control module IBMDKMR, or IBMFKMR in a CICS environment.

Method (chart DKDM)

|The module tests a field in the TCA to determine if the program is |executing on CICS. If so, a CICS program control LOAD macro is issued |for the CICS dump control module IBMFKMR, otherwise the module loads the |transient dump control module IBMDKMR. The appropriate module is then |called, passing the same parameter list as was passed to IBMDKDM.

On return from the transient routine, IBMDKDM decides, from the |parameter returned by IBMDKMR or IBMFKMR, which of the two possible actions (stop or continue) is required. For "stop", IBMDKDM places the appropriate return code in the TCA and branches to the goto code in the TCA, passing the address of the dummy DSA and the address of register 14 in that DSA. For "continue", a normal return is made to the caller.

Linkage

```
R1     = A(PLIST)
PLIST = A(string locator for dump parameter string)
        A(string locator for the user's dump-identifier)
```

Calls

IBMDKMR - Transient dump control module.
|IBMFKMR - Transient dump control module - CICS.

Called By

Compiled code.

IBMDKST - SORT Interface Module

Function

IBMDKST acts as an interface between the system sort/merge facilities and PL/I. In this capacity it serves as a bootstrap to preserve the integrity of the PL/I calling program and to set up the correct environment for the sort/merge routines. It interfaces with either of two SORT/MERGE programs; No. 5736-SMI., or No. 360-SM-483.

The module has four entry points and enables the use of two SORT/MERGE user exit points E15 and E35. The exit points allow a PL/I routine to create and pass records to the SORT/MERGE routine and also to receive records after sorting from the routine.

IBMDKST can be called by any of its four entry points to establish four types of processing.

IBMBKSTA: Invokes the sort/merge program to sort records from an input data set, and write them in sorted sequence onto an output data set.

IBMBKSTB: Invokes the sort/merge program and specifies the use of user exit E15. Using this exit can either supply all the records to be sorted or receive records obtained from an input data set and alter or delete them from the sort. It can also insert additional records into the sort. The sorted records are written directly into an output data set.

IBMBKSTC: Invokes the sort/merge program and specifies the use of user exit E35. Invoking this exit can allow the program to either receive all the records from the sort and handle any output that is required, or examine each sorted record and alter or delete it from the output data set.

IBMBKSTD: Invokes the sort/merge program and specifies the use of user exits E15 and E35. The use of these exits is exactly as described for IBMDKSTB and IBMDKSTC above.

Method (chart DKST)

The actual sorting process is controlled by providing the sort routine with the appropriate control information. This is passed as an argument list by the PL/I program to IBMDKST. All calls to SORT go through the interface module as do all exits invoked by SORT.

When exit E15 is invoked, the entry point of the procedure to be used as the exit routine is passed to SORT by IBMDKST. The sort routine then invokes this procedure once for each record presented for sorting. A character string representation of the record is passed back to the sort routine via the interface module. A return code is also passed back to indicate whether the exit is to be invoked again or not.

When exit E35 is invoked, the entry point concerned is passed in the argument list to IBMDKST as described for exit E15 above. The sort routine, at the end of the sort cycle, invokes this exit so that the sorted records can be presented to the exit routine. The exit is entered once for each record sorted. A character string parameter is declared in the exit procedure to receive the records passed back by the sort routine. A return code is returned to SORT to indicate whether any more records are required or not.

The exit routines are addressed by the exit code in the DSA. Control will return to this code, which will then store the registers belonging to SORT, restore the PL/I registers, and branch to the appropriate exit routine interface. Control then returns via register R14 to the point in IBMDKST immediately following that from which the branch to SORT was originally made.

When the module IBMDKST is called, it saves the contents of register R1 and obtains a DSA large enough to hold two nine word save areas, for use by routines in IBMDKST, and some workspace. The first of the two dynamic save areas (DSA) is for use by the sort/merge routines, and the second for use by the exit routines at exits E15 andE35. The second save area is addressed back to the first via a normal back chain slot and contains a flag which is recognised by the PL/1 housekeeping routines. The back chaining and flagging are required to cater for abnormal termination of a user exit. Such termination could happen under the following conditions:

1. The exit procedure terminates due to an error.

2. The exit procedure executes a GO TO statement naming a procedure at a level equal to or higher than the procedure calling the sort routine.

Having obtained the LSA, the module then generates a parameter list, from the arguments passed to it by the caller. The generated list is set up in the first DSA so that arguments corresponding to it can be passed to the sort routine. The list contains the addresses of user exits,if specified, and return code for interprogram communication.

After the parameter list has been set-up, the following actions are performed before a branch and link is made to the sort/merge program:

1. The sort module is loaded by the interface module, the amount of store being determined from the STORAGE parameter in the OPTION statement passed to IBMDKST.

2. The exit code is stored in the first save area, followed by the address of the first save area. Contained in the exit code is an error handling routine.

   A word in the implementation defined appendage of the TCA(the word is originally set by STXIT) is reset to point to the error handling routine, mentioned above, which handles program interrupts for the duration of the sort.

If there is no exit routine, control returns directly to IBMDKST. Finally, the program check exit is reset for control to pass to PL/I error handling routines if required and a return made to the calling program. When the call to sort is one that does specify an exit routine, then on exit from sort, the following actions are performed.

1. A branch is taken to the exit code in the first save area of the interface module. This exit code picks up the address of the first save area, saves the registers of SORT, re-establishes the PL/I environment and branches to an address in IBMDKST.

2. The program check exit is reset so that control passes to PL/I error handling routines if required.

3. Parameters for the exit routines are set from information passed by SORT.

4. On completion of the exit routine, control returns to SORT via IBMDKST. SORT then, dependent upon the return code it is passed, continues processing as required.

If abnormal termination of the exit routine occurs, a flag X'02', set to ON in the second byte of the second save area, is recognised by PL/I housekeeping routines. In consequence, PL/I housekeeping gives control to the interface module IBMDKST which terminates the sort routine in the following manner.

If the sort is being terminated at an E35 exit, the interface module returns control to SORT with a return code of 4 to see if any sorted records remain for output. If such is the case, SORT returns control, and the address of the next sorted record, to IBMDKST. In consequence, IBMDKST continues to return to SORT with a return code of 4, until SORT has passed the address of every outstanding record. When this circumstance prevails, IBMDKST returns to SORT with a return code of 8 to terminate that routine.

If the sort is being terminated at an E15 exit, no further records are passed to the SORT program, and termination proceeds as described above for the E35 exit.

Control then passes back to the housekeeping routine.

Note: If the SORT/MERGE program No.5736-SM1 is used, then there is no need to terminate the SORT routine in the manner described above. Instead, IBMDKST merely returns control to SORT with a return code of 16 to terminate that routine. Control will then pass back to the housekeeping routine.

Error and Exceptional Conditions

Return is made to caller via the Link Register with a special return code value (8).

Linkage

```
R1    = A(PLIST)
PLIST = A(Sort control statement string locator)
        A(Record control statement string locator)
        A(Option control statement string locator)
        A(PL/I procedure to be invoked as an E15 exit)
        A(PL/I procedure to be invoked as an E35 exit)
        A(Inpfil control statement string locator)
        A(Outfil control statement string locator)
```

Calls

SORT/MERGE program Exit routines.

Called By

Compiled code.

APPENDIX A: LIST OF MODULES

The following list of modules is arranged in alphabetical order of the last three letters of the module name. This ordering is used to save the reader the trouble of remembering whether the module is prefixed with IBMB or IBMD.

Name	Function	Size (approx)
IBMBAAH	ALL, ANY (simple and interleaved arrays)	390 bytes
IBMBAIH	Indexer for interleaved arrays	100 bytes
IBMBAMM	Structure mapping	1760 bytes
IBMBANM	STRING built-in function	1630 bytes
IBMBAPC	PROD (arrays with fixed point integer elements)	580 bytes
IBMBAPF	PROD (arrays with floating point elements)	370 bytes
IBMBAPM	STRING pseudovariable	1230 bytes
IBMBASC	SUM (arrays with fixed-point elements)	420 bytes
IBMBASF	SUM (arrays with floating-point elements)	330 bytes
IBMBAYF	POLY built-in function	380 bytes
IBMBBBA	AND, OR operations (byte-aligned bit strings)	520 bytes
IBMBBBN	NOT operation (byte-aligned bit strings)	400 bytes
IBMBBCI	INDEX (character strings)	200 bytes
IBMBBCK	Concatenate, REPEAT (character strings)	610 bytes
IBMBBCT	TRANSLATE (character strings)	770 bytes
IBMBBCV	VERIFY (character strings)	210 bytes
IBMBBGB	BOOL (bit strings)	660 bytes
IBMBBGC	Compare (general bit strings)	240 bytes
IBMBBGF	Assign (byte-aligned bit strings) and Fill (general bit strings)	390 bytes
IBMBBGI	INDEX (bit strings)	350 bytes
IBMBBGK	Concatenate, REPEAT, General Assign (bit strings)	890 bytes
IBMBBGS	SUBSTR SID	330 bytes
IBMBBGV	VERIFY (bit strings)	420 bytes
IBMBCAC	Conversion director (arithmetic to character)	700 bytes
IBMBCBB	Conversion (bit to bit)	350 bytes
IBMBCBC	Conversion (bit to character)	220 bytes
IBMBCBQ	Conversion (bit to pictured character)	240 bytes
IBMBCCA	Conversion director (character to arithmetic)	520 bytes
IBMBCCB	Conversion (character to bit)	420 bytes
IBMBCCC	HIGH, LOW, Assign (character strings)	270 bytes
IBMBCCQ	Conversion (character to pictured character)	410 bytes
IBMDCCS	String conversion director bootstrap	340 bytes
IBMBCE	Conversion (fixed decimal - free decimal - float - fixed binary)	720 bytes
IBMBCGP	Check input (pictured decimal)	870 bytes
IBMBCGQ	Check input (pictured character)	200 bytes
IBMBCGT	Table of powers of ten	140 bytes
IBMBCGZ	Set a subfield of a complex number to zero	300 bytes
IBMBCH	Conversion (fixed binary - float - free decimal)	480 bytes
IBMBCK	Conversion (fixed decimal - free decimal - fixed decimal)	370 bytes
IBMBCM	Conversion (pictured decimal to packed decimal)	810 bytes
IBMBCO	Conversion (packed decimal to pictured decimal)	1080 bytes
IBMBCP	Conversion (bit to fixed binary or float)	490 bytes
IBMBCR	Conversion (fixed binary or float to bit)	400 bytes
IBMBCT	Conversion (decimal constant to packed decimal)	670 bytes
IBMBCU	Conversion (binary constant to float)	780 bytes
IBMBCV	Conversion (packed decimal to E format)	670 bytes
IBMBCW	Conversion (packed decimal to F format)	490 bytes
IBMBCY	Conversion (fixed binary to fixed binary and float to float)	250 bytes

IBMDEFL	FLOW option	1200 bytes
IBMBEOC	ON-code	220 bytes
IBMBEOL	ONLOC built-in function	160 bytes
IBMBERC	CHECK system action	490 bytes
IBMDERR	Error handler	1500 bytes
IBMDEVO	Event Variable operations	16 bytes
IBMDIEC	Interlanguage housekeeping	500 bytes
IBMDIEF	Interlanguage housekeeping	910 bytes
IBMDIEP	Interlanguage housekeeping	1000 bytes
IBMDJDS	DISPLAY	740 bytes
IBMDJDT	DATE built-in function	80 bytes
IBMDJDY	DELAY	130 bytes
IBMDJDZ	DISPLAY without EVENT	590 bytes
IBMDJTT	TIME built-in function	150 bytes
IBMBJWI	WAIT (array events)	390 bytes
IBMDJWT	WAIT (multiple events)	800 bytes
IBMGJWT	WAIT (single event)	220 bytes
IBMDKCP	Checkpoint/restart interface	820 bytes
IBMDKDM	Dump bootstrap	140 bytes
IBMDKST	SORT interface	1440 bytes
IBMBMAL	SQRT (long float real)	170 bytes
IBMBMAS	SQRT (short float real)	170 bytes
IBMBMAX	SQRT (short float complex)	290 bytes
IBMBMAY	SQRT (long float complex)	300 bytes
IBMBMBL	EXP (long float real)	460 bytes
IBMBMBS	EXP (short float real)	260 bytes
IBMBMBX	EXP (short float complex)	140 bytes
IBMBMBY	EXP (long float complex)	140 bytes
IBMBMCL	ERF, ERFC (long float real)	640 bytes
IBMBMCS	ERF, ERFC (short float real)	410 bytes
IBMBMDL	LOG, LOG2, LOG10 (long float real)	340 bytes
IBMBMDS	LOG, LOG2, LOG10 (short float real)	260 bytes
IBMBMDX	LOG (short float complex)	230 bytes
IBMBMDY	LOG (long float complex)	230 bytes
IBMBMGL	SIN, SIND, COS, COSD (long float real)	390 bytes
IBMBMGS	SIN, SIND, COS, COSD (short float real)	310 bytes
IBMBMGX	SIN, SINH, COS, COSH (short float complex)	310 bytes
IBMBMGY	SIN, SINH, COS, COSH (long float complex)	370 bytes
IBMBMHL	TAN, TAND (long float real)	320 bytes
IBMBMHS	TAN, TAND (short float real)	260 bytes
IBMBMHX	TAN, TANH (short float complex)	230 bytes
IBMBMHY	TAN, TANH (long float complex)	230 bytes
IBMBMIL	SINH, COSH (long float real)	240 bytes
IBMBMIS	SINH, COSH (short float real)	160 bytes
IBMBMJL	TANH (long float real)	260 bytes
IBMBMJS	TANH (short float real)	200 bytes
IBMBMKL	ATAN, ATAND (long float real)	470 bytes
IBMBMKS	ATAN, ATAND (short float real)	360 bytes
IBMBMKX	ATAN, ATANH (short float complex)	260 bytes
IBMBMKY	ATAN, ATANH (long float complex)	260 bytes
IBMBMLL	ATANH (long float real)	260 bytes
IBMBMLS	ATANH (short float real)	180 bytes
IBMBMML	ASIN, ACOS (long float real)	350 bytes
IBMBMMS	ASIN, ACOS (short float real)	260 bytes
IBMBMOD	ADD (fixed decimal real or complex)	290 bytes
IBMBMPU	MULTIPLY (fixed binary complex)	290 bytes
IBMBMPV	MULTIPLY (fixed decimal complex)	280 bytes
IBMBMQU	DIVIDE (fixed binary complex)	460 bytes
IBMBMQV	DIVIDE (fixed decimal complex)	580 bytes
IBMBMRU	ABS (fixed binary complex)	210 bytes
IBMBMRV	ABS (fixed decimal complex)	540 bytes
IBMBMRX	ABS (short float complex)	120 bytes
IBMBMRY	ABS (long float complex)	130 bytes
IBMBMUD	Shift and assign/load (fixed decimal real)	360 bytes
IBMBMVU	Multiplication and Division (fixed binary complex)	290 bytes

IBMBMVV	Multiplication and Division (fixed decimal complex)	660 bytes
IBMBMVW	Multiplication (long and short float complex)	120 bytes
IBMBMWX	Division (short float complex)	100 bytes
IBMBMWY	Division (long float complex)	100 bytes
IBMBMXL	Integer exponentiation (long float real)	140 bytes
IBMBMXS	Integer exponentiation (short float real)	140 bytes
IBMBMXW	Integer exponentiation (short and long float complex)	410 bytes
IBMBMYL	General exponentiation (long float real)	160 bytes
IBMBMYS	General exponentiation (short float real)	150 bytes
IBMBMYX	General exponentiation (short float complex)	260 bytes
IBMBMYY	General exponentiation (long float complex)	270 bytes
IBMDOCL	OPEN/CLOSE bootstrap	240 bytes
IBMBPAF	Controlled variable management	150 bytes
IBMBPAM	AREA management	540 bytes
IBMBPGO	Reset CHECK enablement	40 bytes
IBMDPGR	Storage management	610 bytes
IBMDPIR	Program initialization from system	420 bytes
IBMDPJR	Program initialization from caller	330 bytes
IBMDPOL	Overlay space saving module	8 bytes
IBMDPOV	Overlay	110 bytes
IBMBPRC	Return code module.	40 bytes
IBMDRIO	Record I/O interface module	280 bytes
IBMBSAI	Input conversion director (A, P, and B formats)	420 bytes
IBMBSAO	Output conversion director (A format)	130 bytes
IBMBSBO	Output conversion director (character-P and B formats)	400 bytes
IBMBSCI	Input conversion director (C format)	300 bytes
IBMBSCO	Output conversion director (C format)	290 bytes
IBMDSCP	COPY	230 bytes
IBMDSCV	Conversion fix-up bootstrap	90 bytes
IBMDSDI	Data-directed input	2090 bytes
IBMDSDJ	Data-directed input	2090 bytes
IBMDSDO	Data-directed output	1210 bytes
IBMDSED	Edit-directed I/O housekeeping	1050 bytes
IBMDSEE	Edit-directed combination module	1420 bytes
IBMDSEH	Edit-directed combination subset module	880 bytes
IBMDSEI	Edit-directed input	440 bytes
IBMDSEO	Edit-directed output	210 bytes
IBMBSFI	Input conversion director (F and E formats)	240 bytes
IBMBSFO	Output conversion director (F and E formats)	210 bytes
IBMDSII	GET FILE initialization	420 bytes
IBMDSIO	PUT FILE initialization	330 bytes
IBMDSIS	GET or PUT STRING initialization	350 bytes
IBMDSLI	List-directed input	2220 bytes
IBMDSLJ	List-directed input	2070 bytes
IBMDSLO	List-directed output	1610 bytes
IBMBSMW	Missing output width module	340 bytes
IBMBSPI	Input conversion director (P format)	370 bytes
IBMDSPL	PAGE, LINE, and SKIP	530 bytes
IBMBSPO	Output conversion director (P format)	290 bytes
IBMDSTF	Stream input transmitter	440 bytes
IBMDSTI	Stream print F-format transmitter	190 bytes
IBMDSXC	X and COLUMN format items	440 bytes
IBMBTOC	COMPLETION pseudovariable and Event variable assignment	130 bytes
DFHPL1I	CICS initialization bootstrap	100 bytes

APPENDIX B: LIST OF LIBRARY MACRO INSTRUCTIONS

Name	Function
IBMBXCH	Chains a specified element to a specified chain.
|IBMBXCIC	Map of CICS appendage
IBMBXDBG	Debug macro.
IBMBXDBL	Debug macro.
IBMBXDBM	Debug macro
IBMBXDC	Dechains a specified element.
IBMBXDD	Provides a DSECT map of the 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.
IBMBXFLT	Provides DSECT maps of the heading for the flow statement table and for the meanings of the bits in the flag byte of each statement number entry.
IBMBXEV	Provides a DSECT map of an Event Variable (EV).
IBMBXFV	Frees a Variable Data Area (VDA).
IBMDXGC	Contains the GOTO code that is copied into the TCA by module 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 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	Map of the DOS PLSTART parameter list.
|IBMBXPLC	Map of the CICS initialization parameter list.
IBMBXRRI	Provides a record checking table for reference in library modules.
IBMBXRRT	Generates code to set up the registers for record checking 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 DSECT maps of an EVTAB element.
IBMDXCOM	Provides a DSECT map of the DOS communication region.
IBMDXDGT	Debug macro.
IBMDXDTF	Provides a DSECT map of the Define The File Block (DTF).
IBMDXFCB	Provides a DSECT map of the File Control Block (FCB).
IBMDXOSA	Defines a DSECT for Open/Close workspace.
IBMDXTA	Provides a DSECT map of the implementation defined appendage of the Task Communications Area (TCA).

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 executed 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 macro instruction IBMBXDBG.

LEVEL=LAB selects the label trace facility provided by macro 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 cther 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, source 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 IBMBXBDI macro instructions in modules IBMDPIR and IBMDPII and change the PHASE operand 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 onto the appropriate system library. The following 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 to 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 information

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 IBMBXDBL 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

The chart references for the flowcharts in this part are derived from the last four letters of the module name; for example, the flowchart for module IBMDERR is on chart DERR. The charts are arranged in alphabetical order of the last <u>three</u> letters of the chart reference.

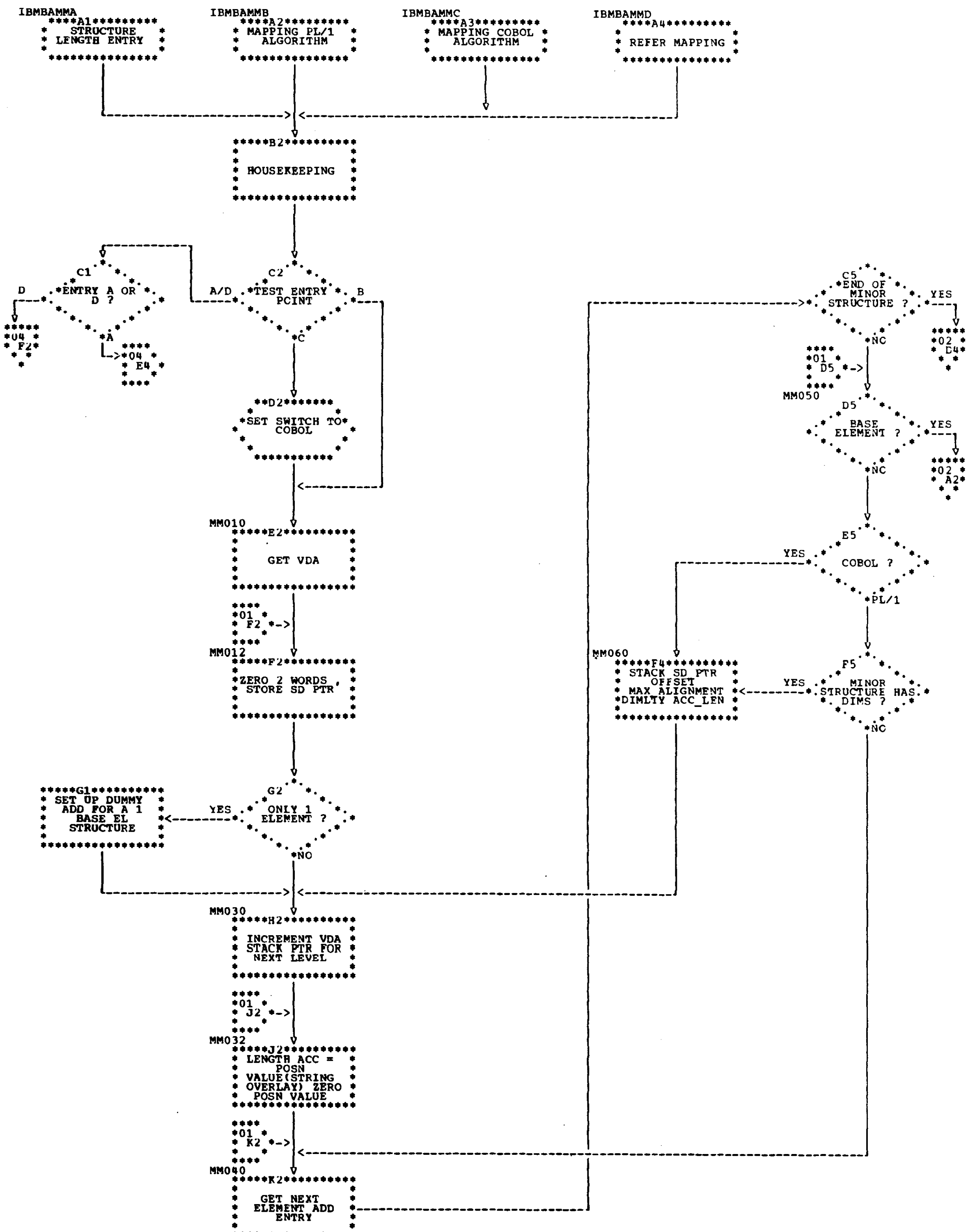


Chart BAMM. Structure mapping (part 1 of 4)

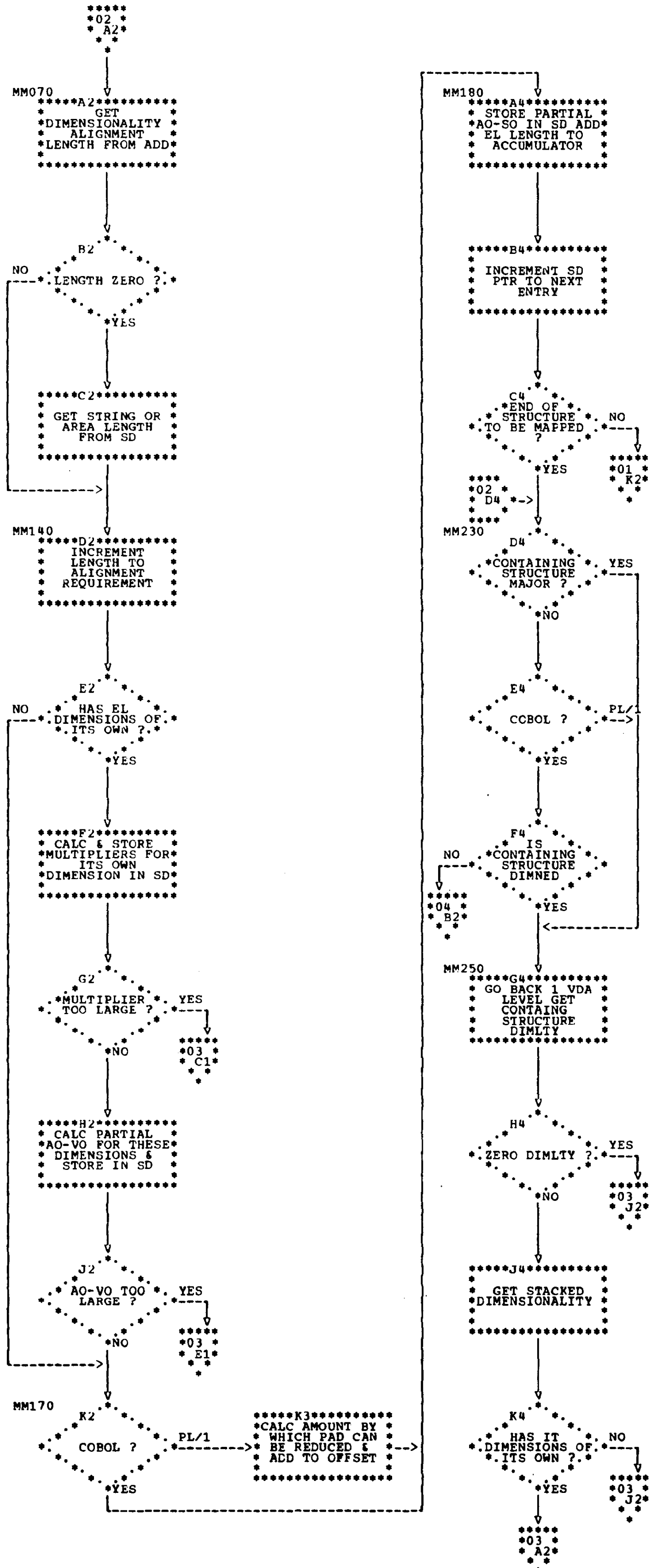


Chart BAMM. Structure mapping (part 2 of 4)

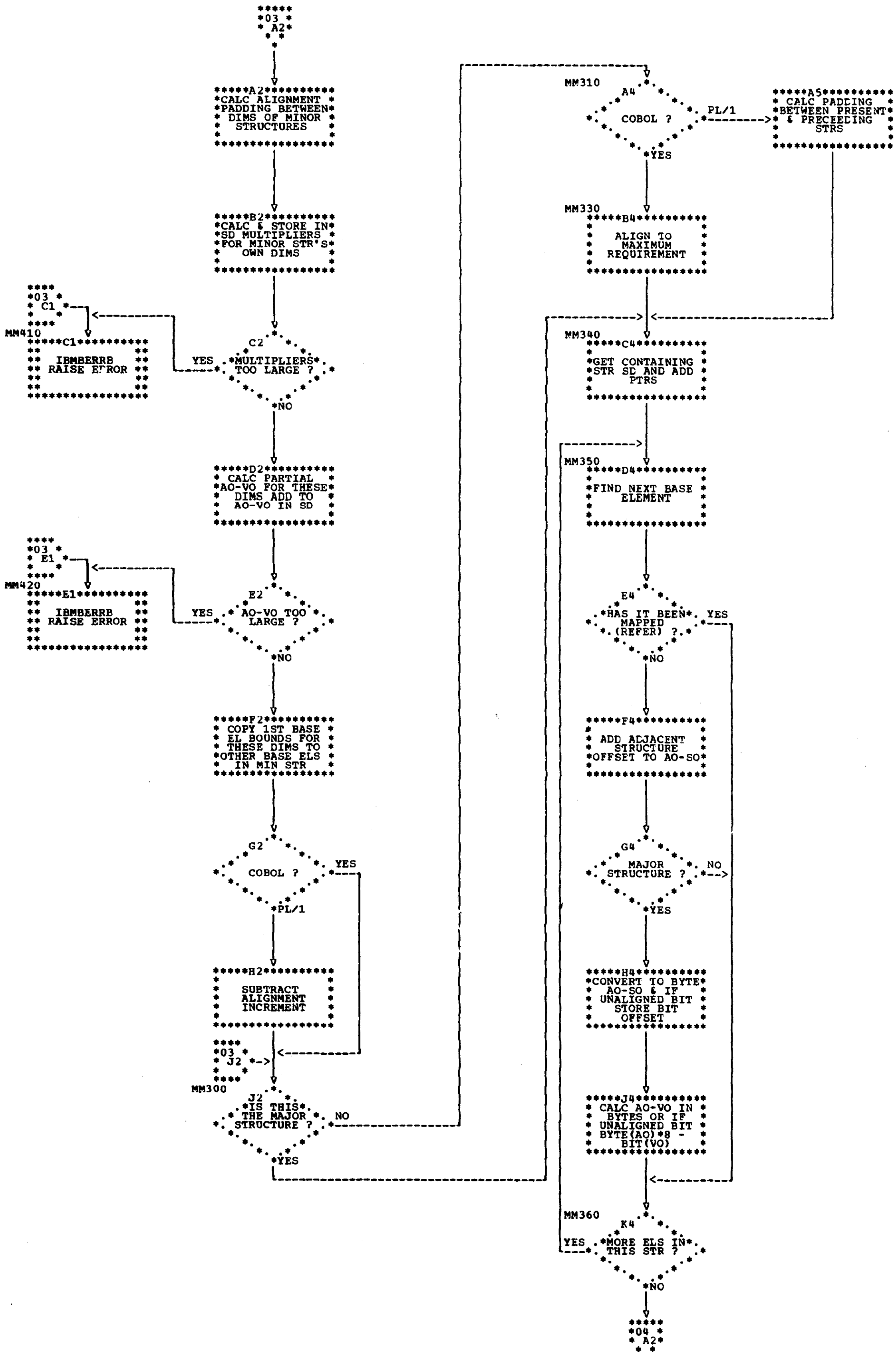


Chart BAMM. Structure mapping (part 3 of 4)

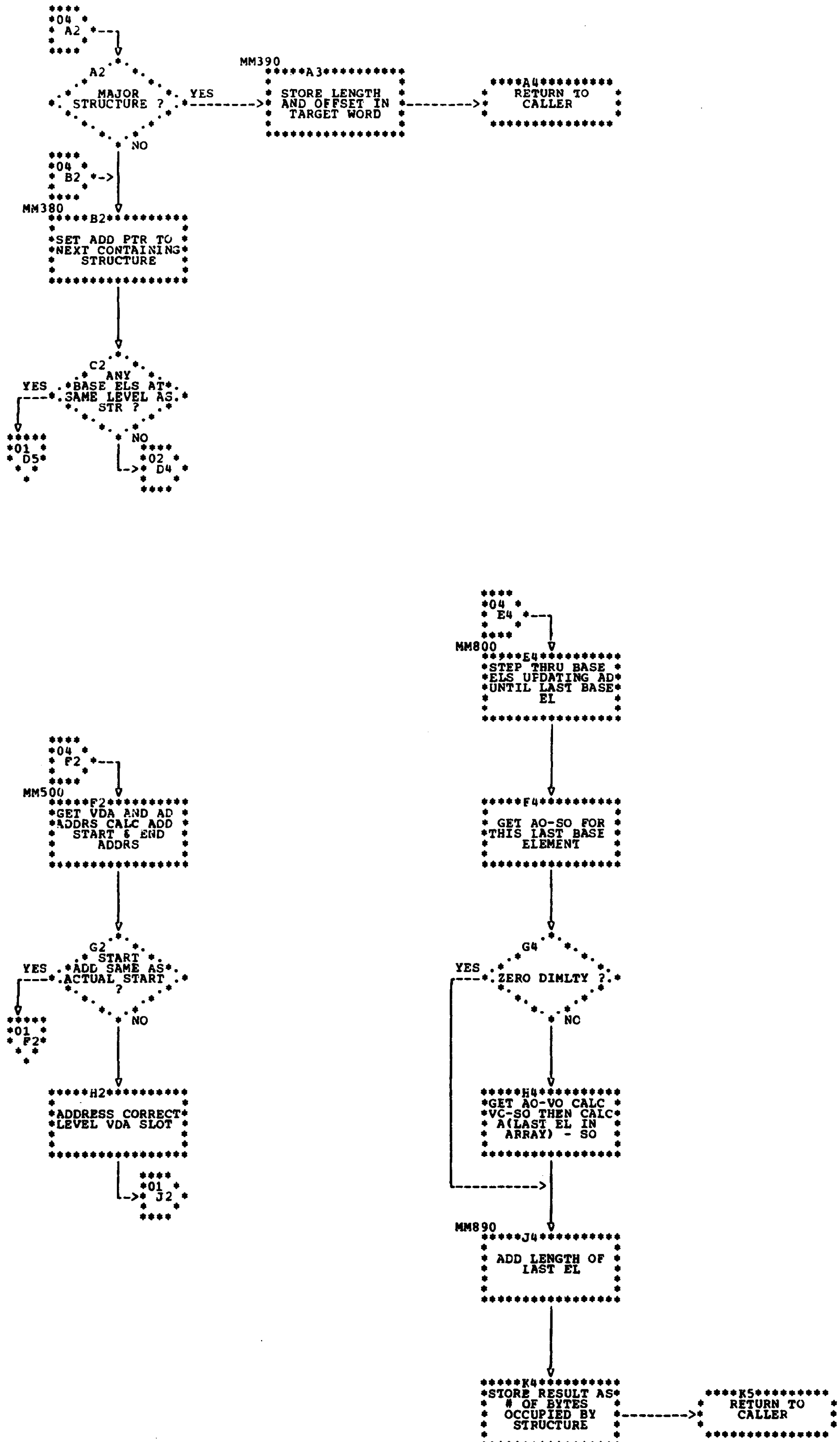


Chart BAMM. Structure mapping (part 4 of 4)

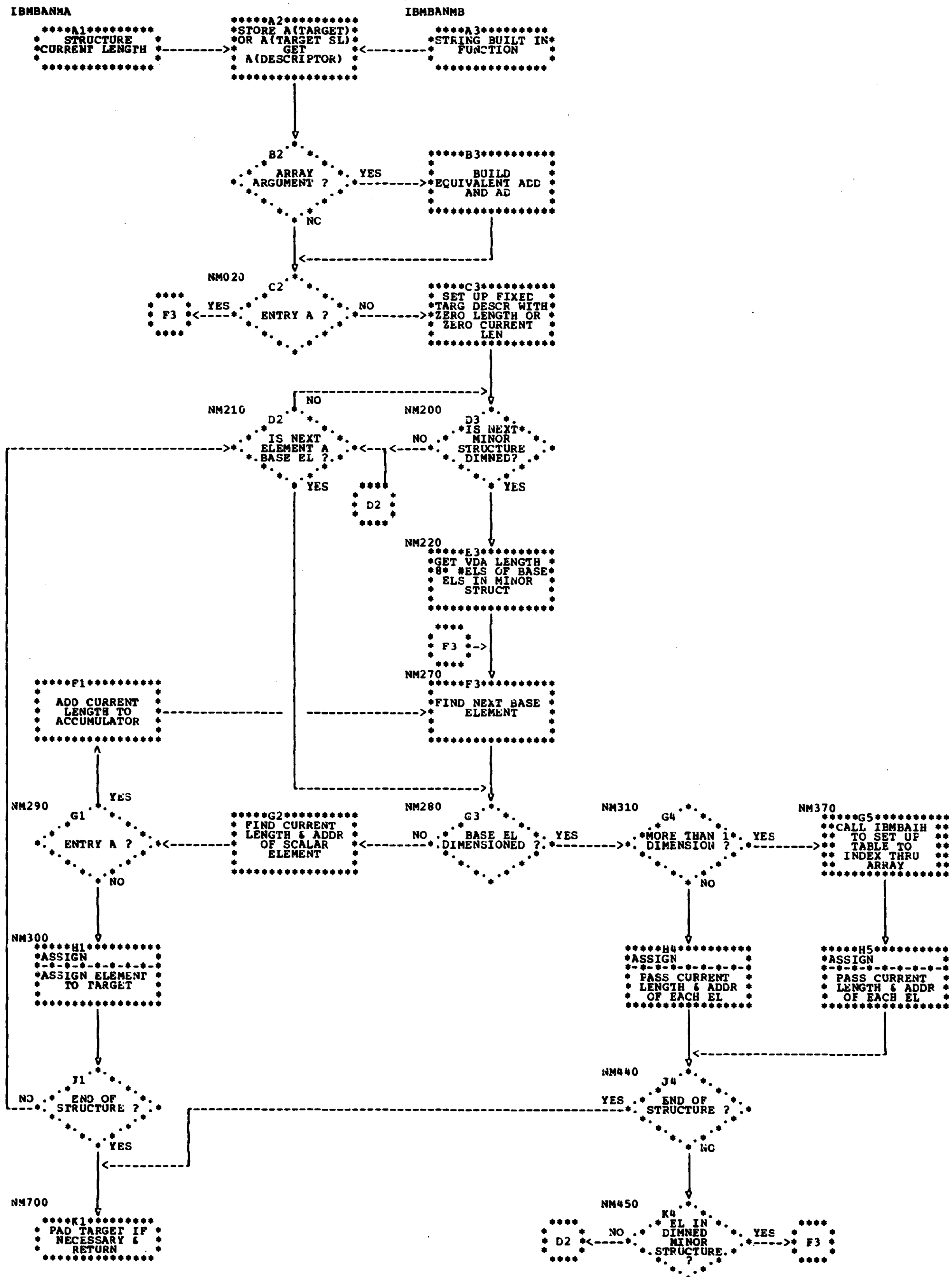


Chart BANM. STRING built-in function (part 1 of 2)

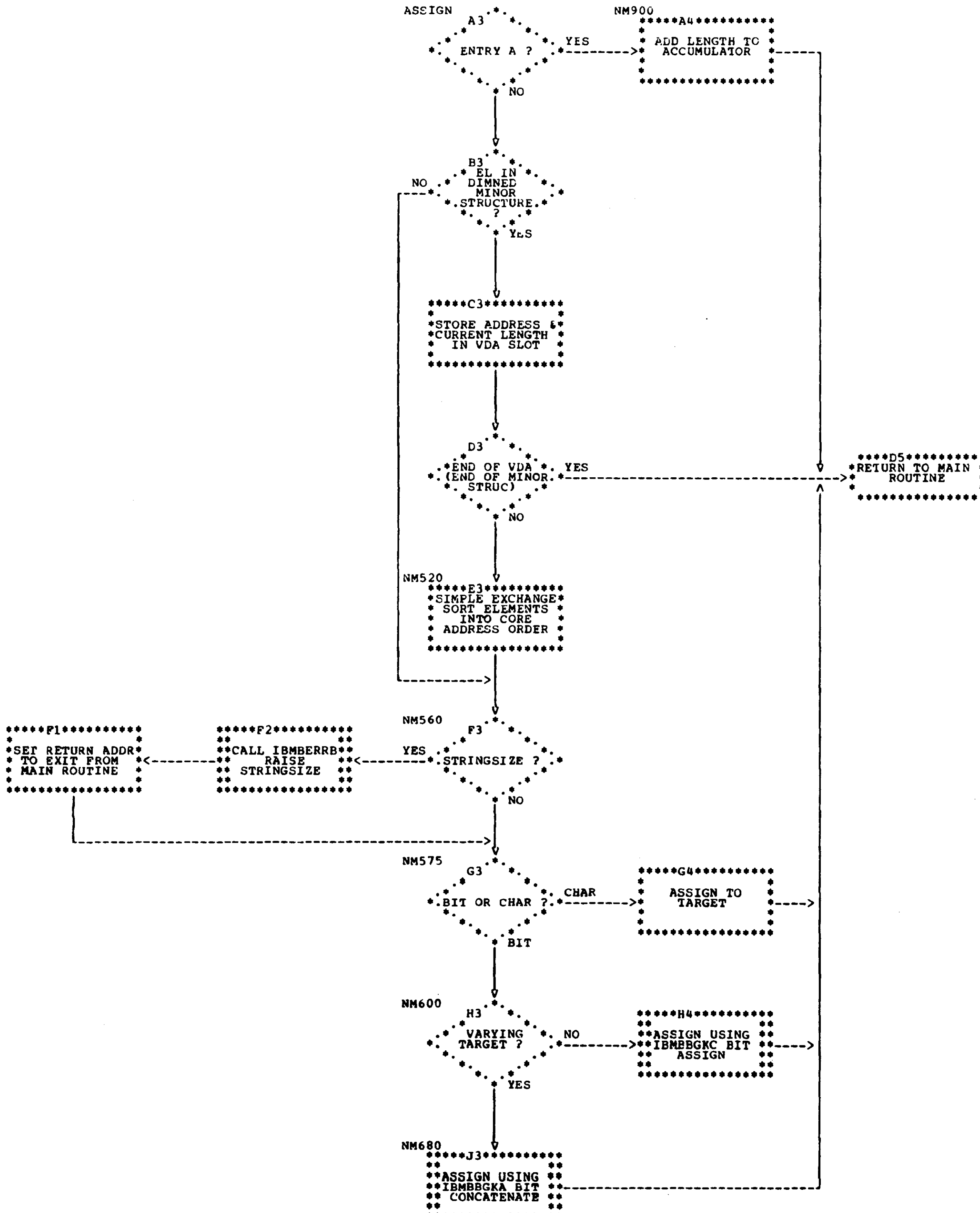


Chart BANM. STRING built-in function (part 2 of 2)

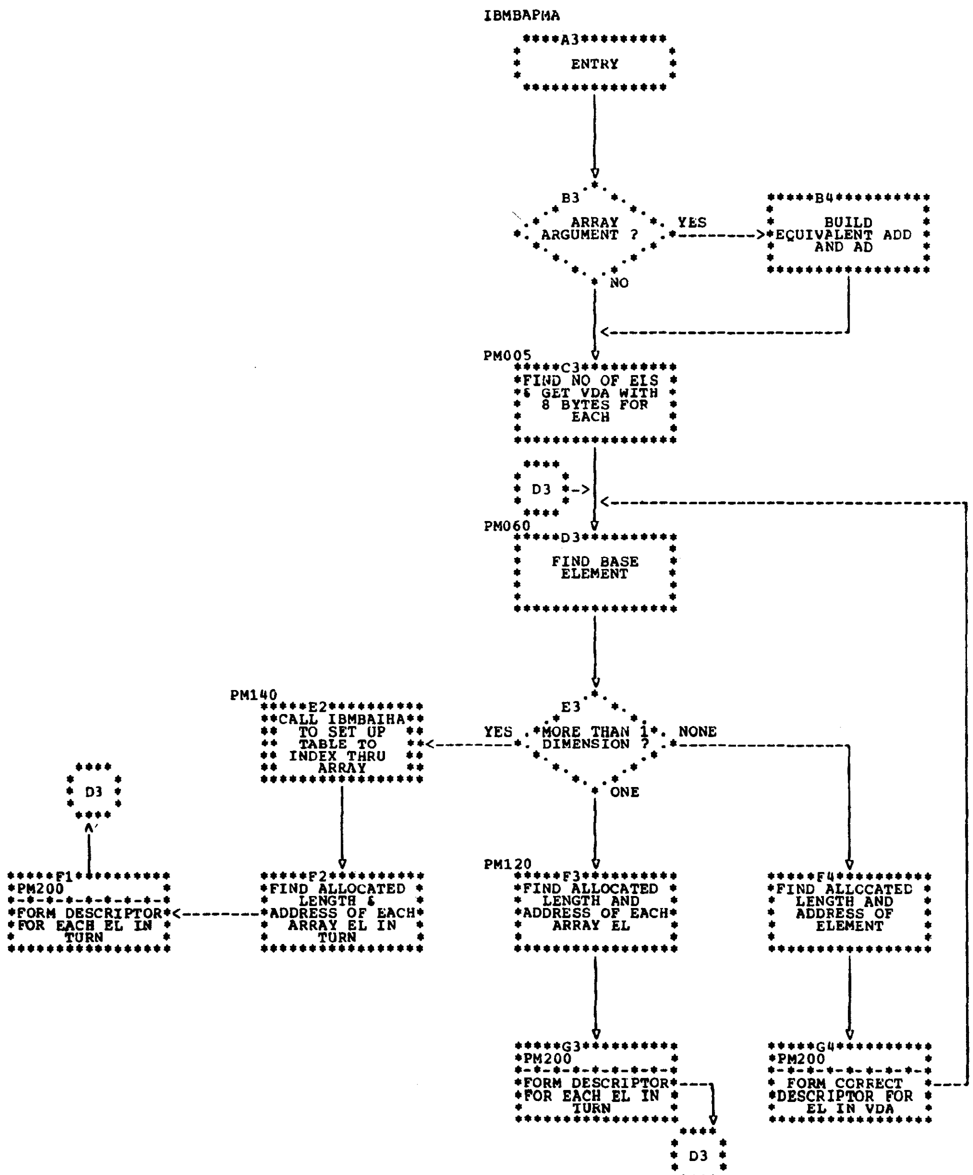


Chart BAPM. STRING pseudovariable (part 1 of 2)

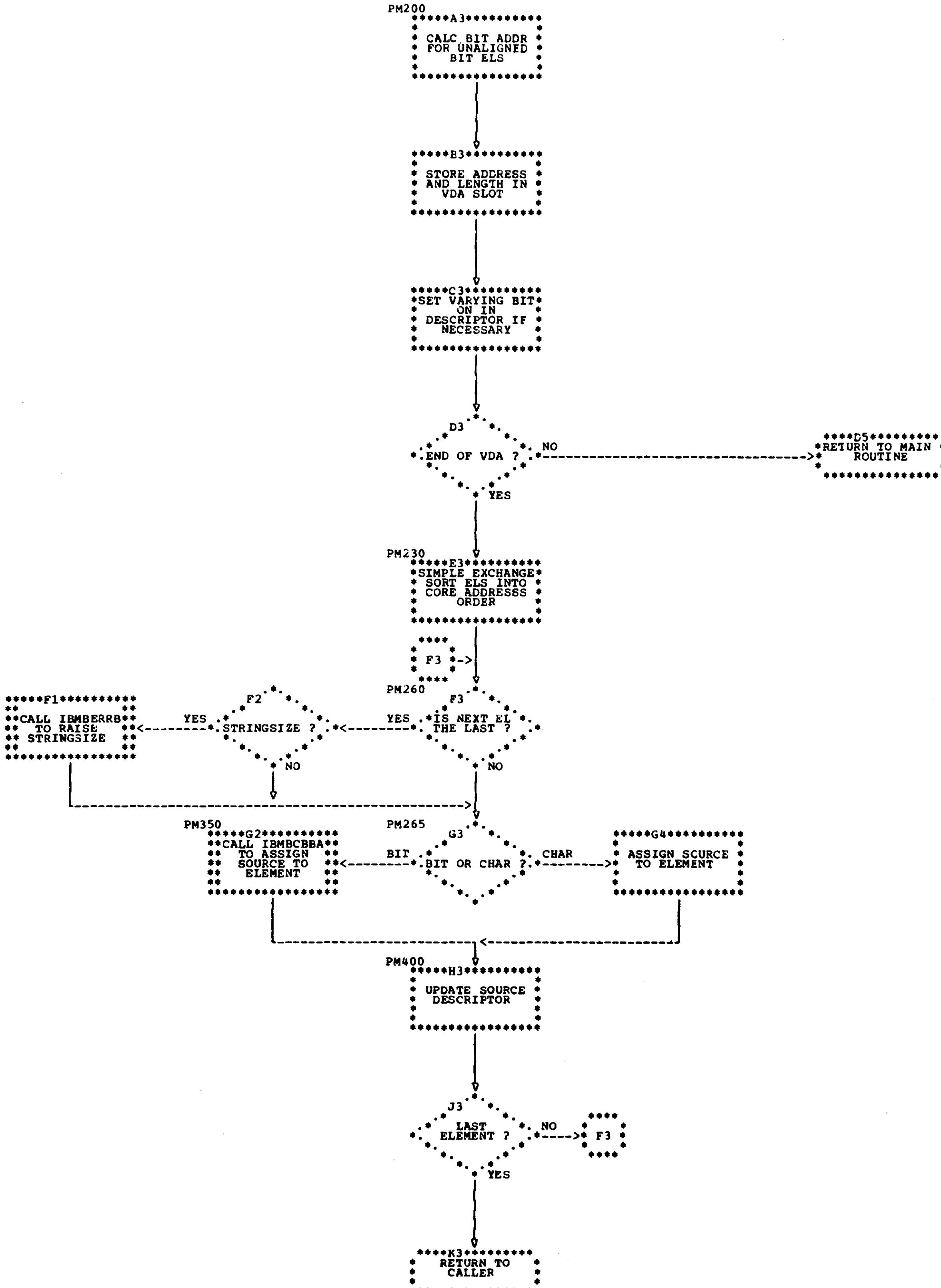


Chart BAPM. STRING pseudovariable (part 2 of 2)

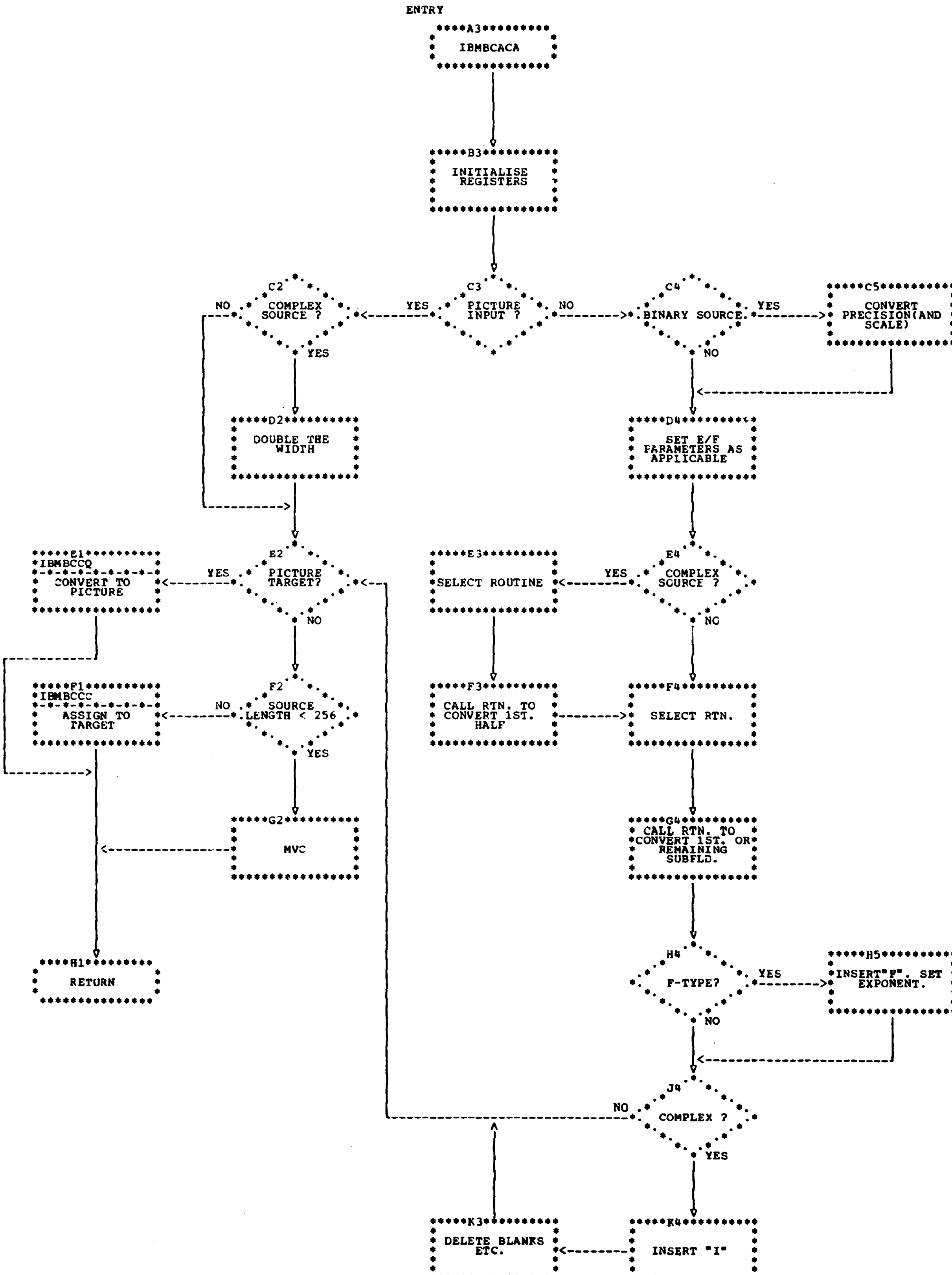


Chart BCAC. Conversion director (arithmetic to character)

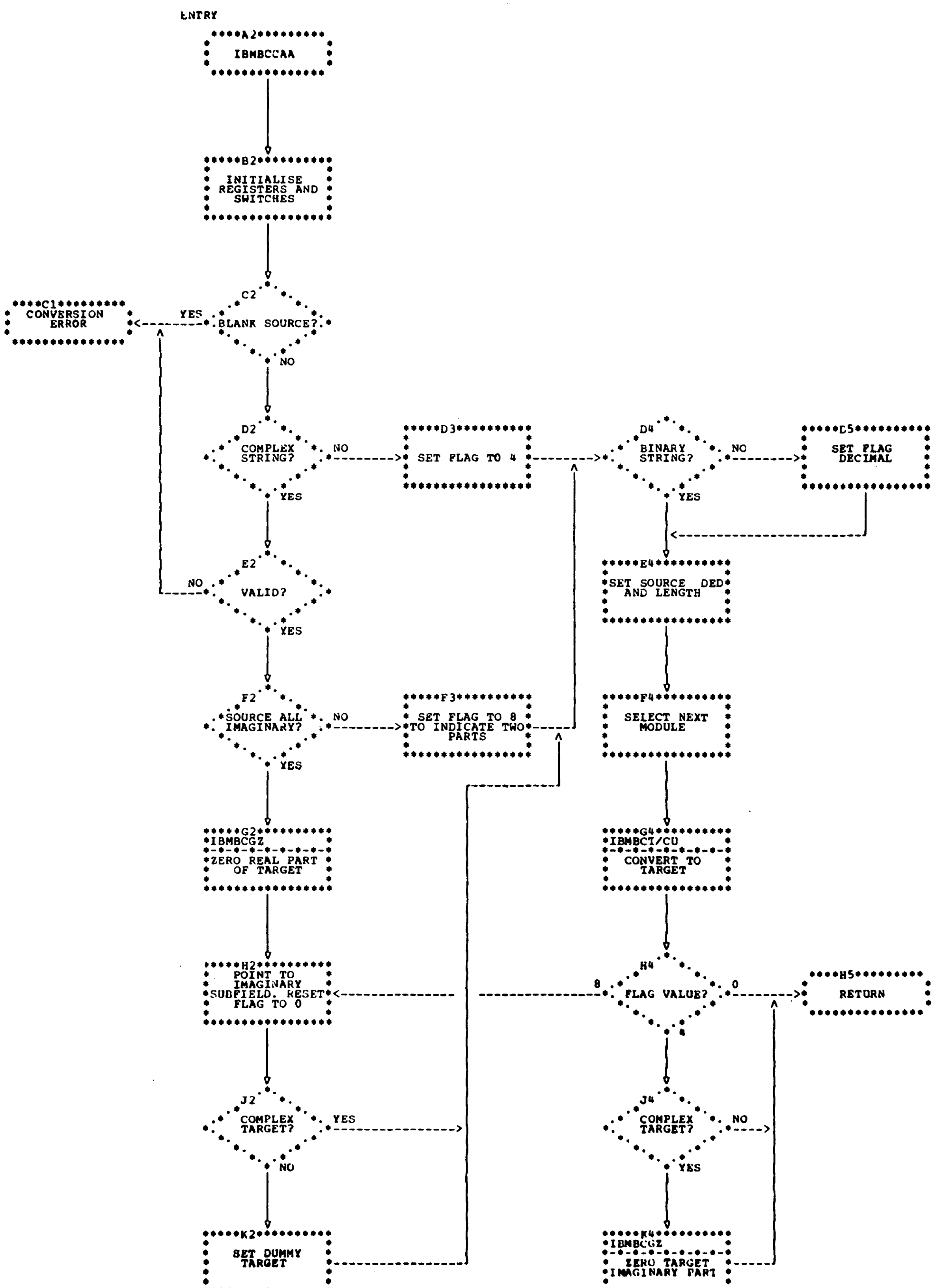


Chart BCCA. Conversion director (character to arithmetic)

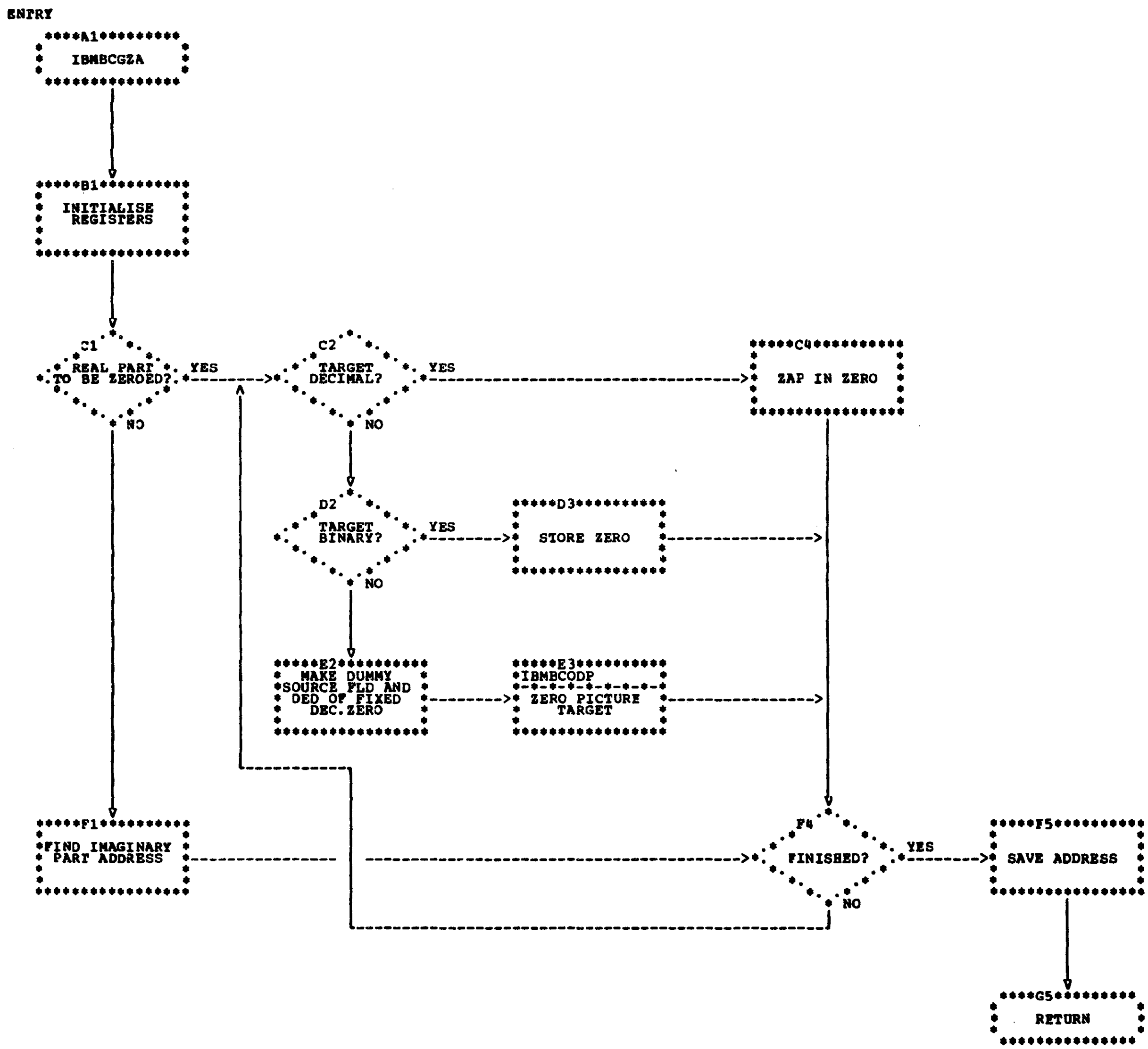


Chart BCGZ. Zero a subfield of a complex number

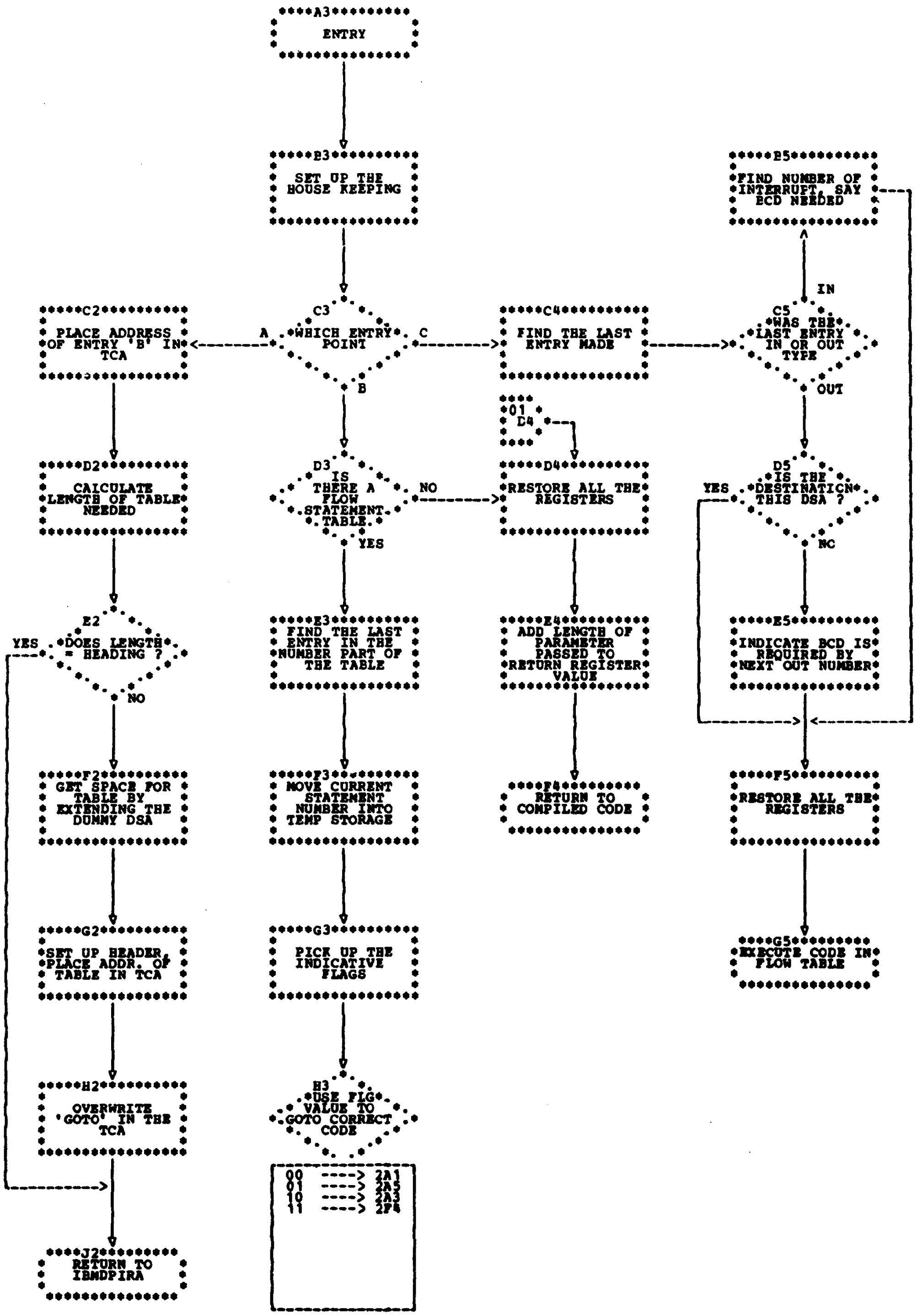


Chart DEFL. FLOW option module (part 1 of 2)

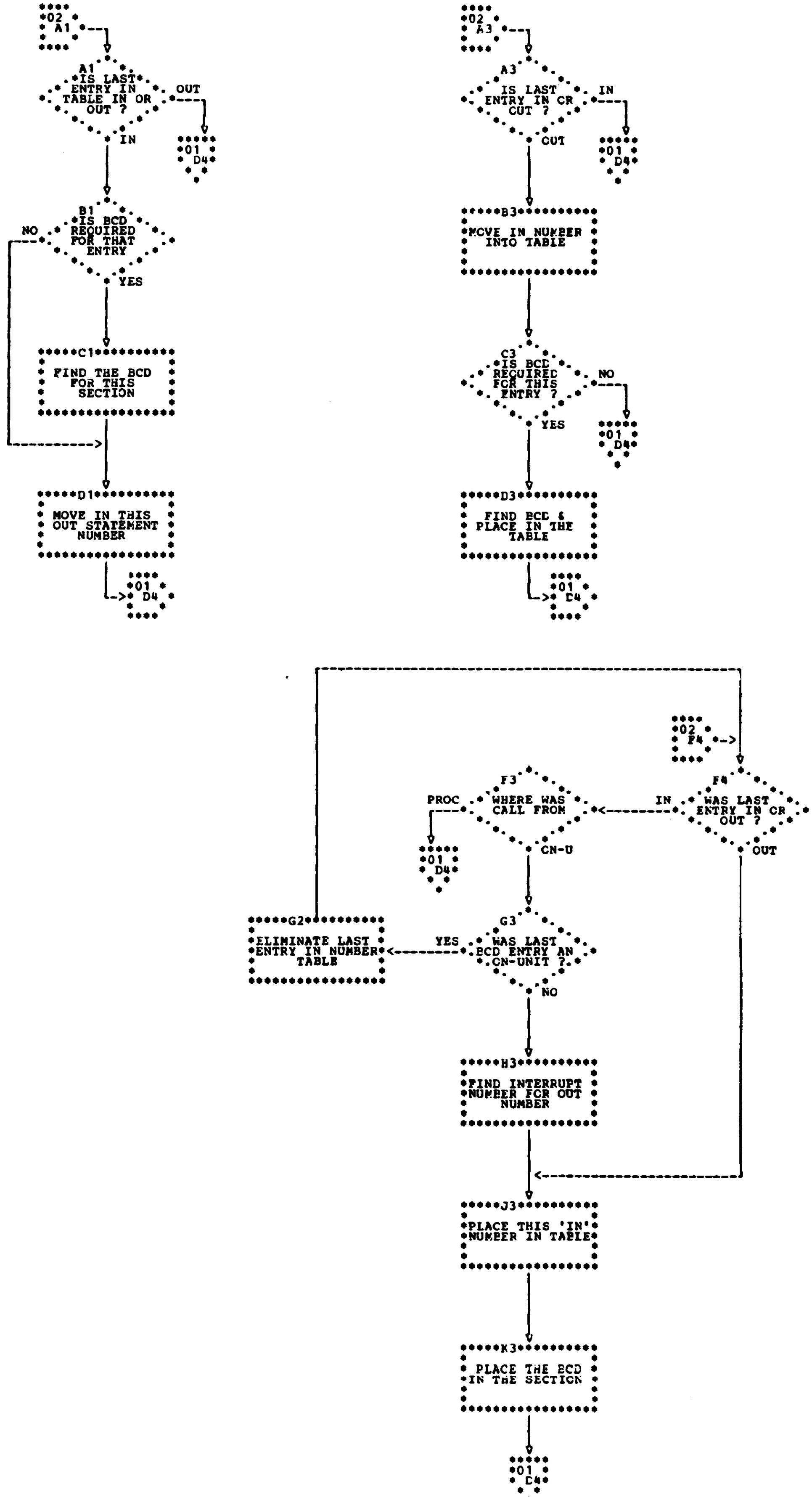


Chart DEFL. FLOW option module (part 2 of 2)

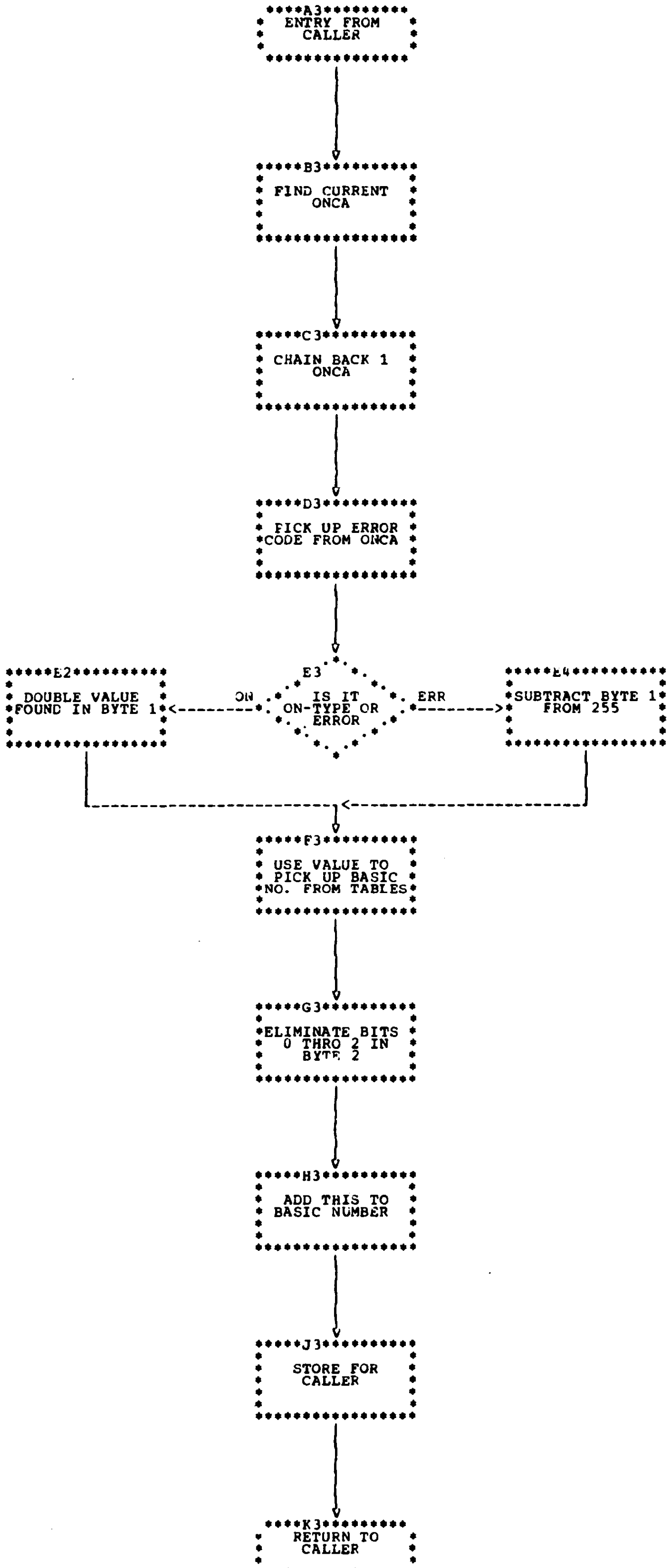


Chart BEOC. On-code module

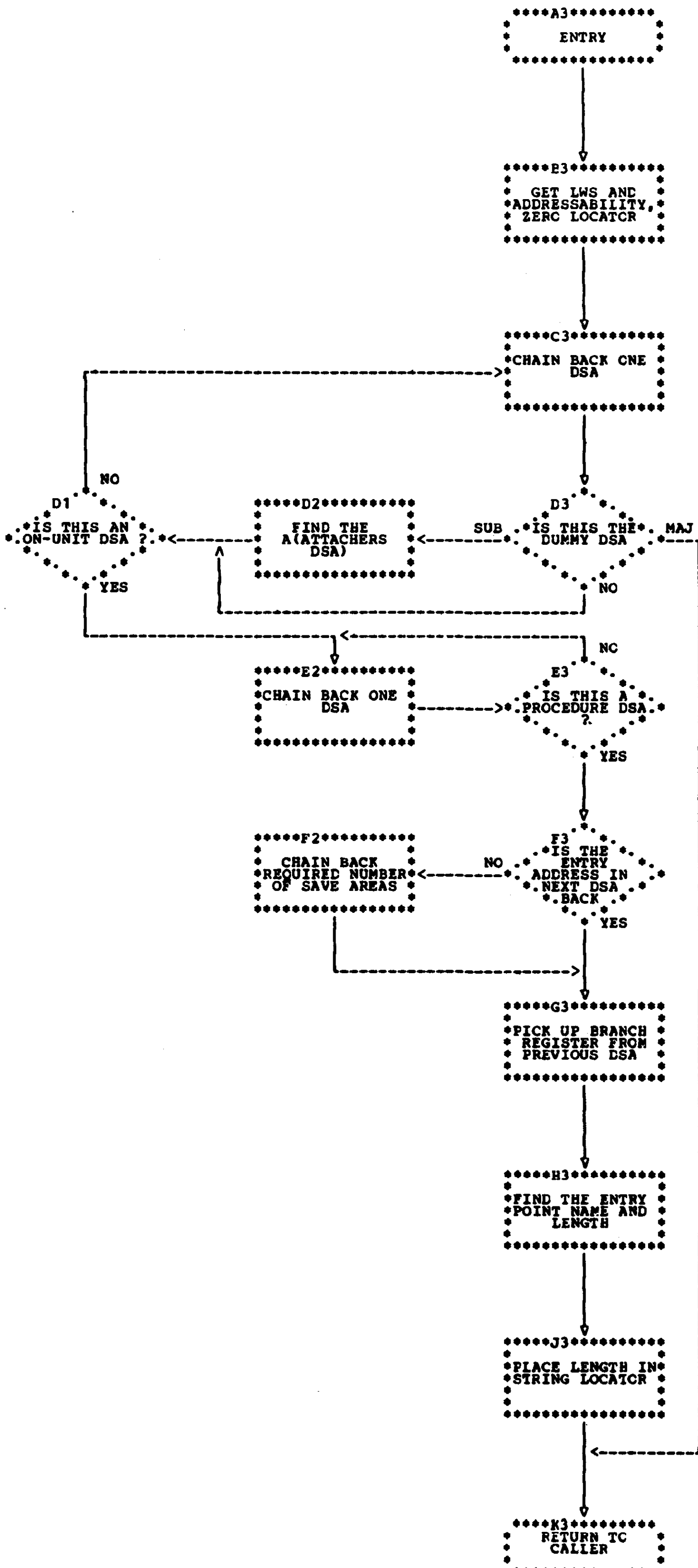


Chart BEOL. ONLOC built-in function

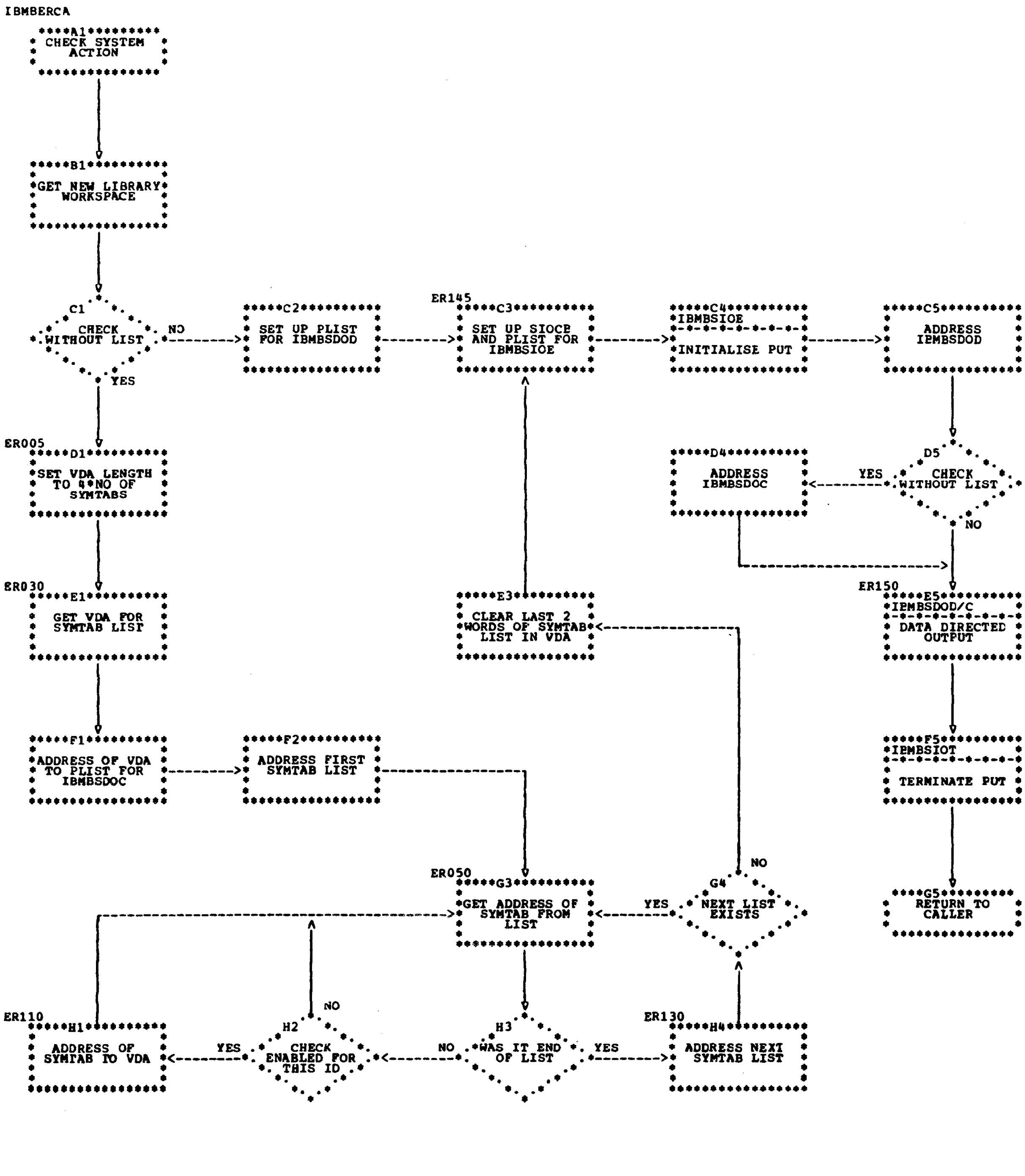


Chart BERC. CHECK system action

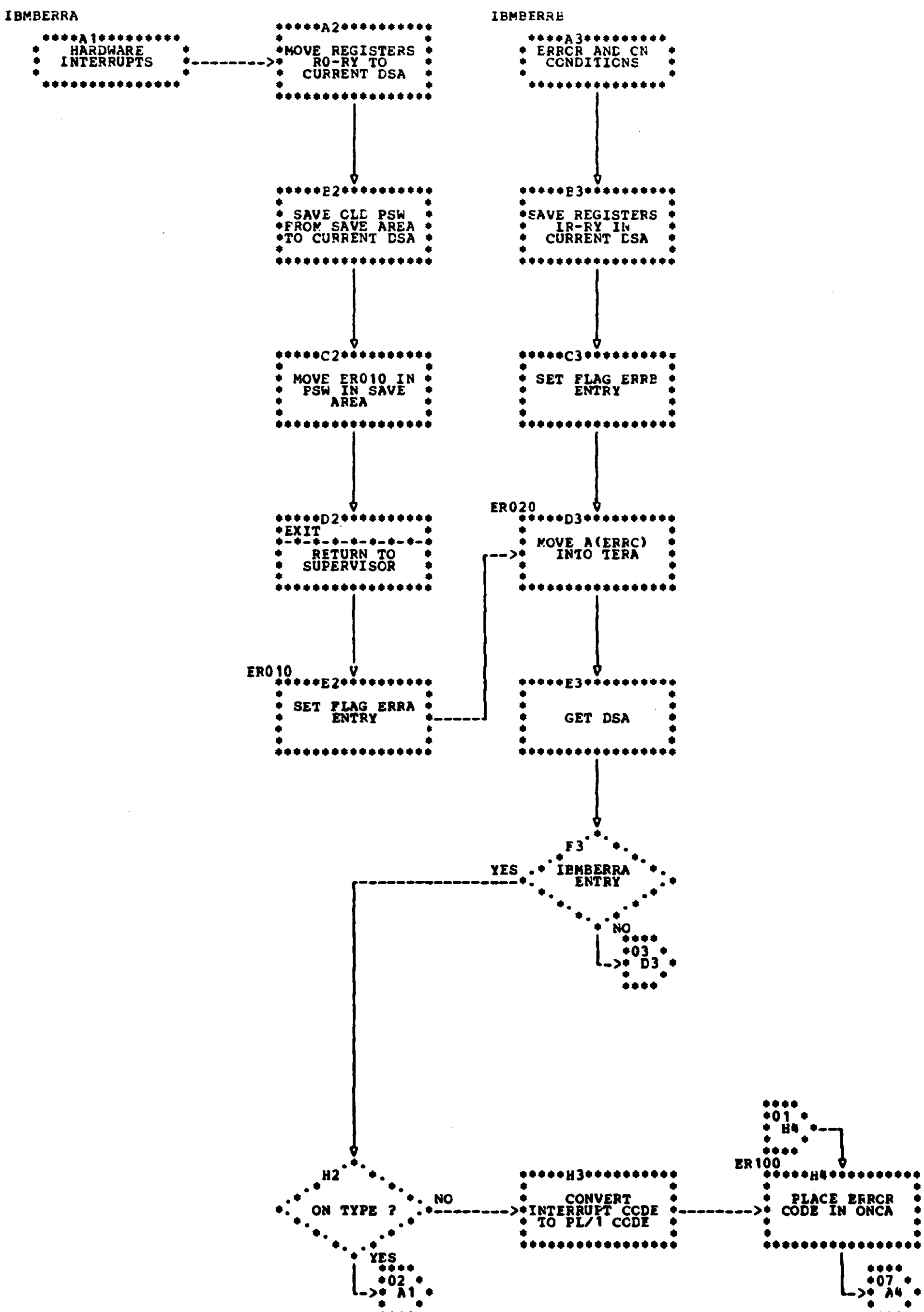


Chart DERR. Error handler (part 1 of 8)

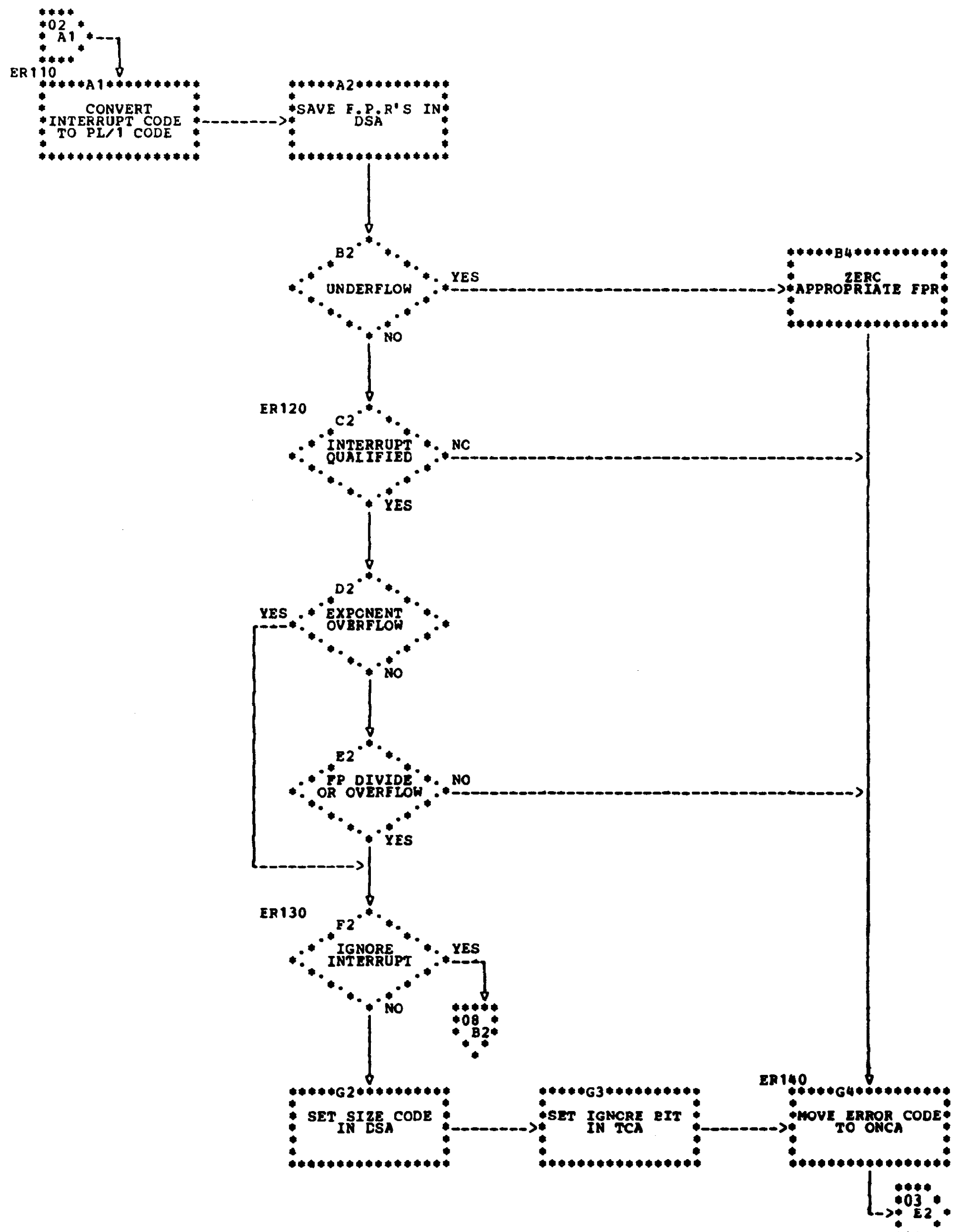


Chart DERR. Error handler (part 2 of 8)

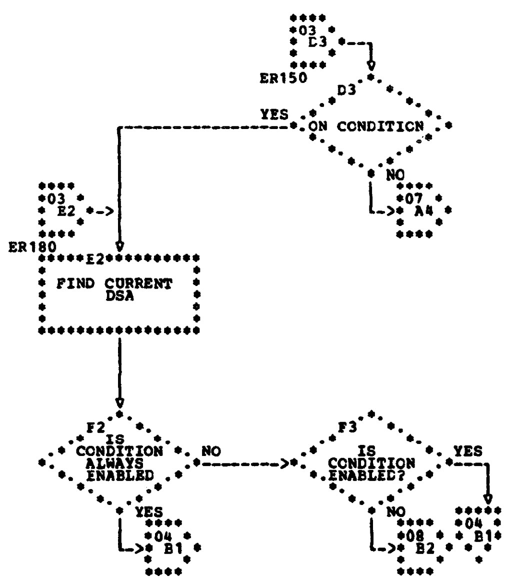


Chart DERR. Error handler (part 3 of 8)

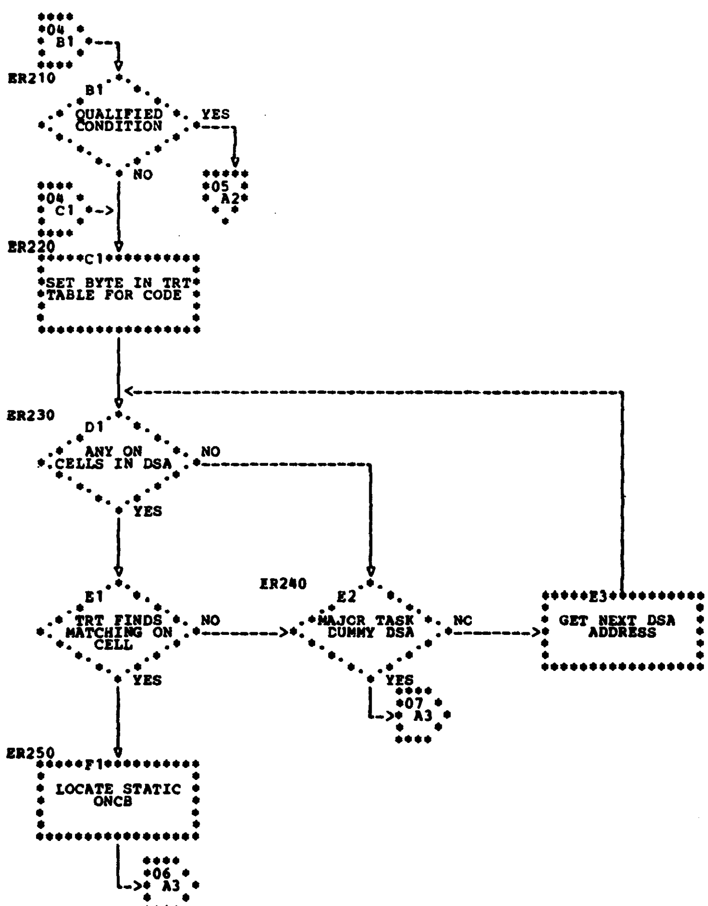


Chart DERR. Error handler (part 4 of 8)

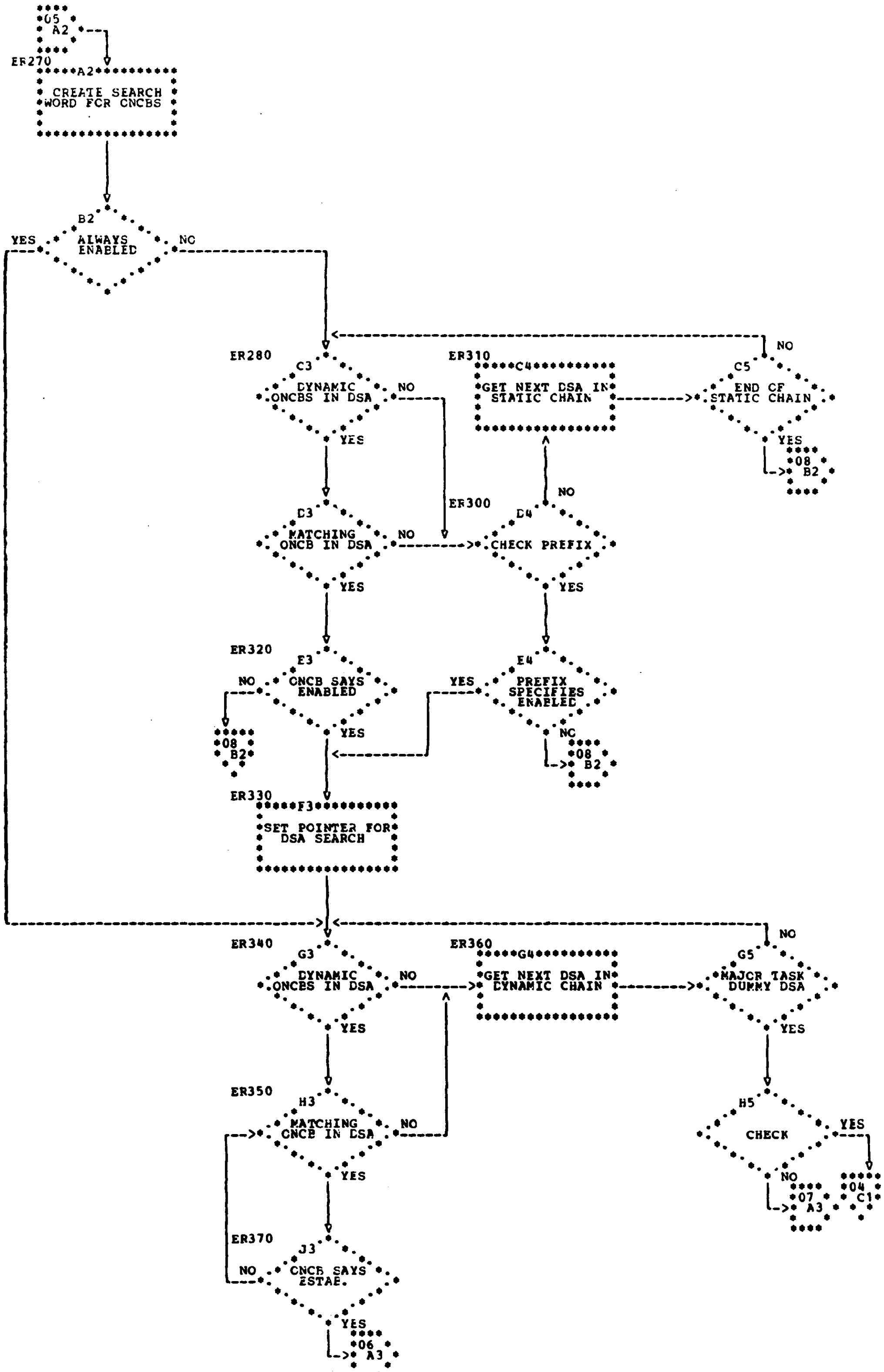


Chart DERR. Error handler (part 5 of 8)

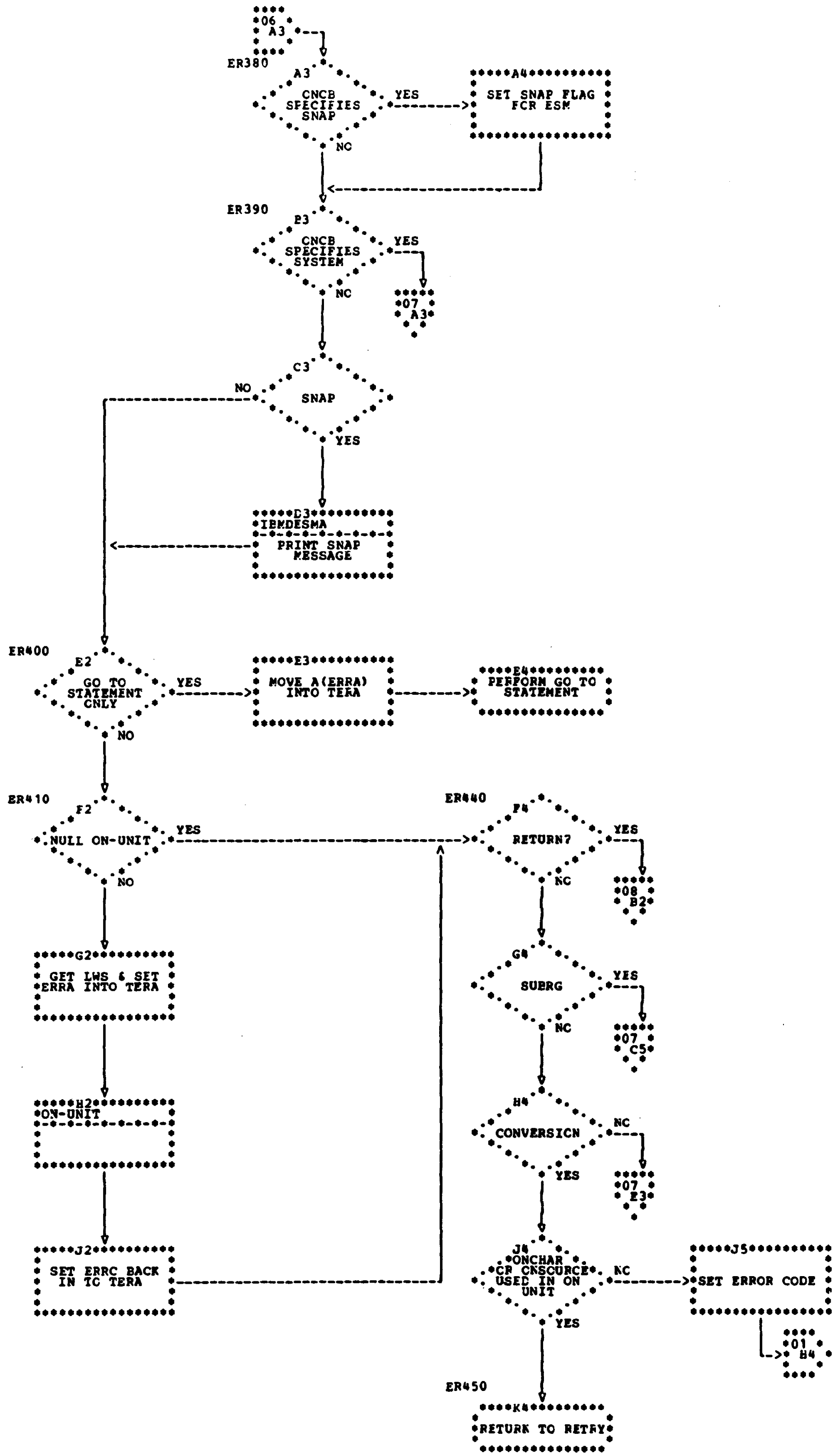


Chart DERR. Error handler (part 6 of 8)

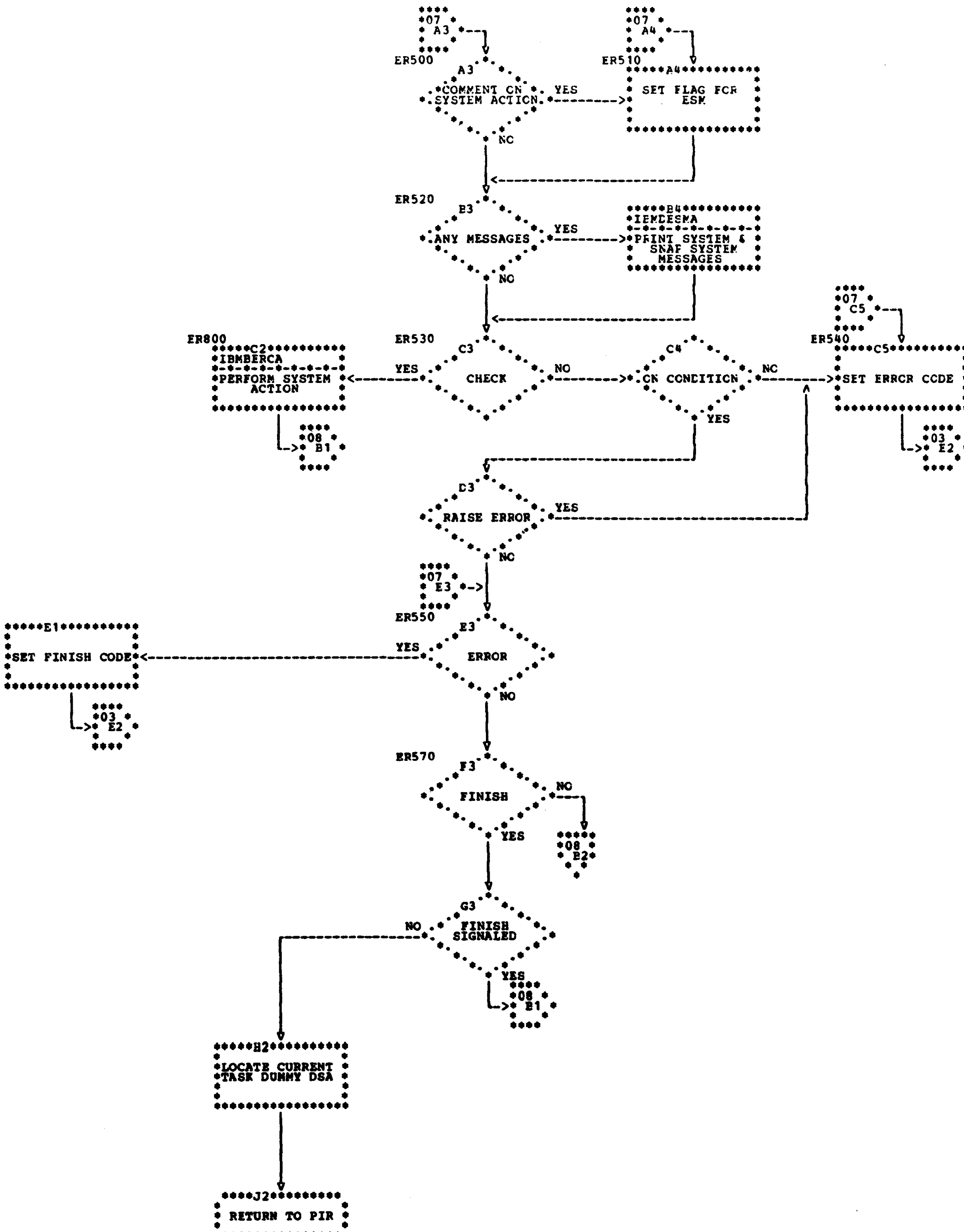


Chart DERR. Error handler (part 7 of 8)

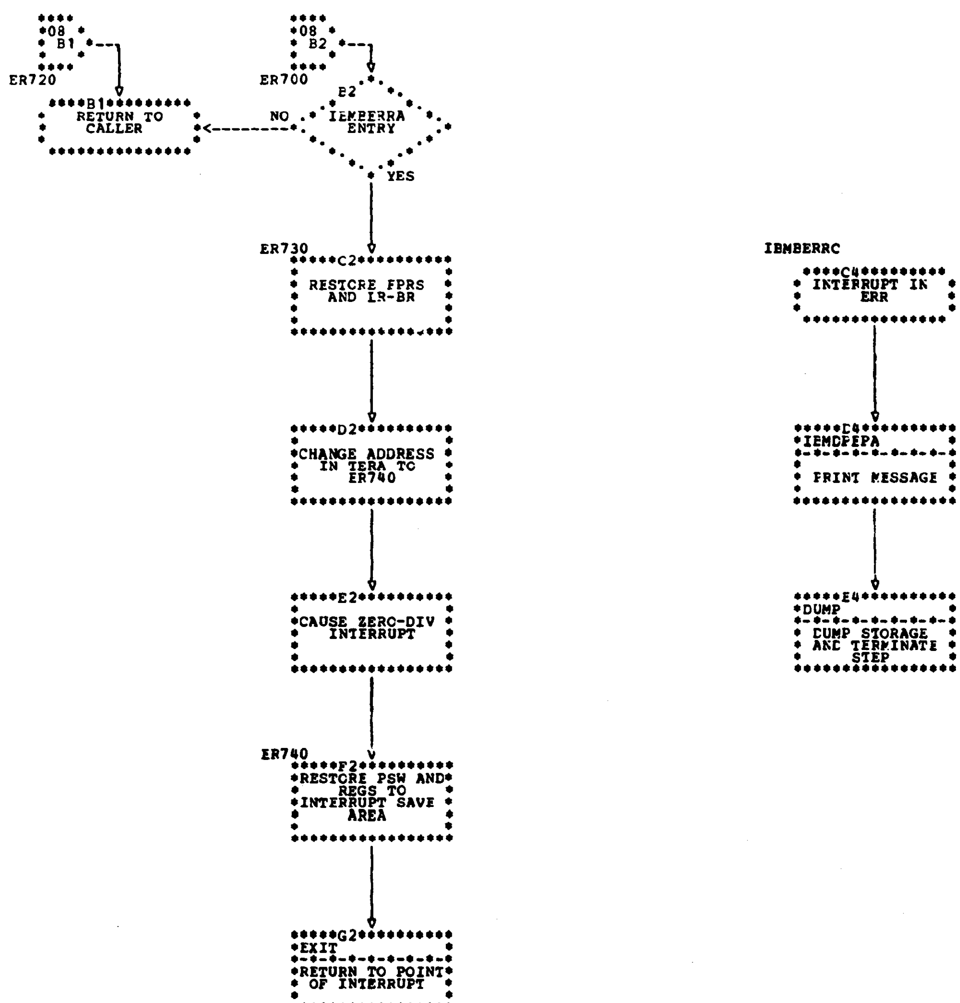


Chart DERR. Error handler (part 8 of 8)

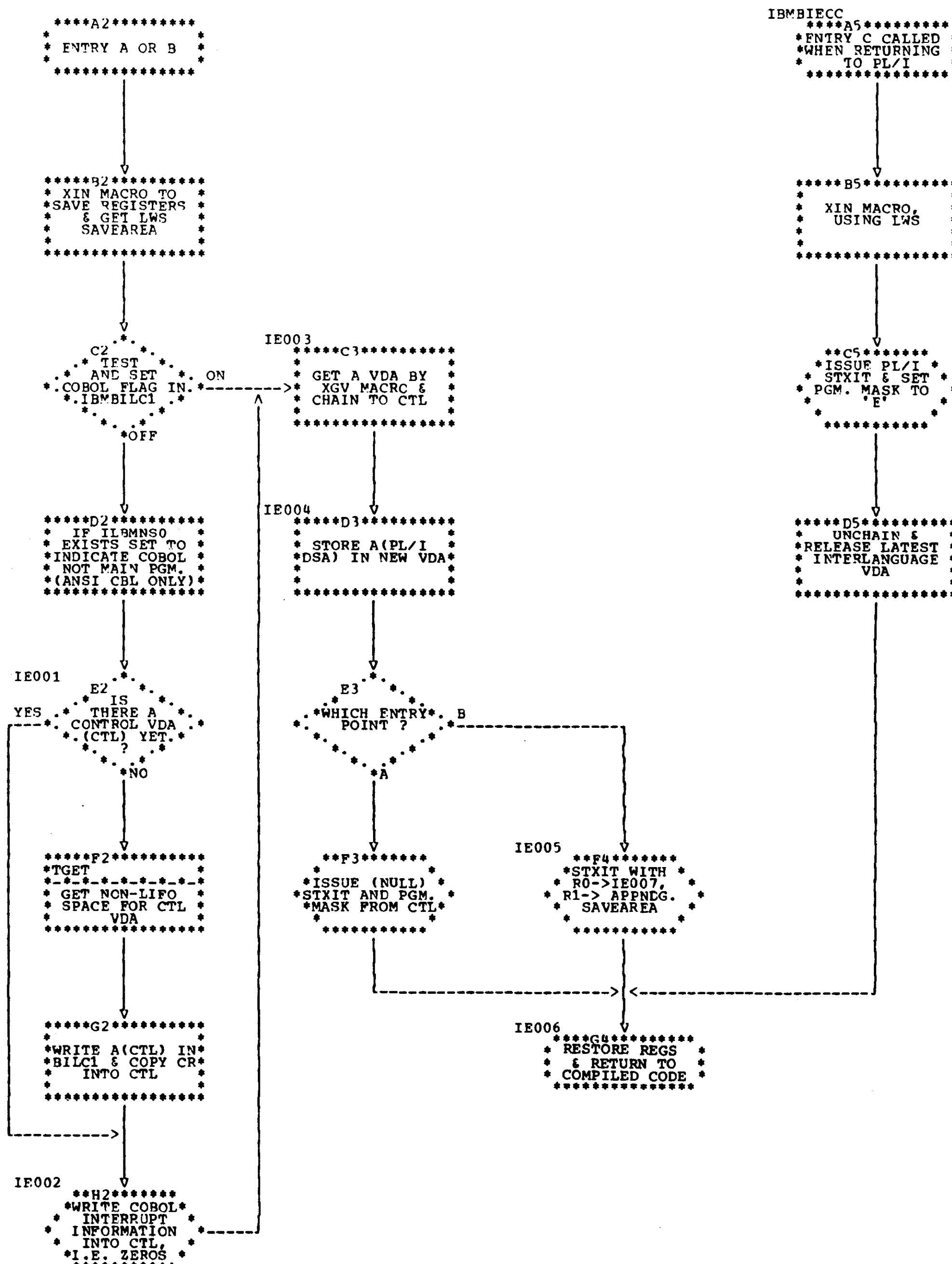


Chart DIEC. Interlanguage housekeeping (part 1 of 2)

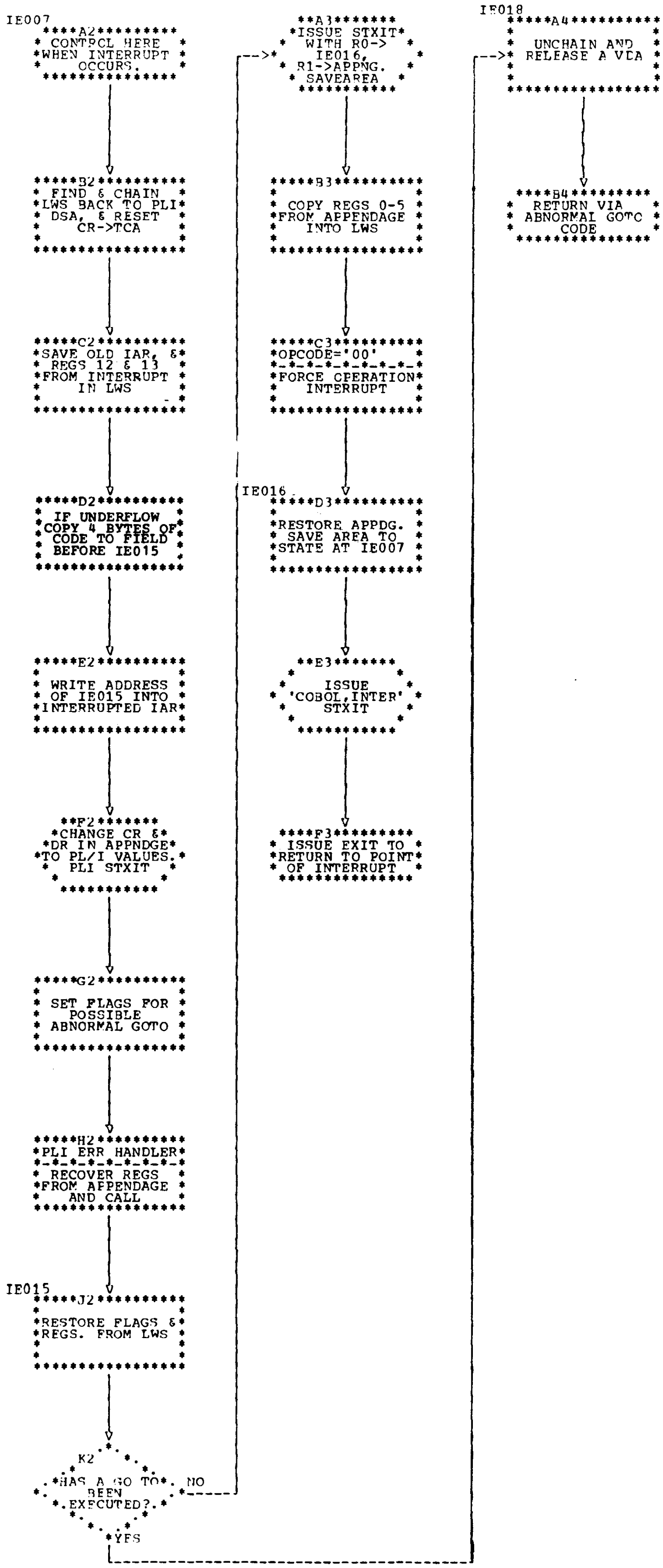


Chart DIEC. Interlanguage housekeeping (part 2 of 2)

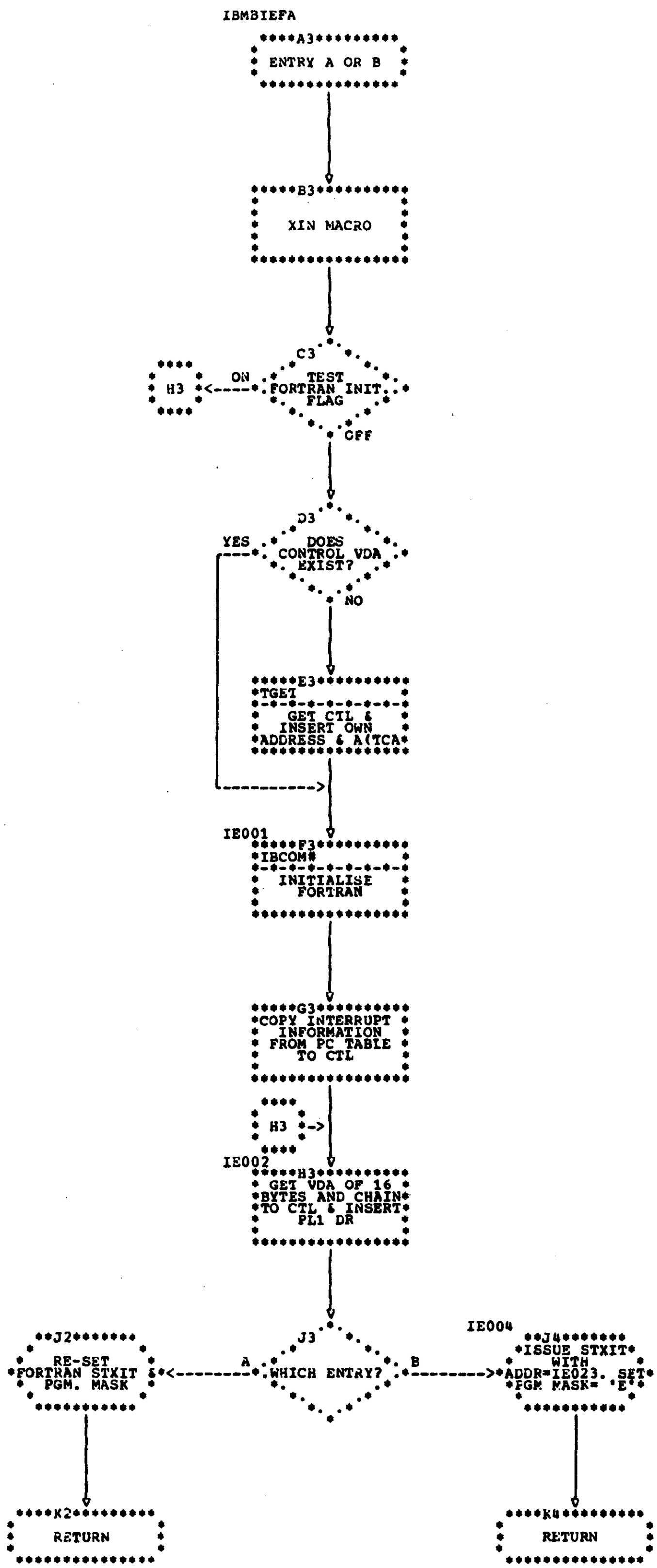


Chart DIEF. Interlanguage housekeeping (part 1 of 3)

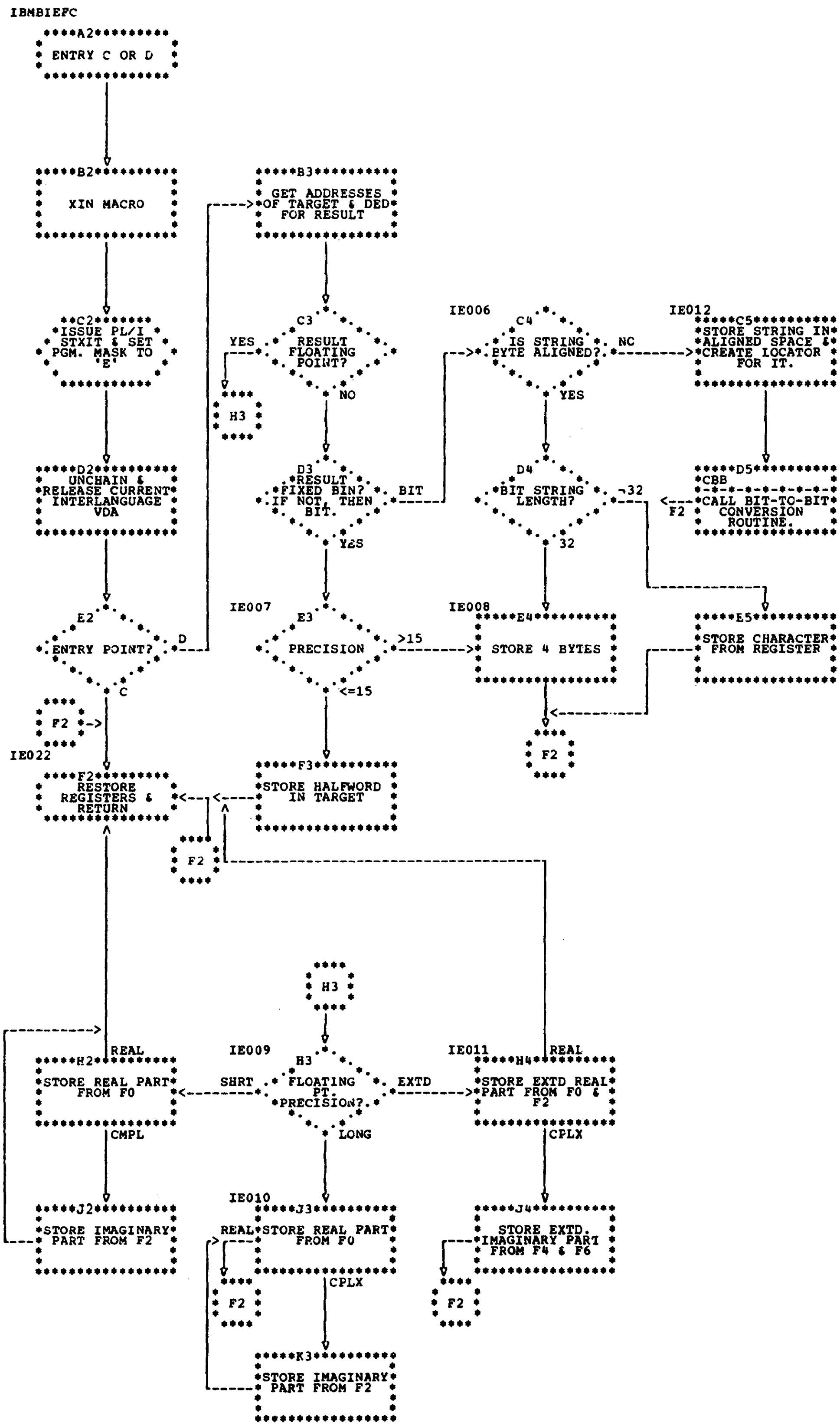


Chart DIEF. Interlanguage housekeeping (part 2 of 3)

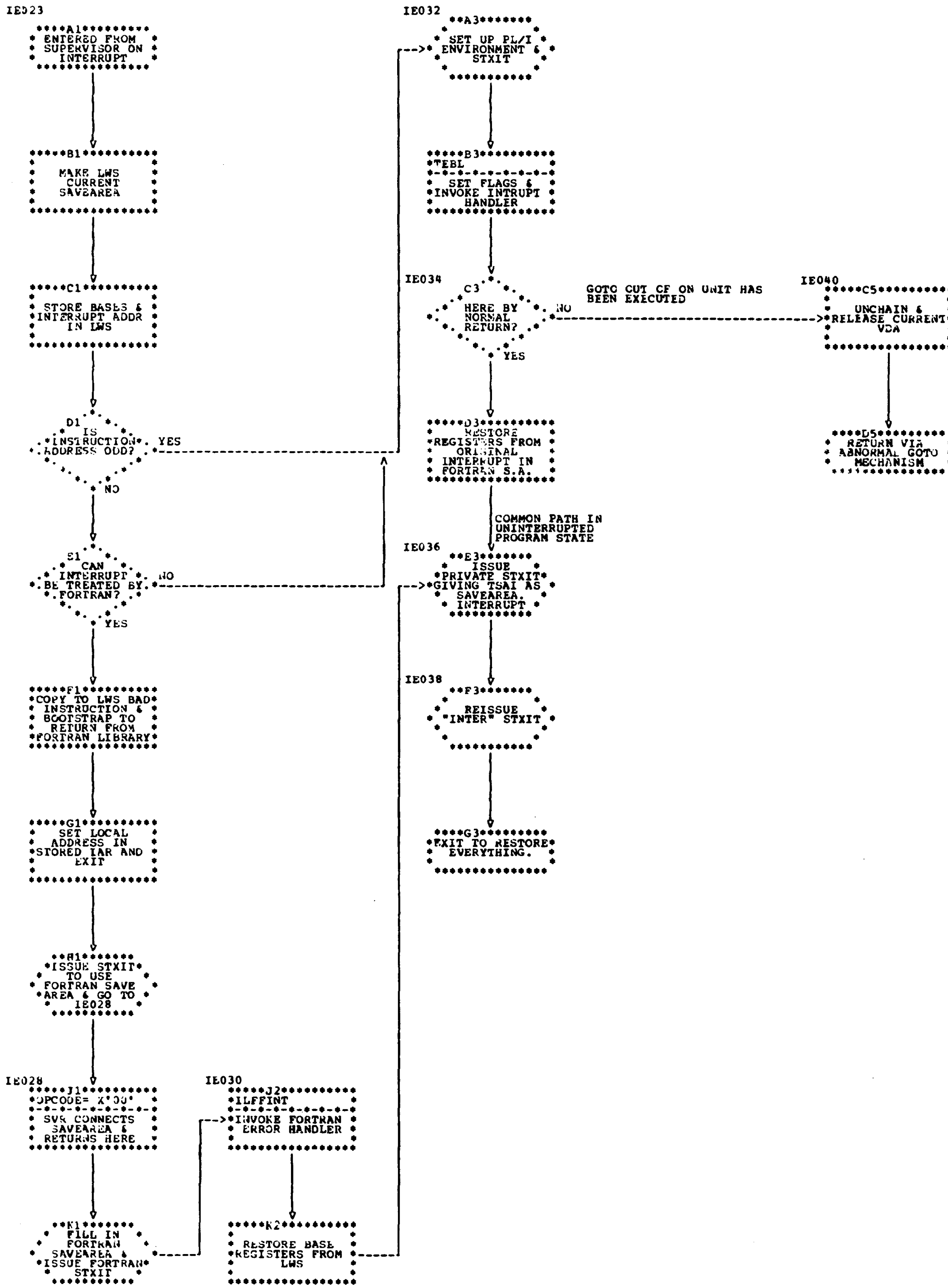


Chart DIEF. Interlanguage housekeeping (part 3 of 3)

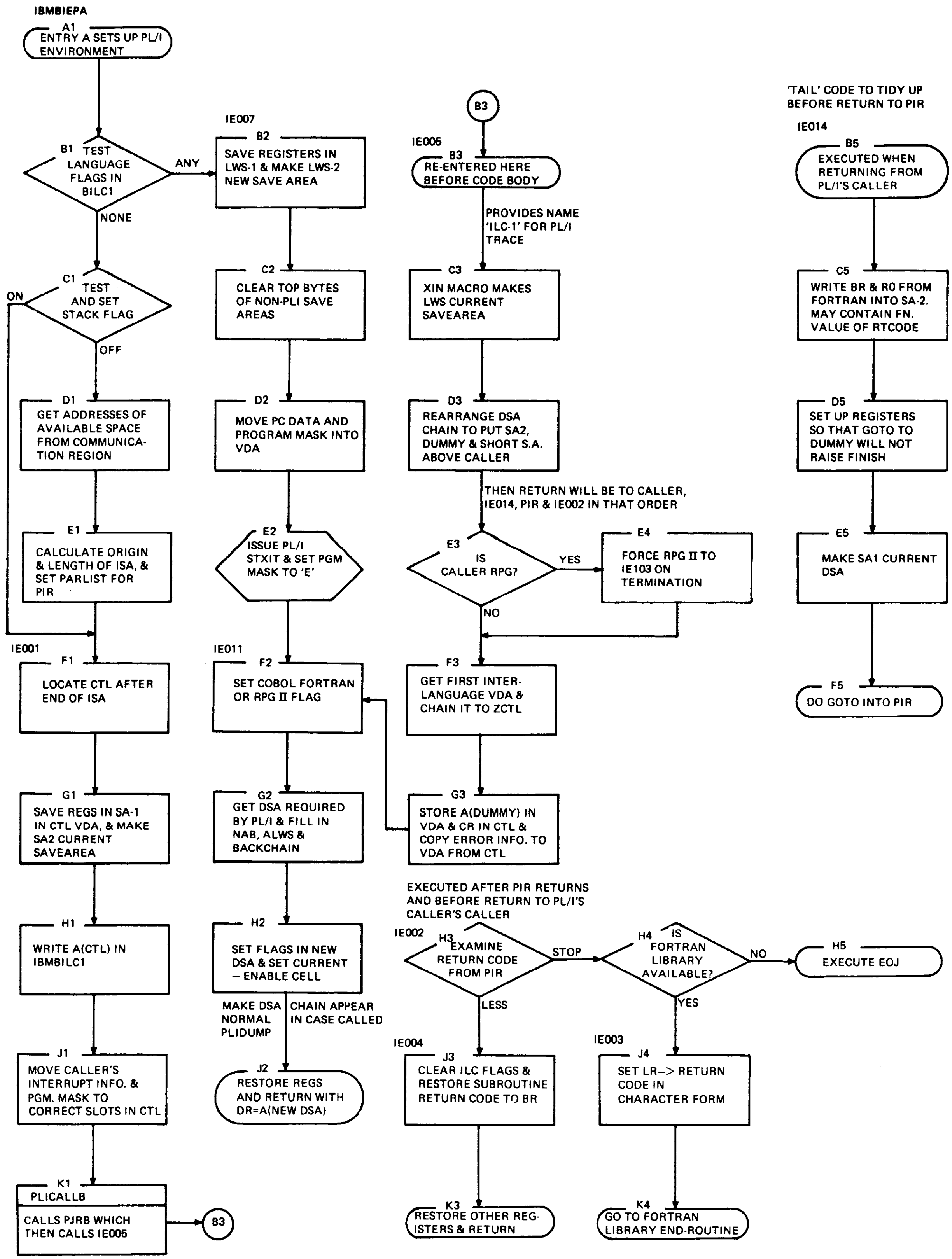
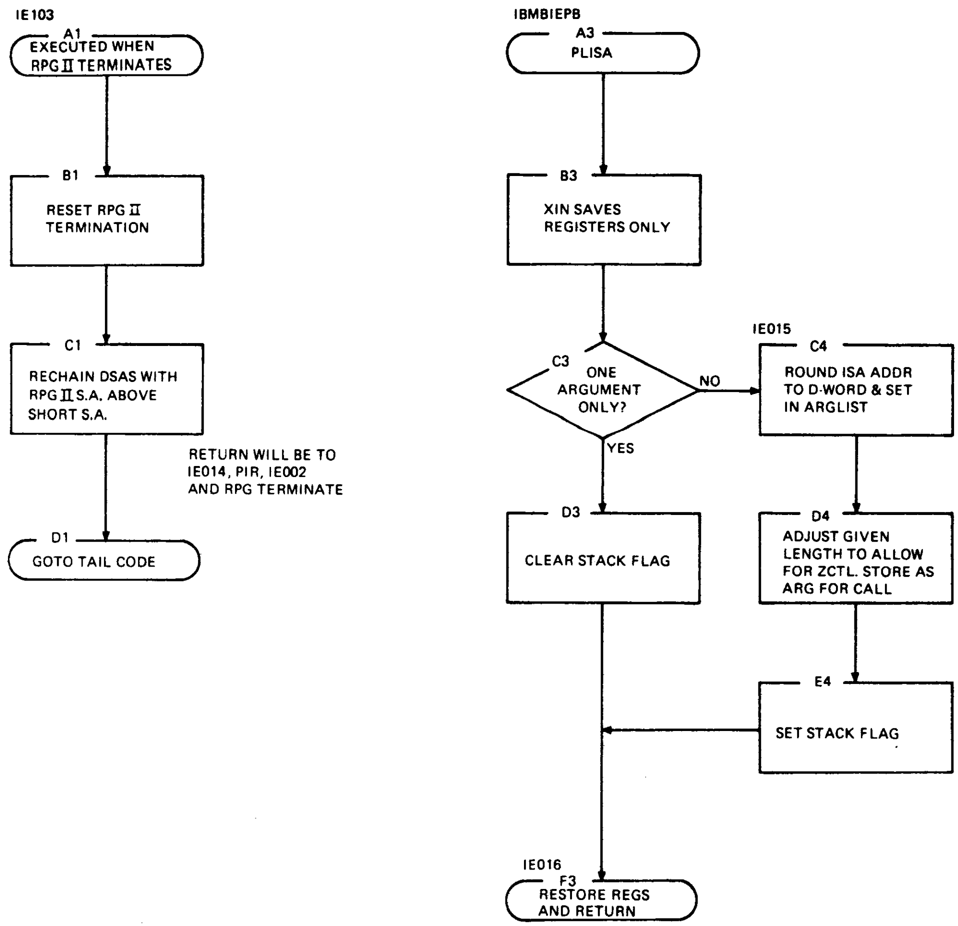


Chart DIEP. Interlanguage housekeeping (part 1 of 3)



|Chart DIEP. Interlanguage housekeeping (part 2 of 3)

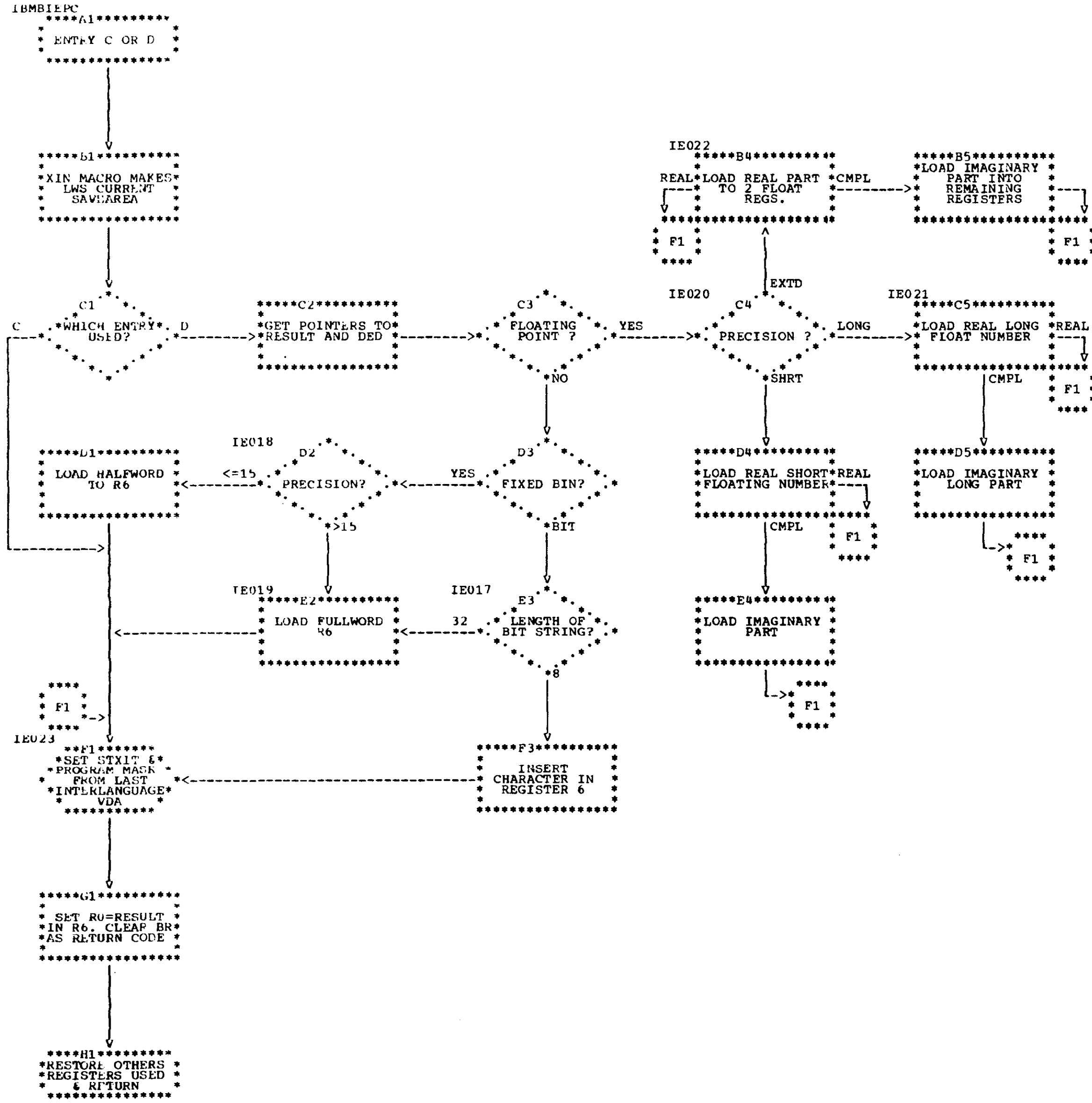


Chart DIEP. Interlanguage housekeeping (part 3 of 3)

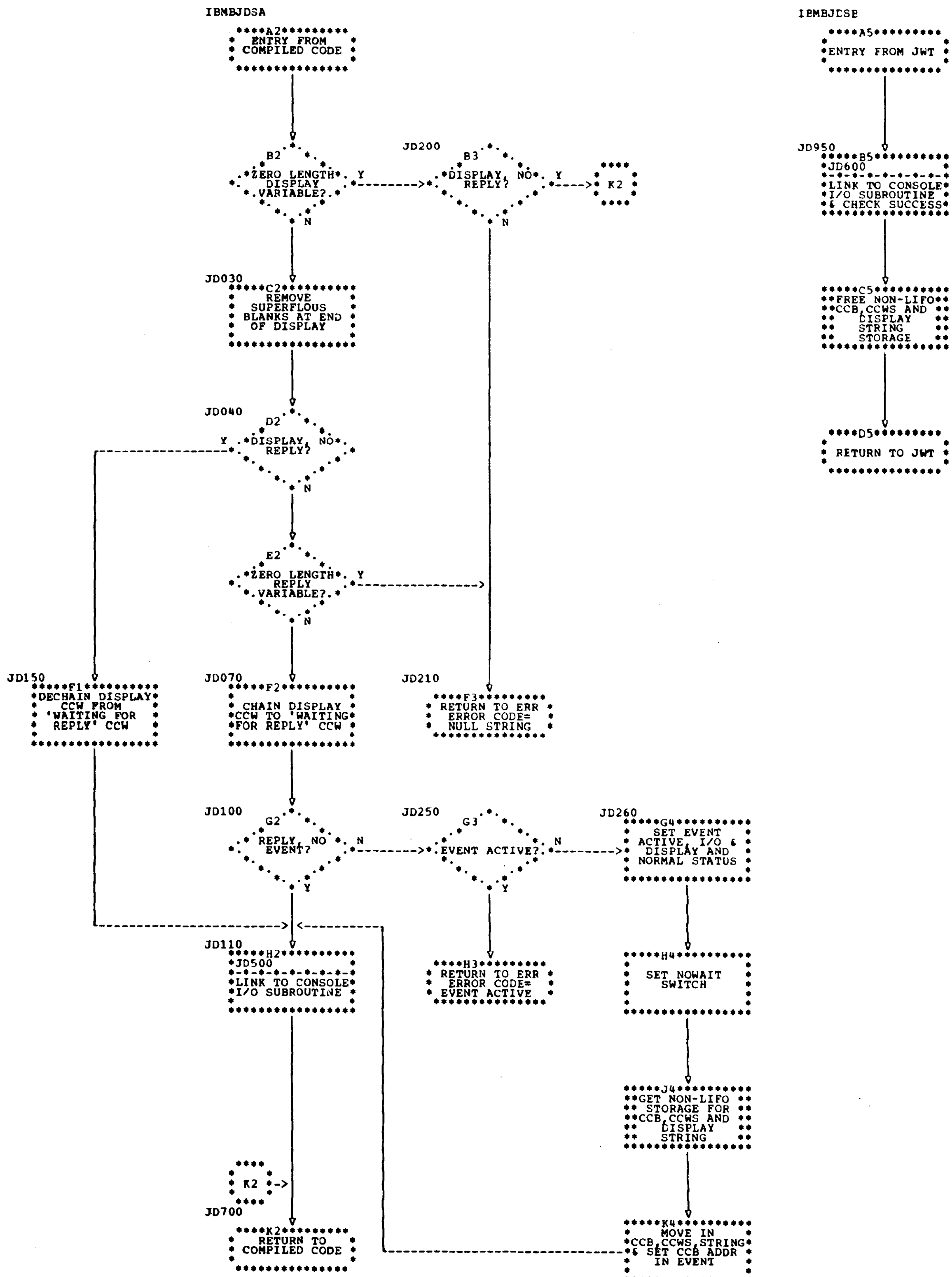


Chart DJDS. DISPLAY module (part 1 of 2)

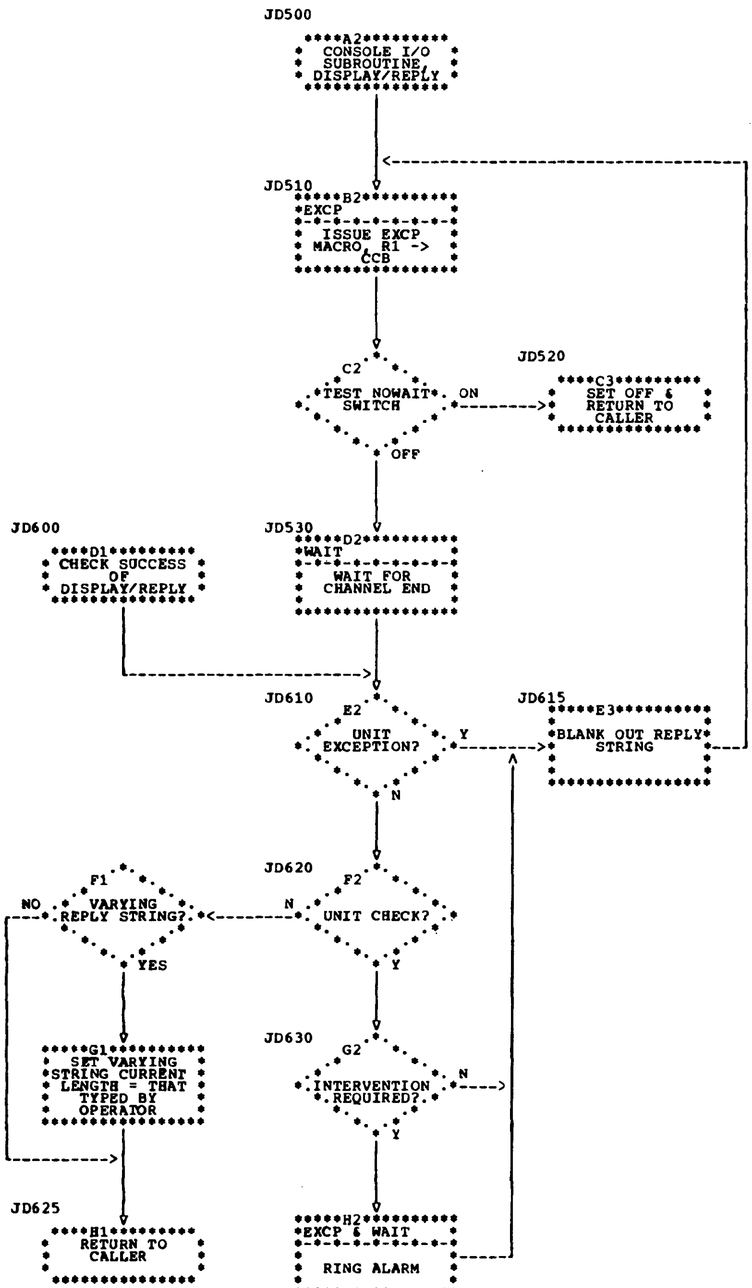


Chart DJDS. DISPLAY module (part 2 of 2)

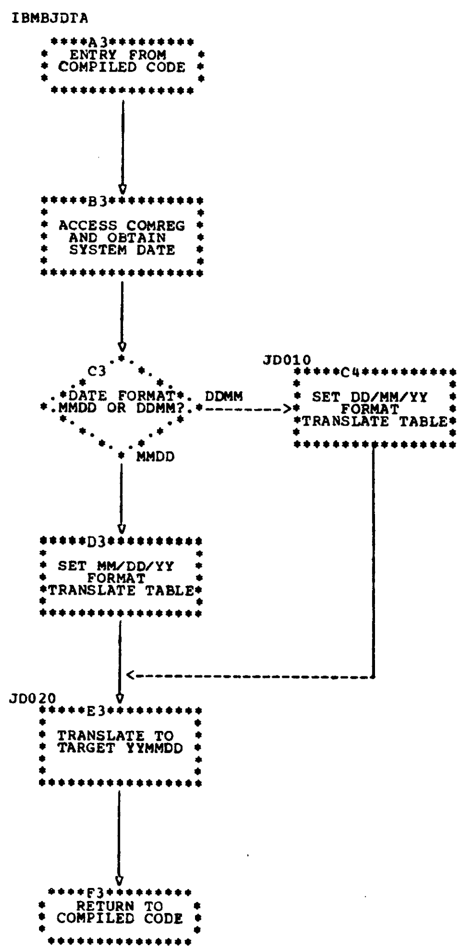


Chart DJDT. DATE built-in function

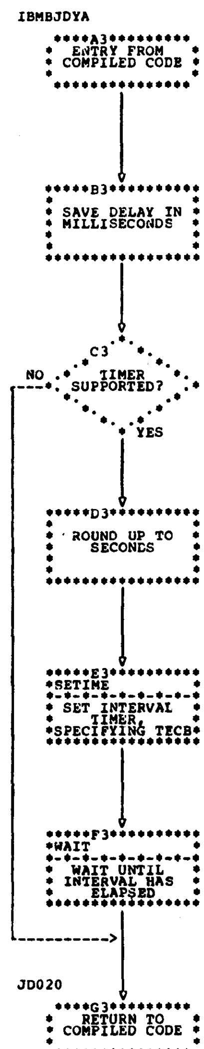


Chart DJDY. DELAY module

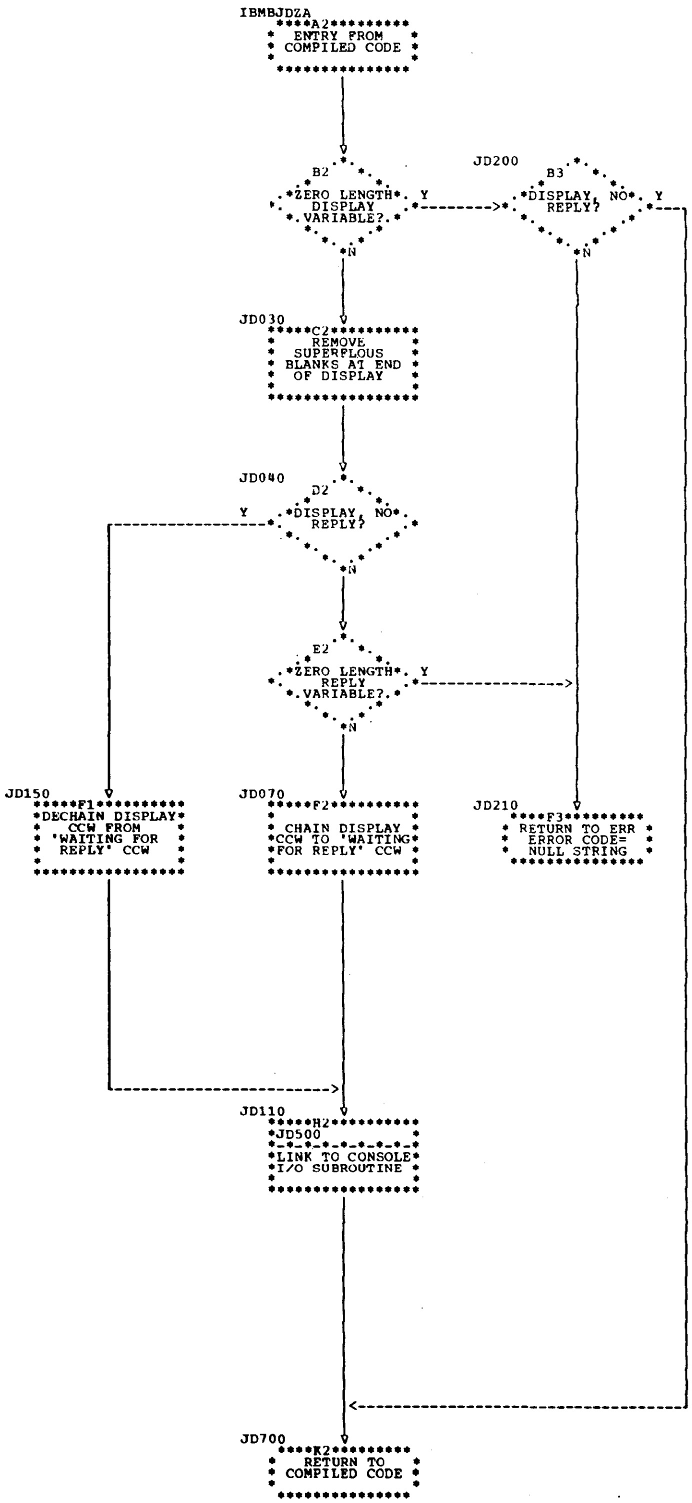


Chart DJDZ. DISPLAY without EVENT (part 1 of 2)

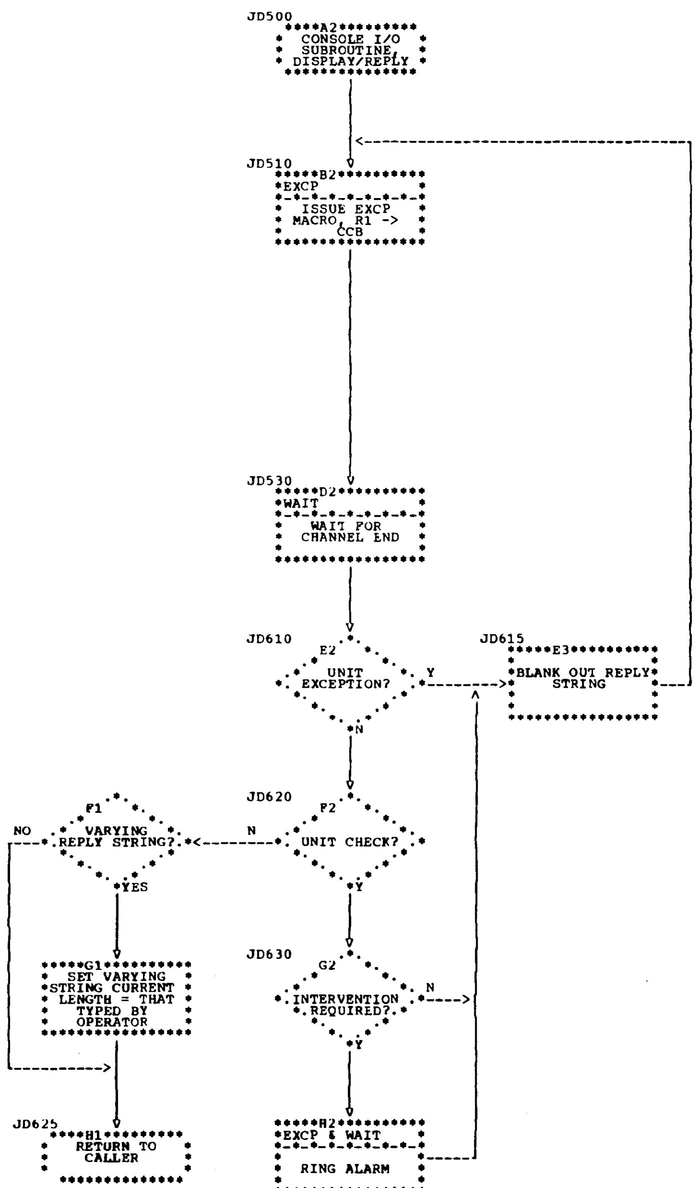


Chart DJDZ. DISPLAY without EVENT (part 2 of 2)

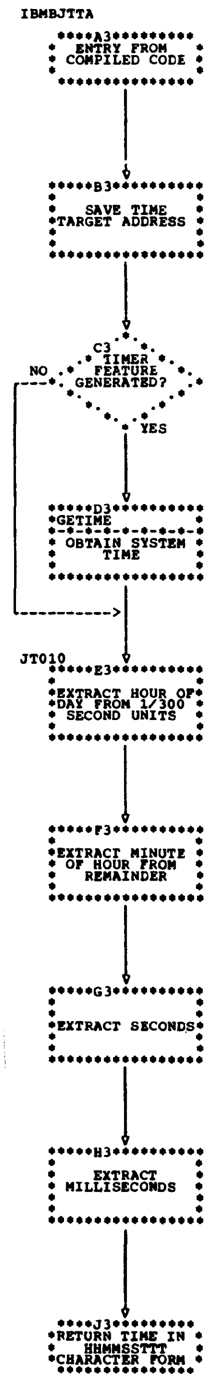


Chart DJTT. TIME built-in function

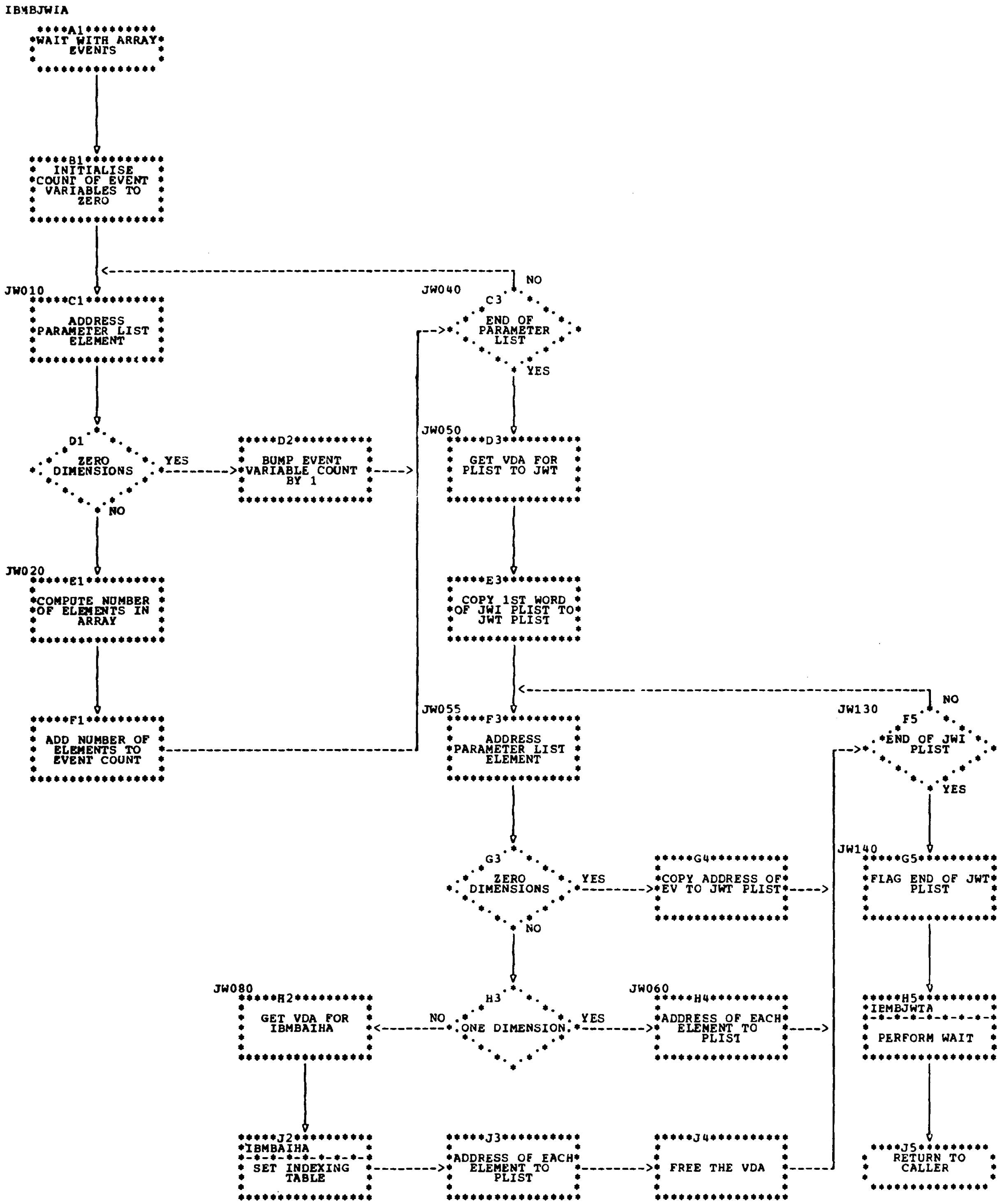


Chart BJWI. WAIT (array event variables)

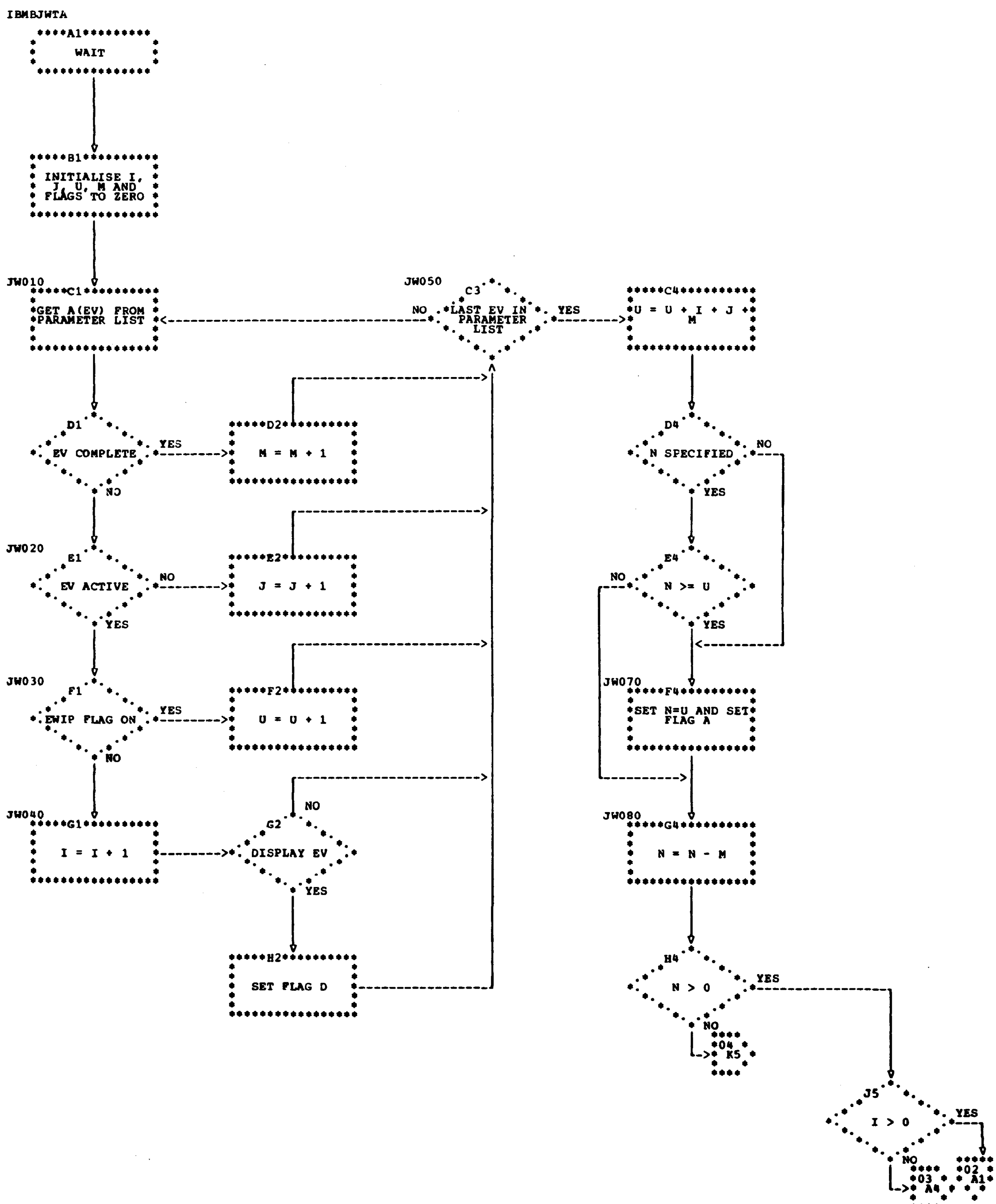


Chart DJWT. WAIT (multiple events) (part 1 of 5)

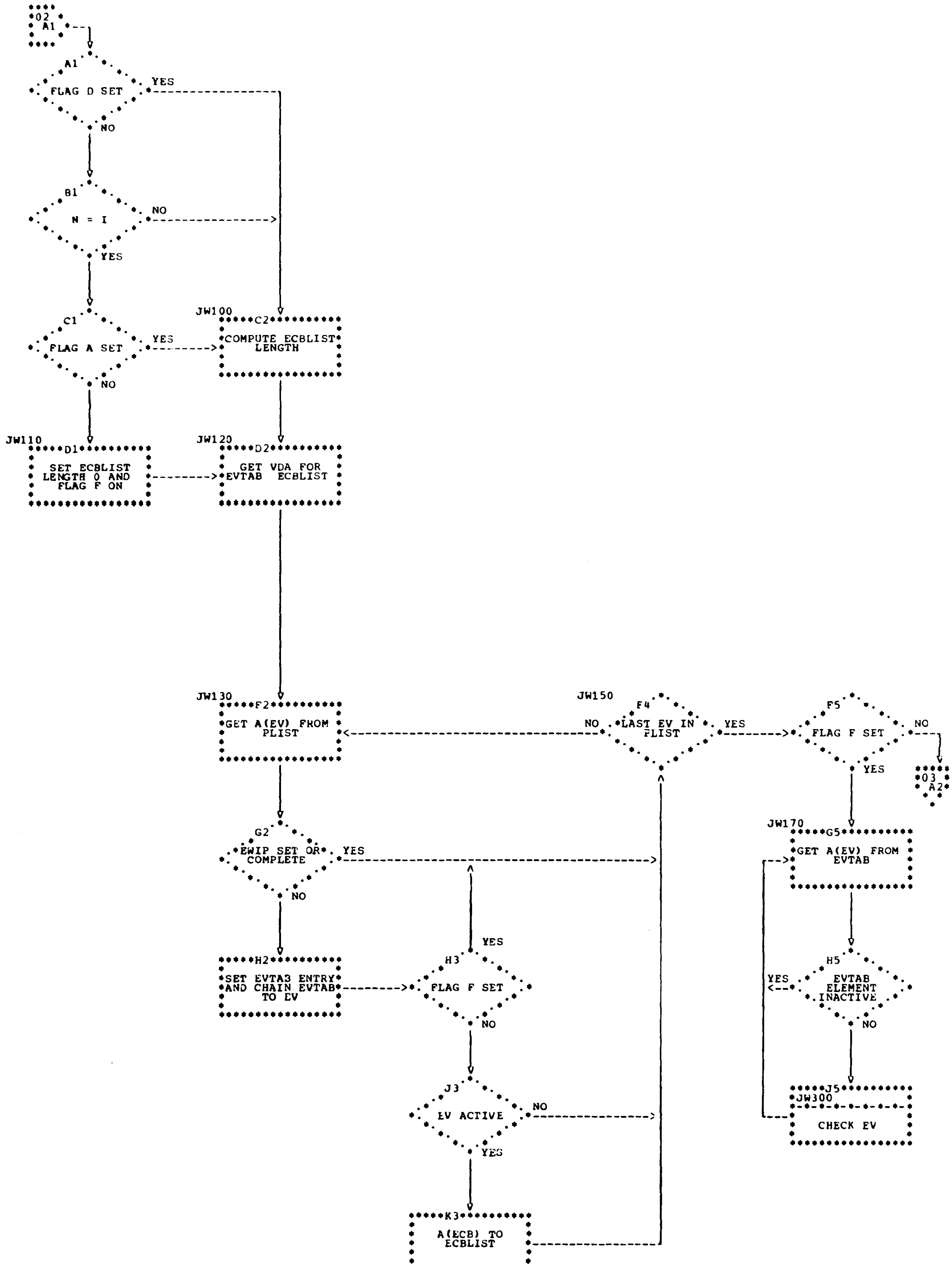


Chart DJWT. WAIT (multiple events) (part 2 of 5)

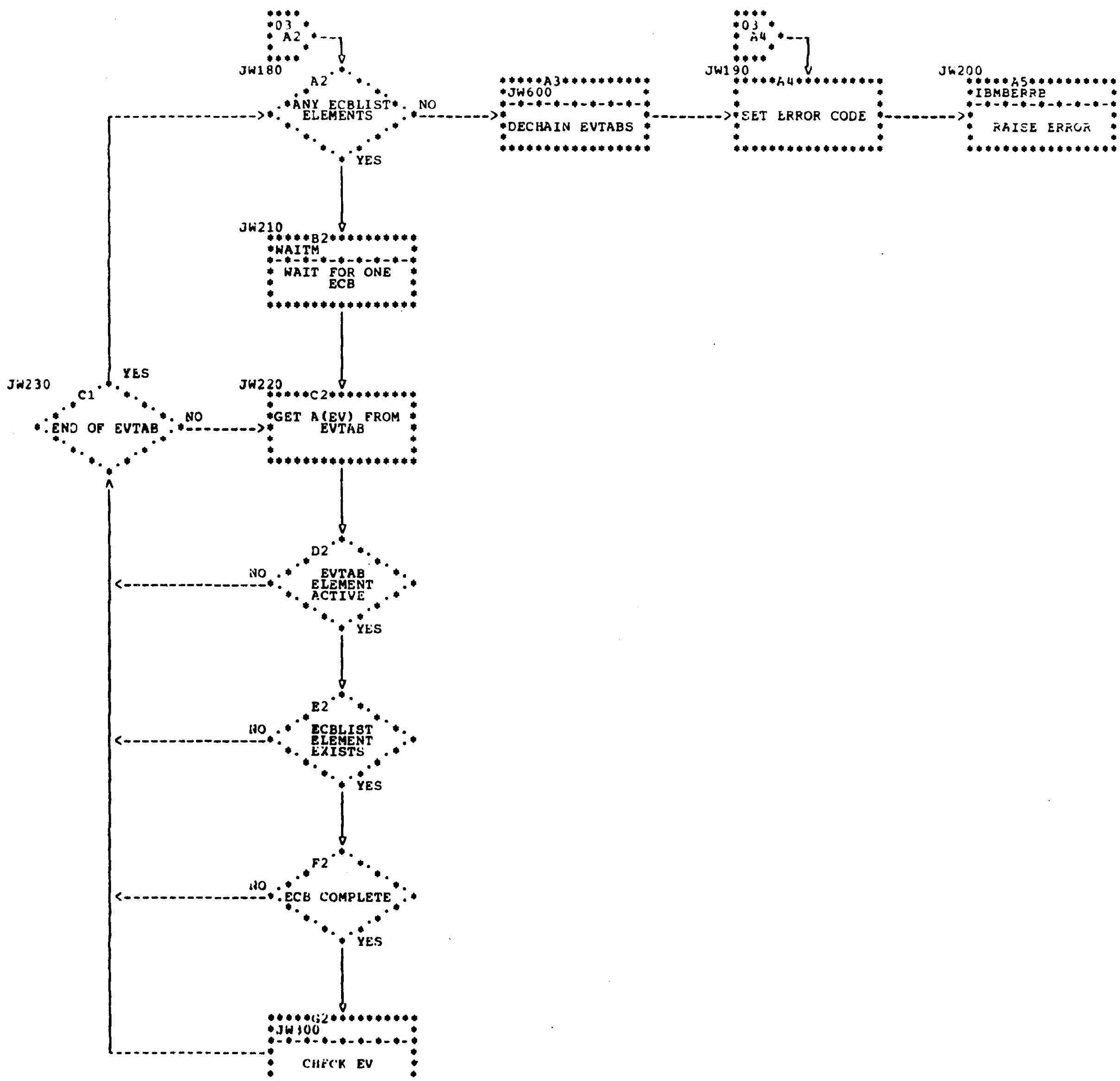


Chart DJWT. WAIT (multiple events) (part 3 of 5)

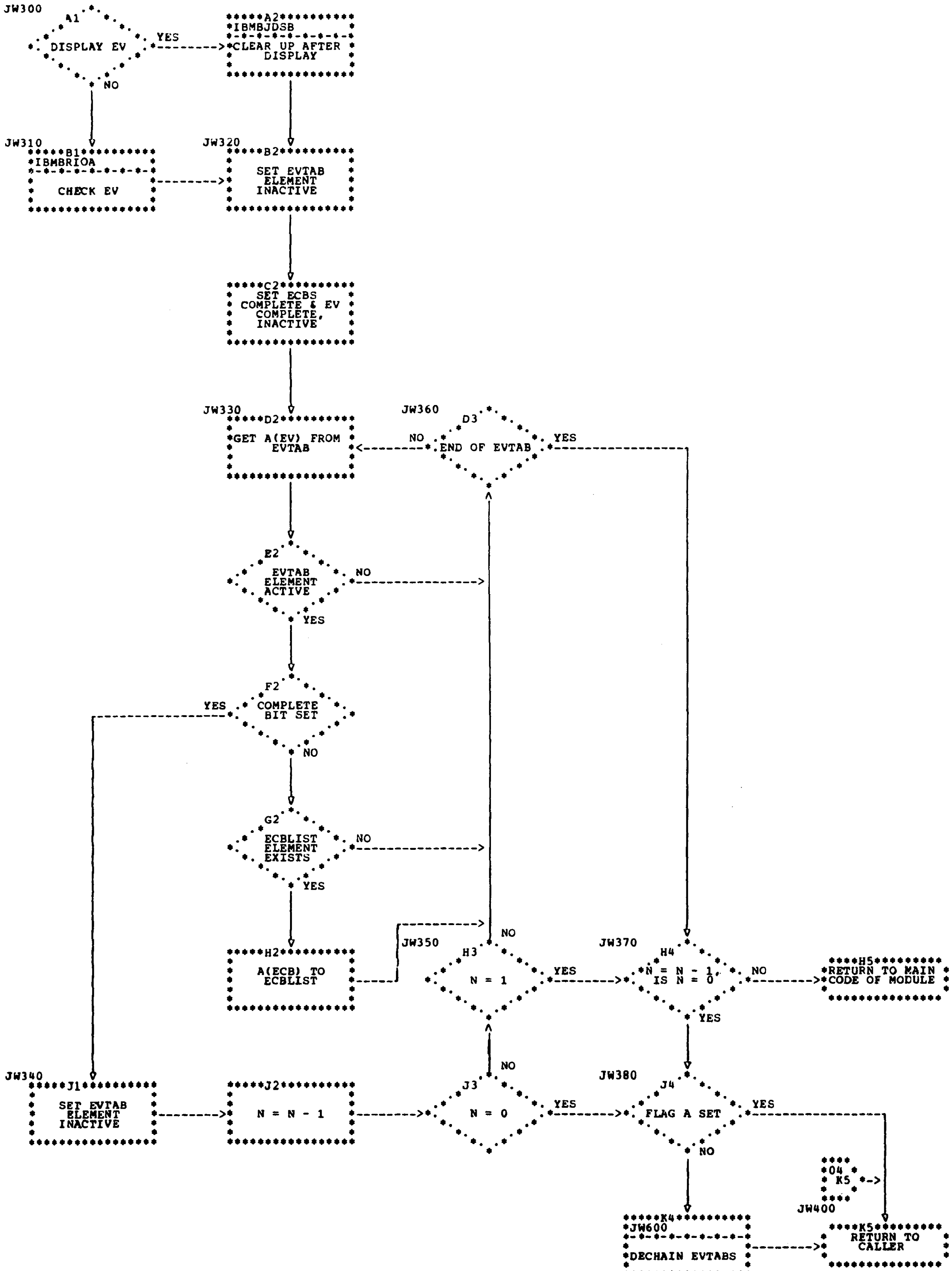


Chart DJWT. WAIT (multiple events) (part 4 of 5)

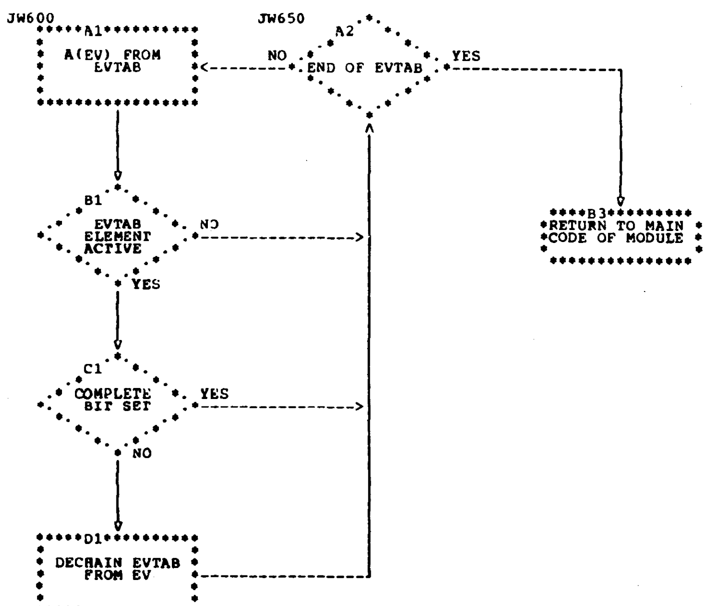


Chart DJWT. WAIT (multiple events) (part 5 of 5)

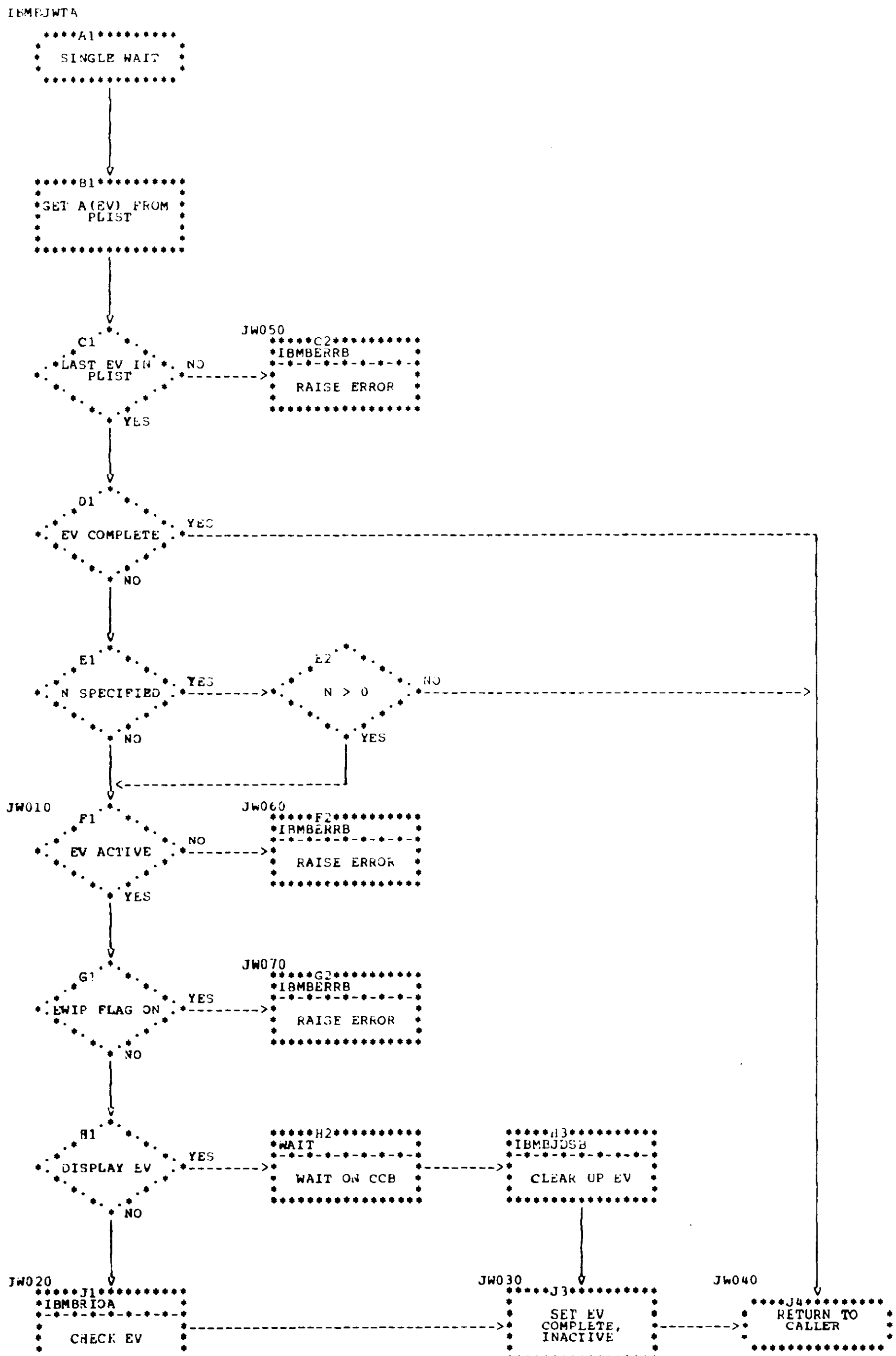


Chart GJWT. WAIT (single event)

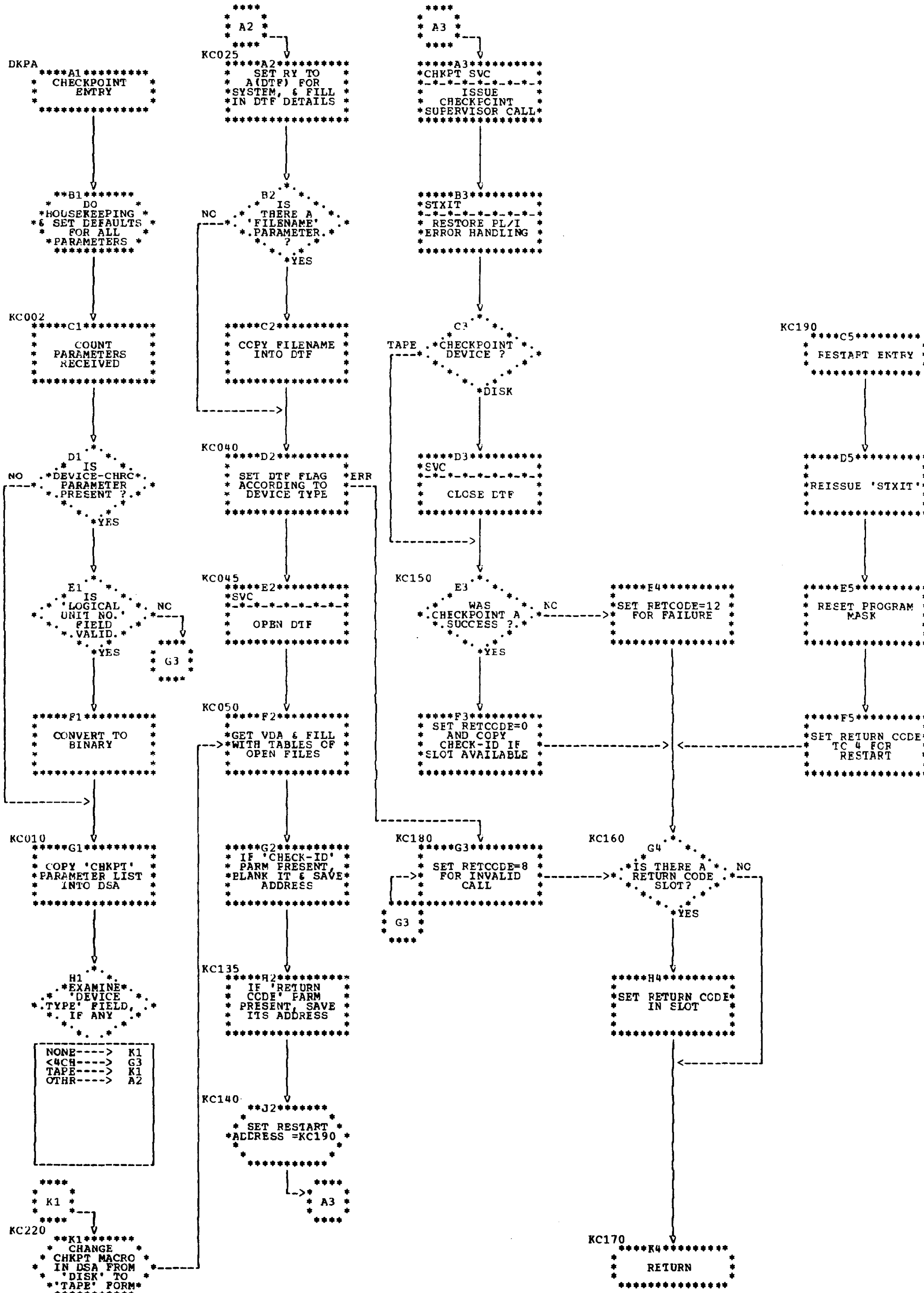


Chart DKCP. CHECKPOINT/RESTART interface

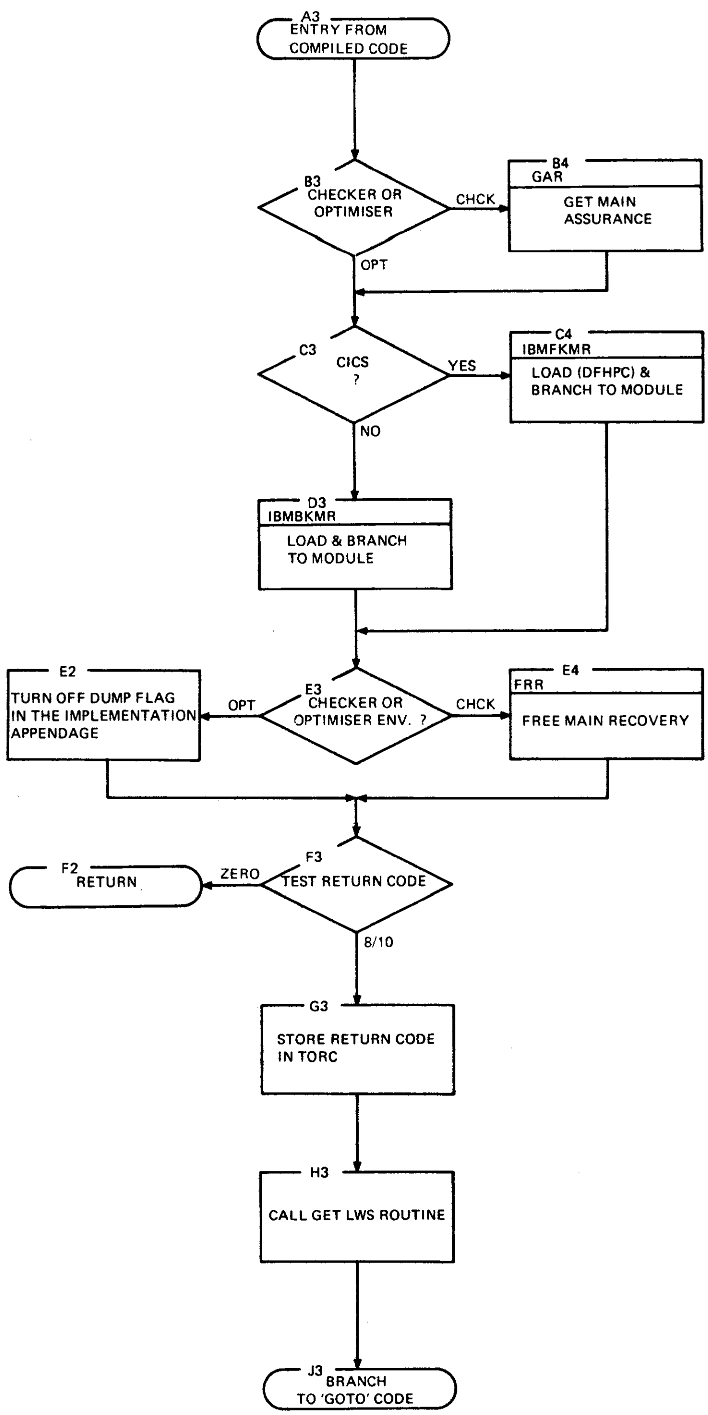


Chart DKDM. Dump bootstrap

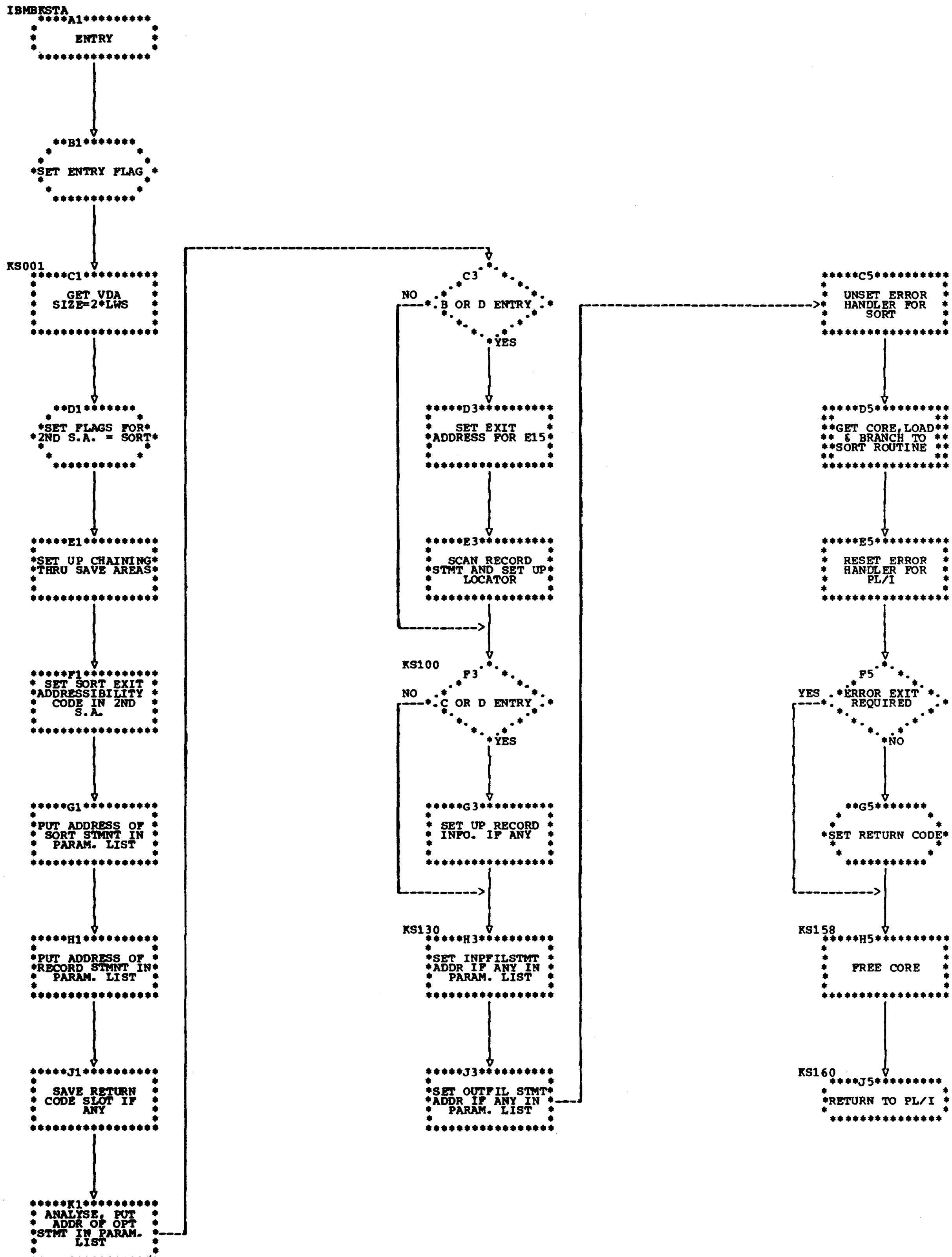


Chart DKST. SORT utility interface (part 1 of 2)

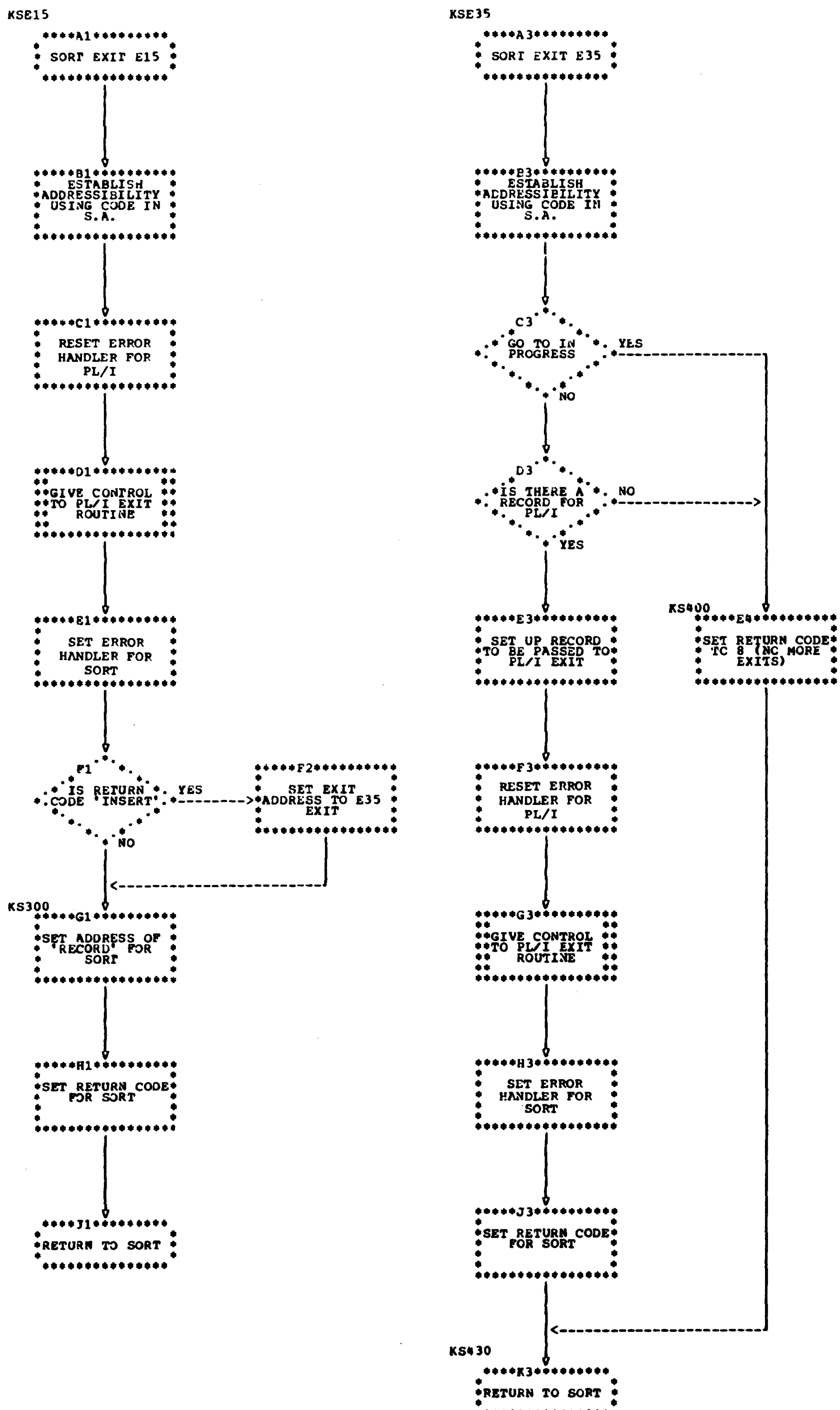


Chart DKST. SORT utility interface (part 2 of 2)

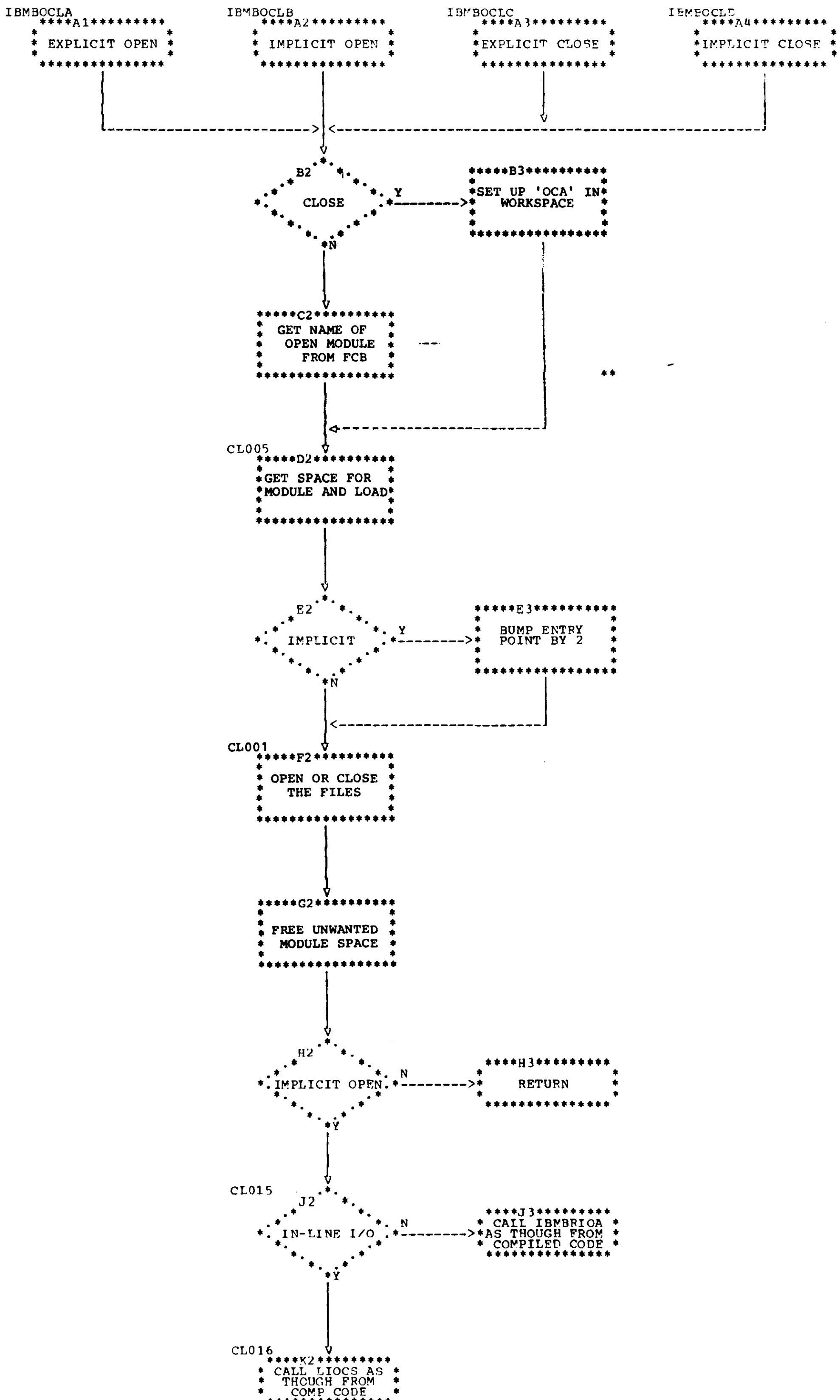


Chart DOCL. Open/close bootstrap

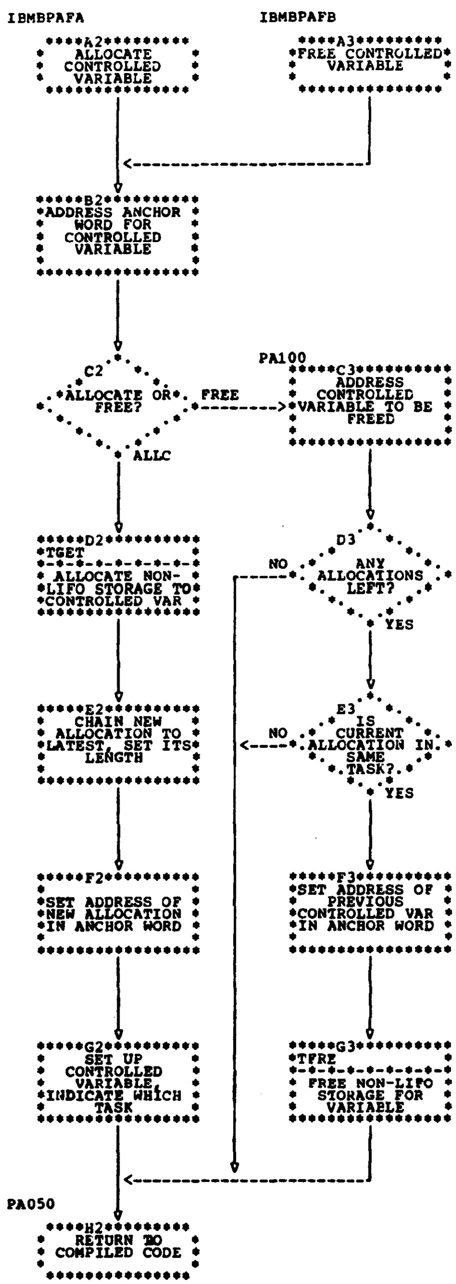


Chart BPAF. Controlled variable management

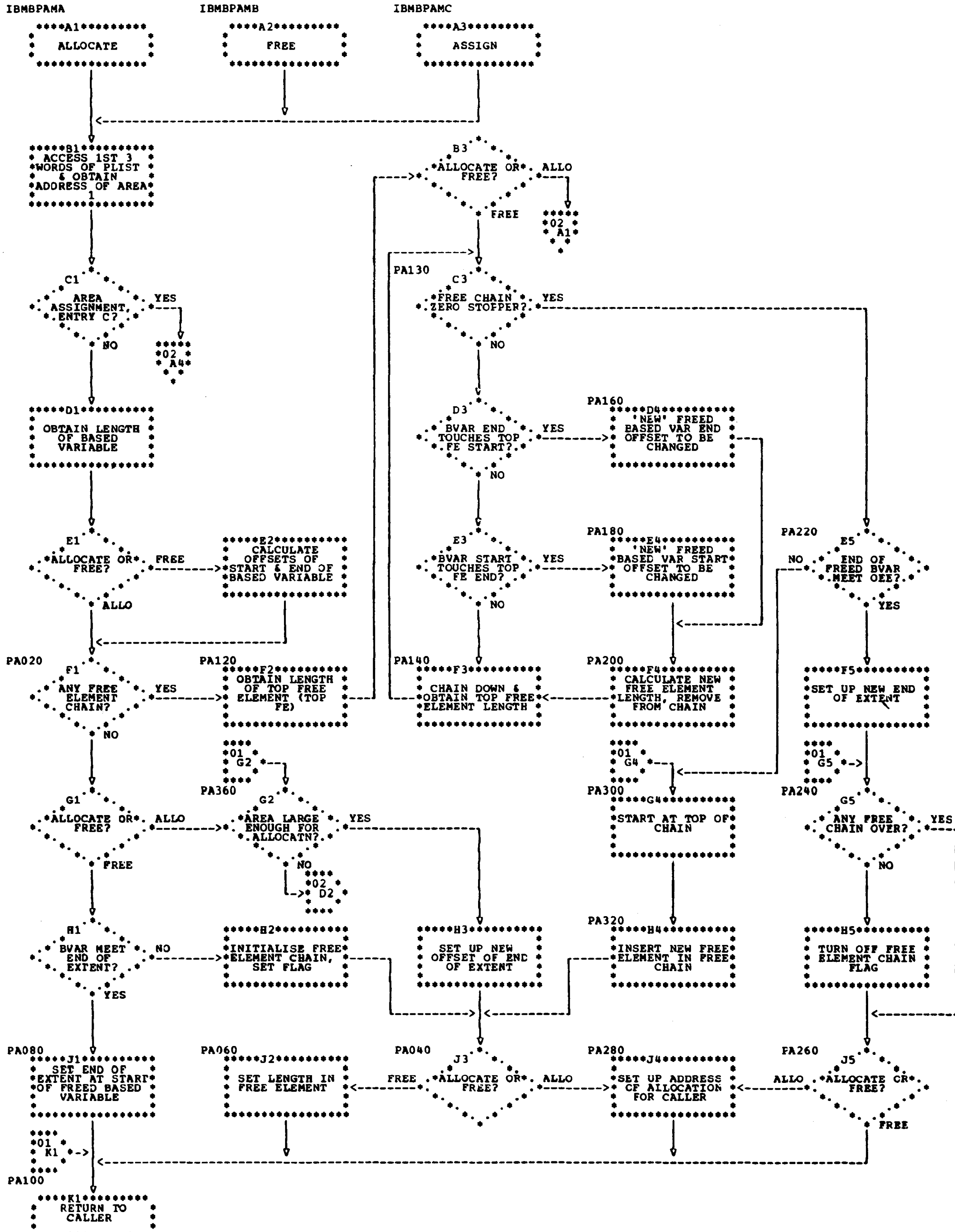


Chart BPAM. Area management (part 1 of 2)

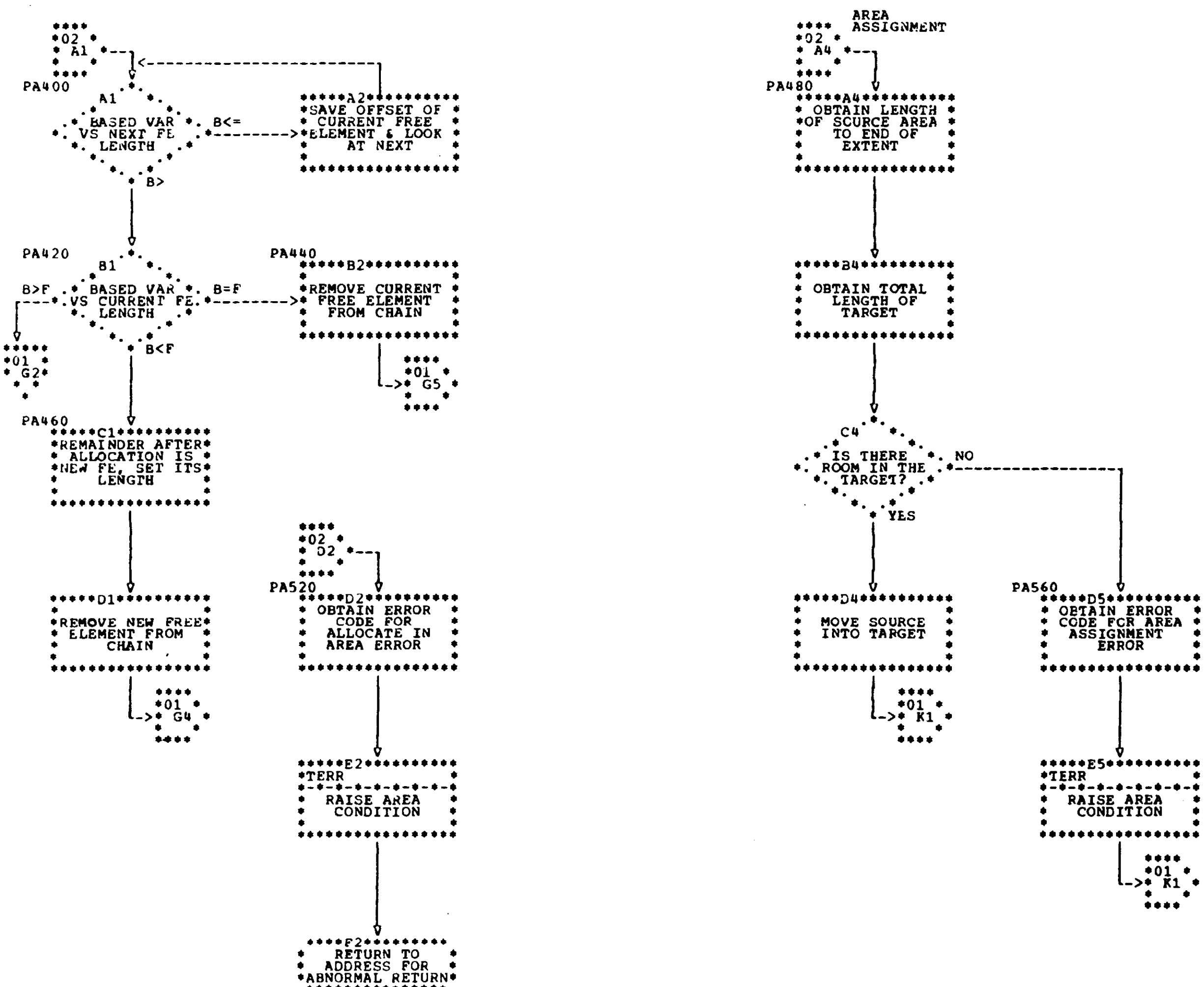


Chart BPAM. Area management (part 2 of 2)

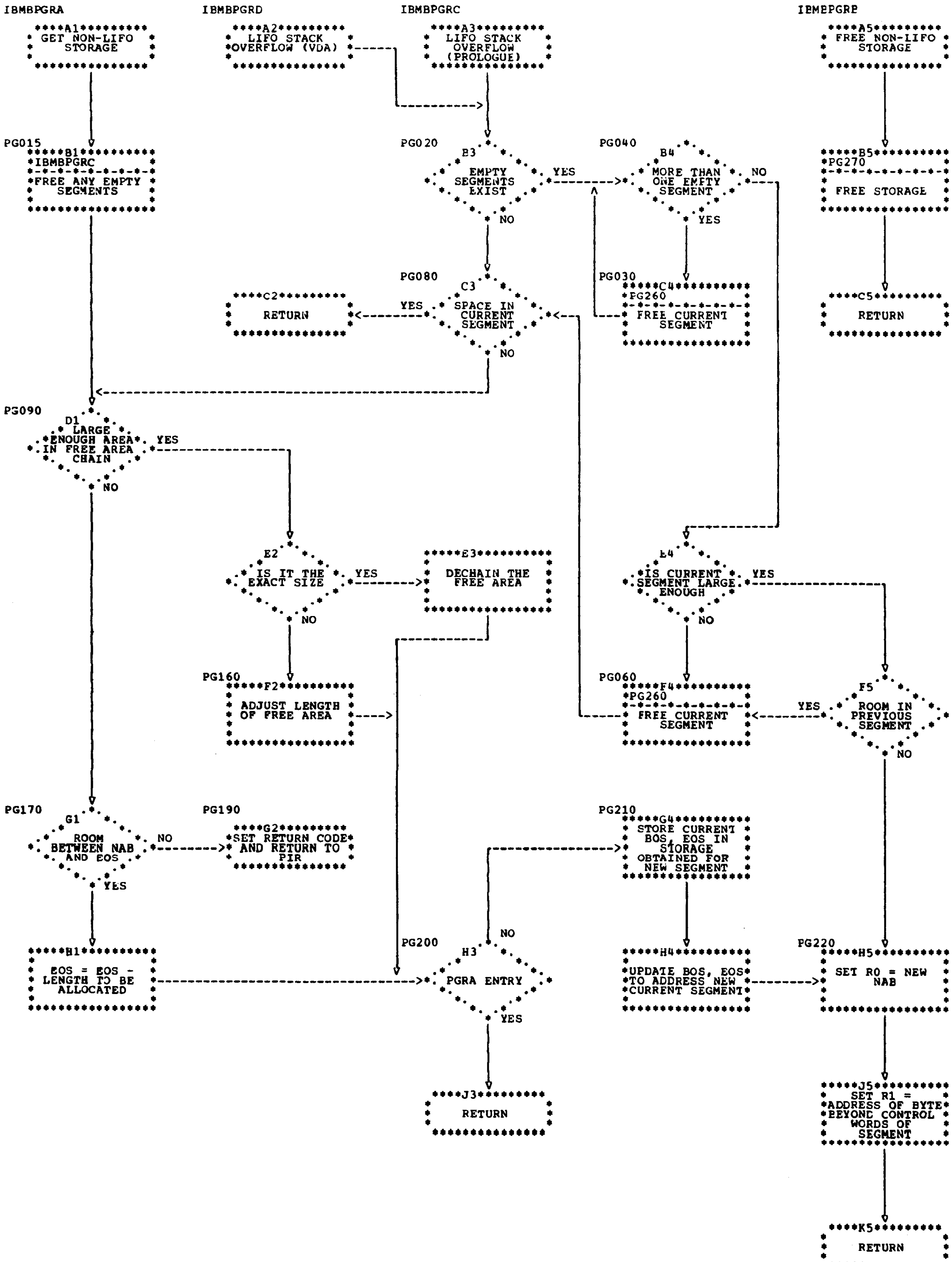


Chart DPGR. Storage management (part 1 of 2)

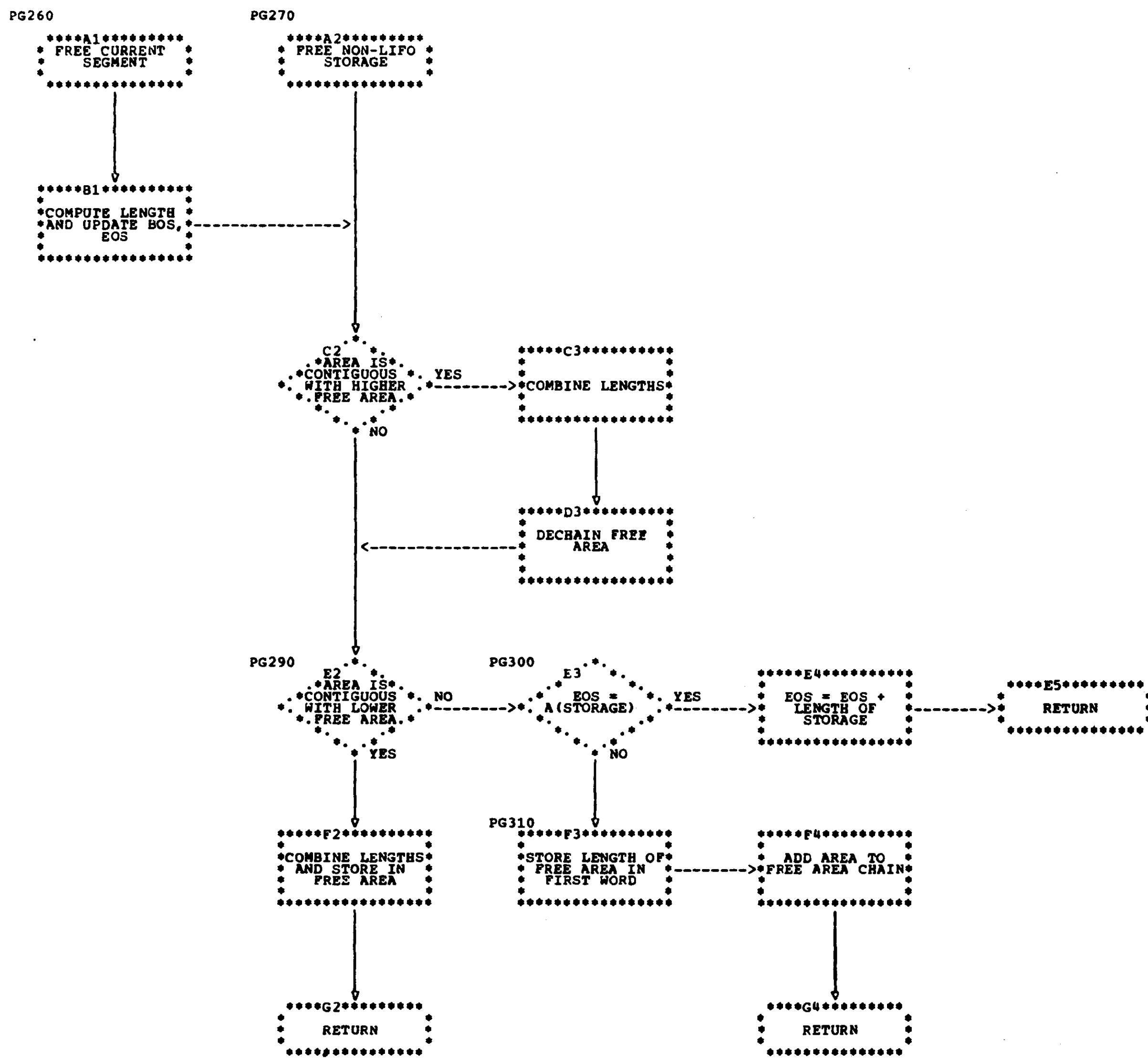


Chart DPGR. Storage management (part 2 of 2)

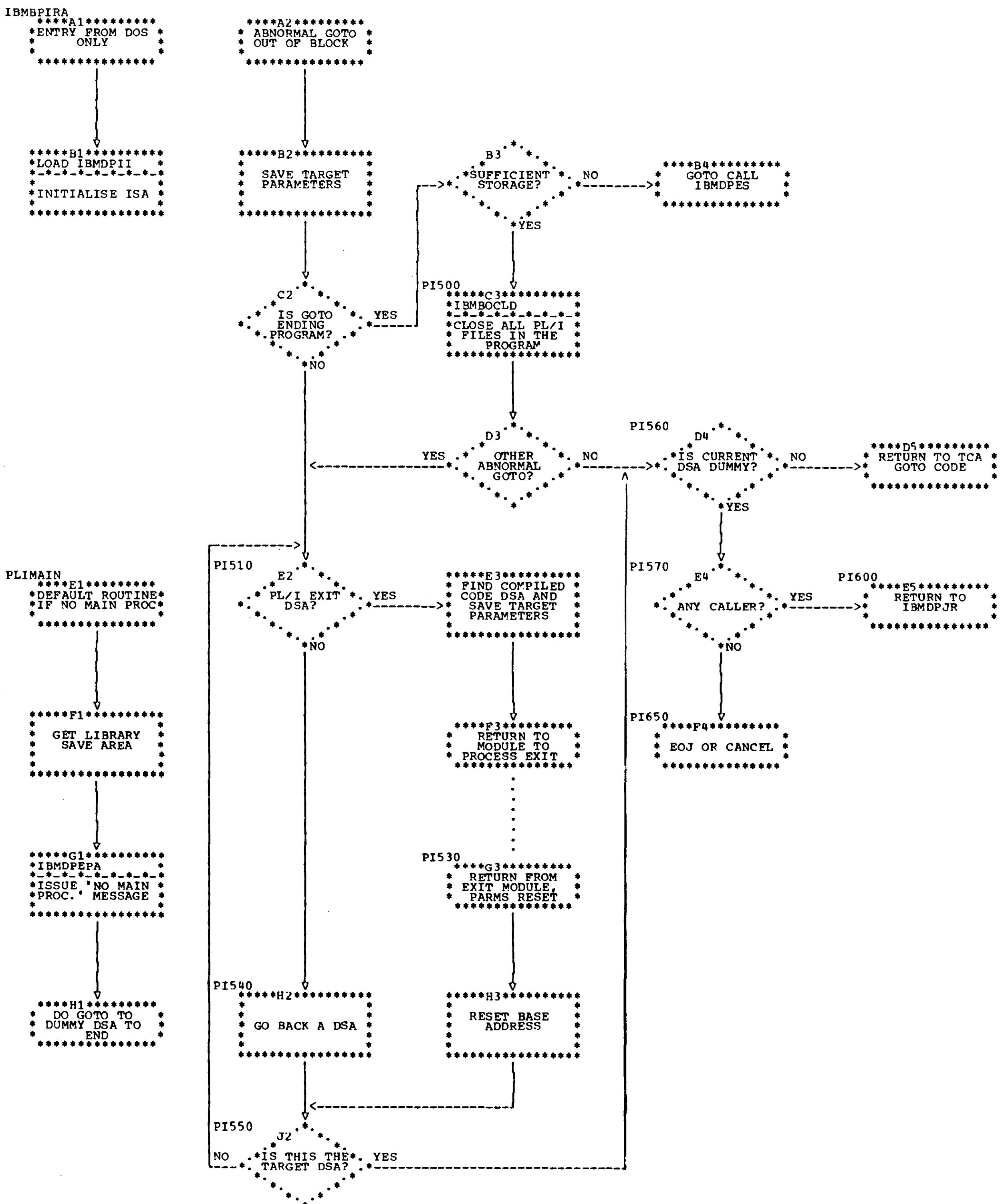


Chart DPIR. Program initialization and termination

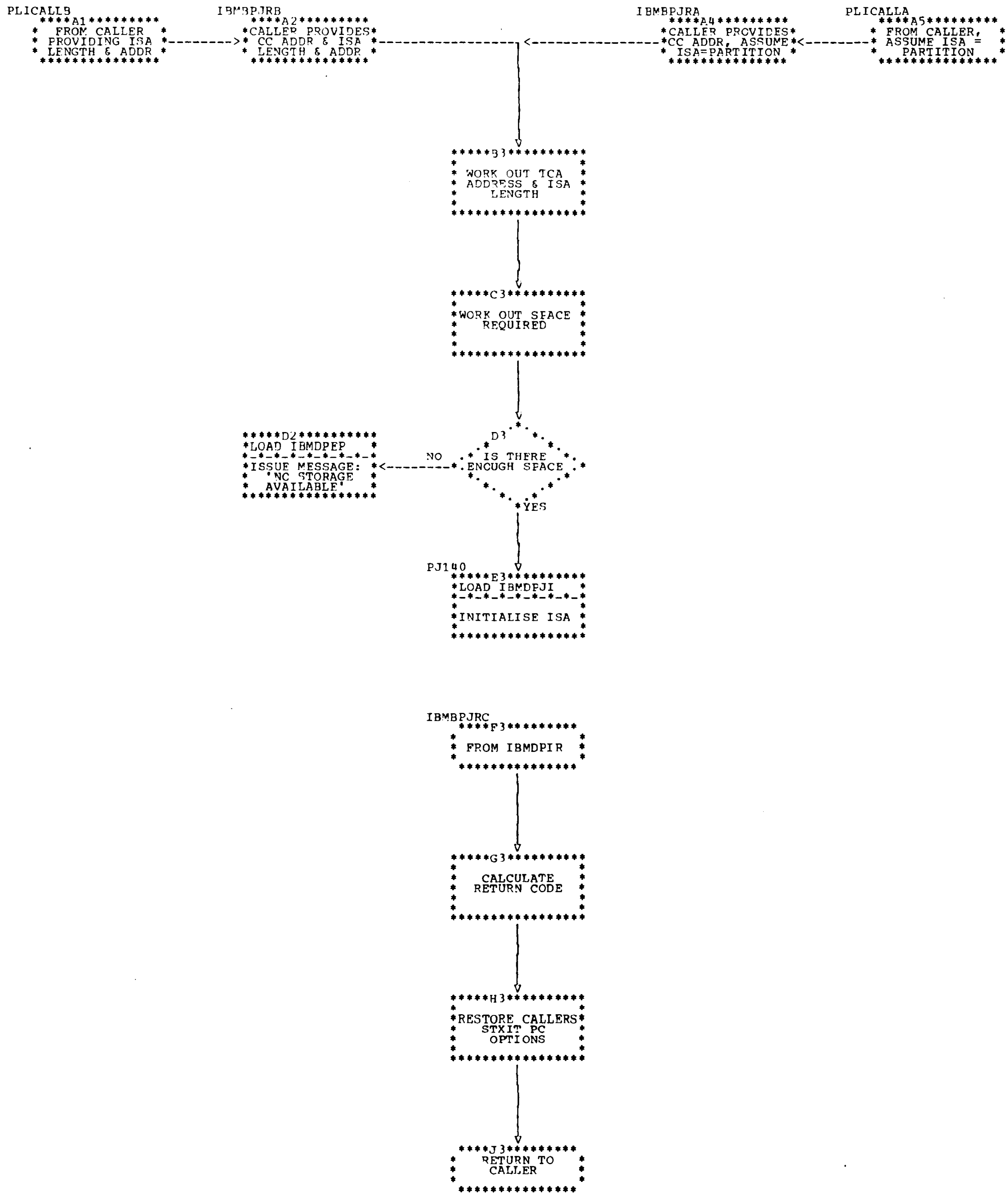


Chart DPJR. Program initialization from caller

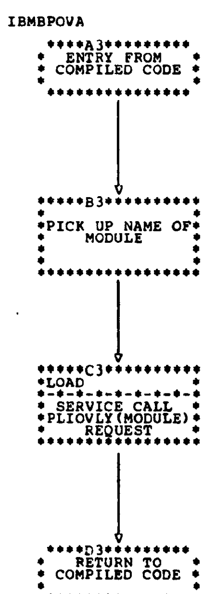


Chart DPOV. Overlay module

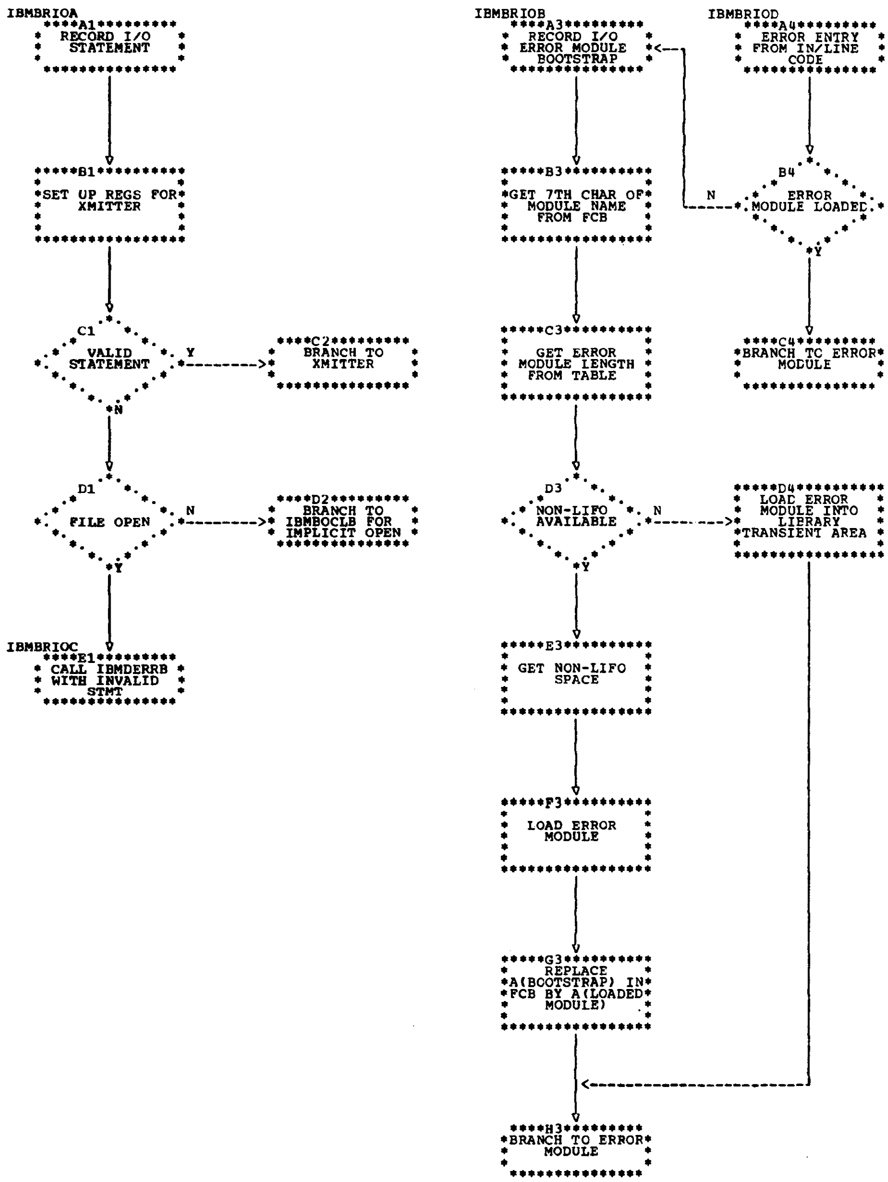


Chart DRIO. Record I/O interface

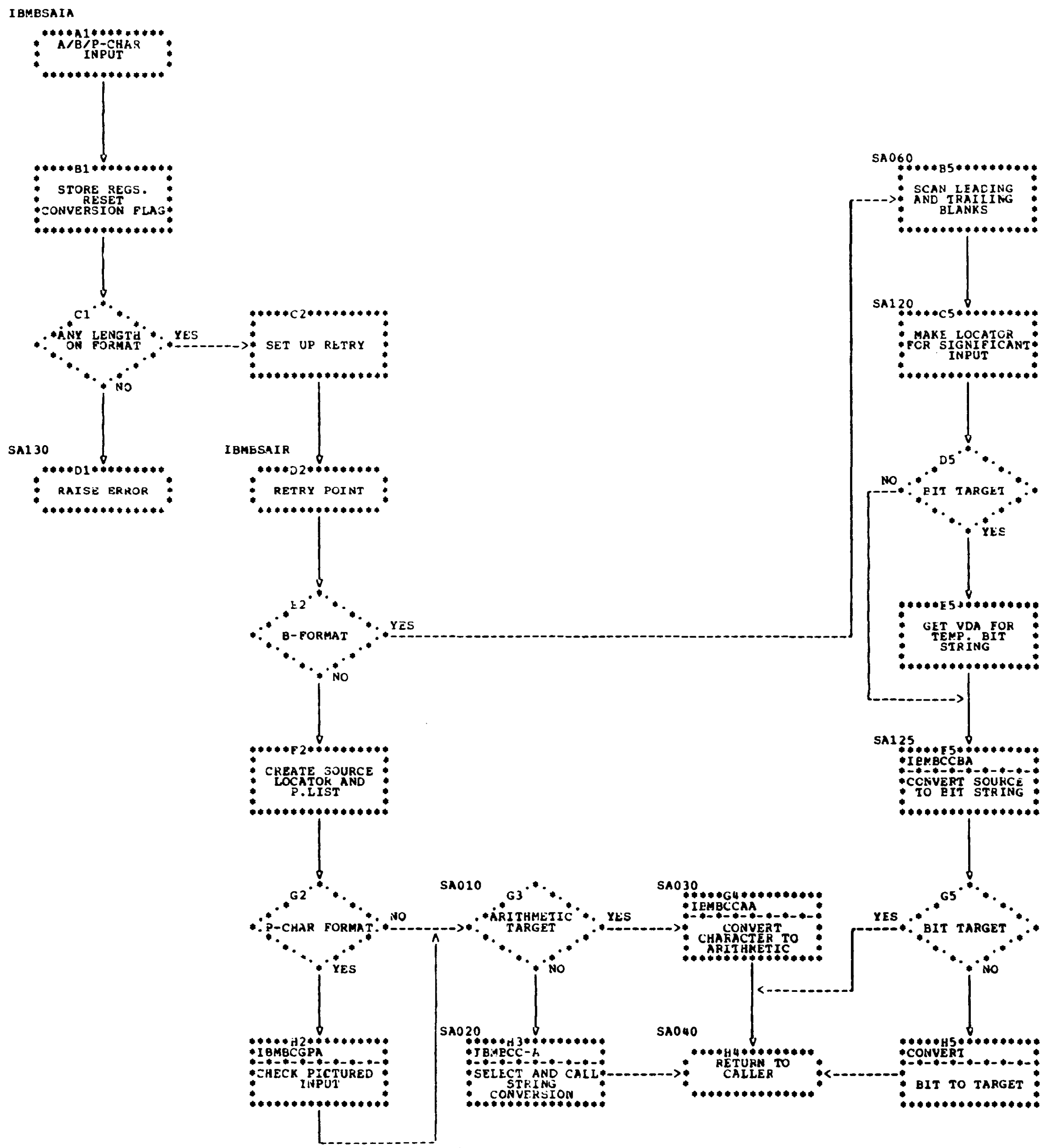


Chart BSAI. Input conversion director (A, P, and B formats)

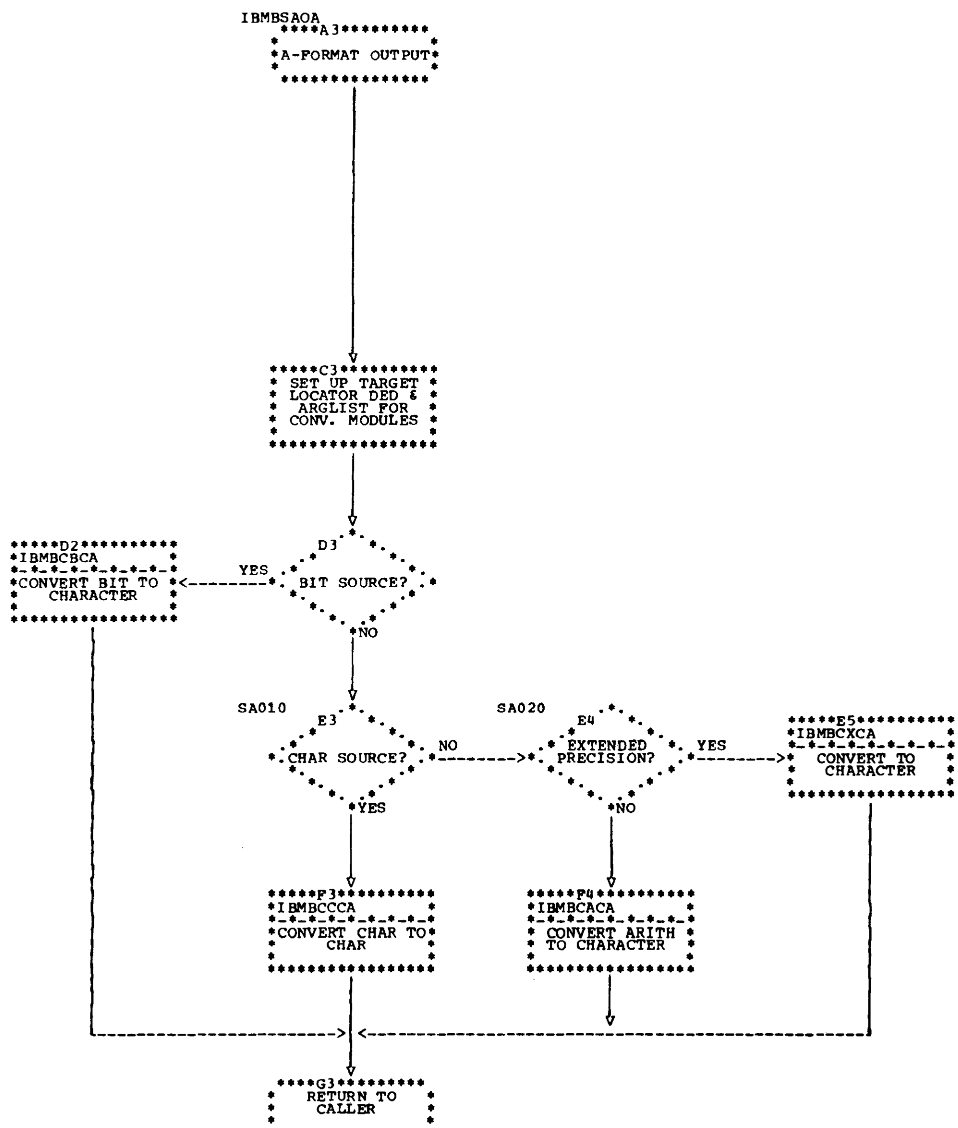


Chart BSAO. Output conversion director (A, P, and B formats)

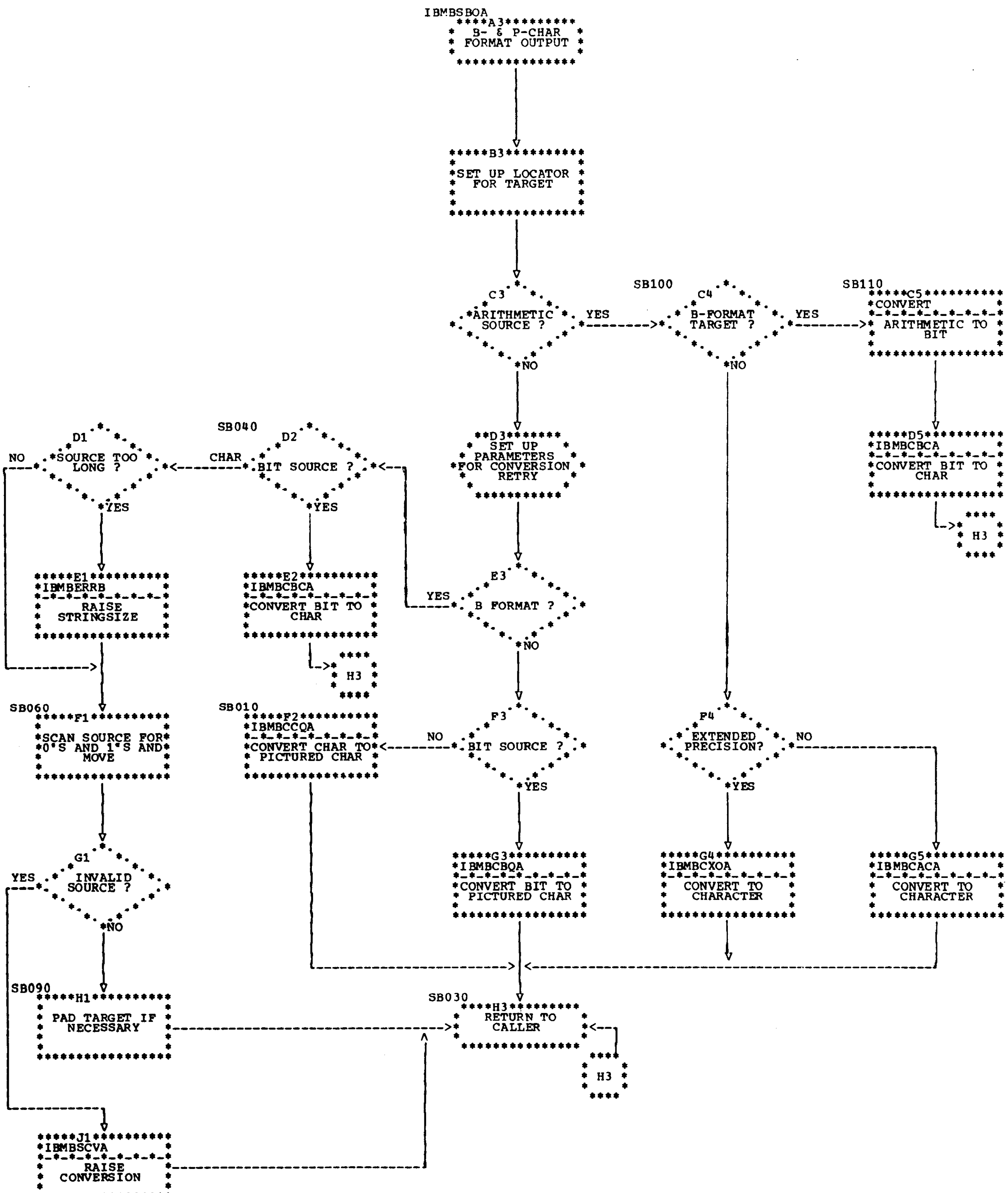


Chart BSBO. Output conversion director (B and P formats)

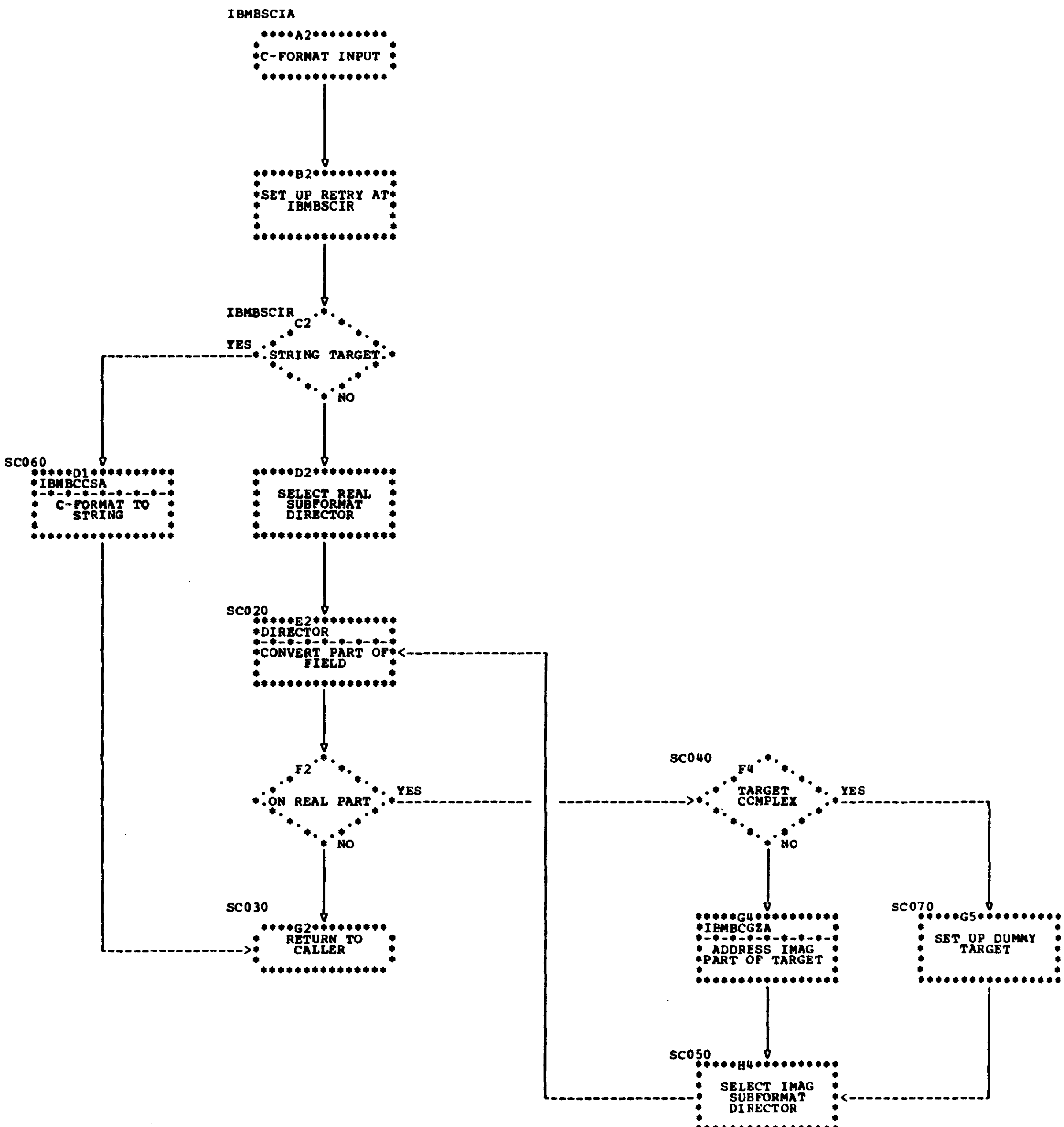


Chart BSCI. Input conversion director (C format)

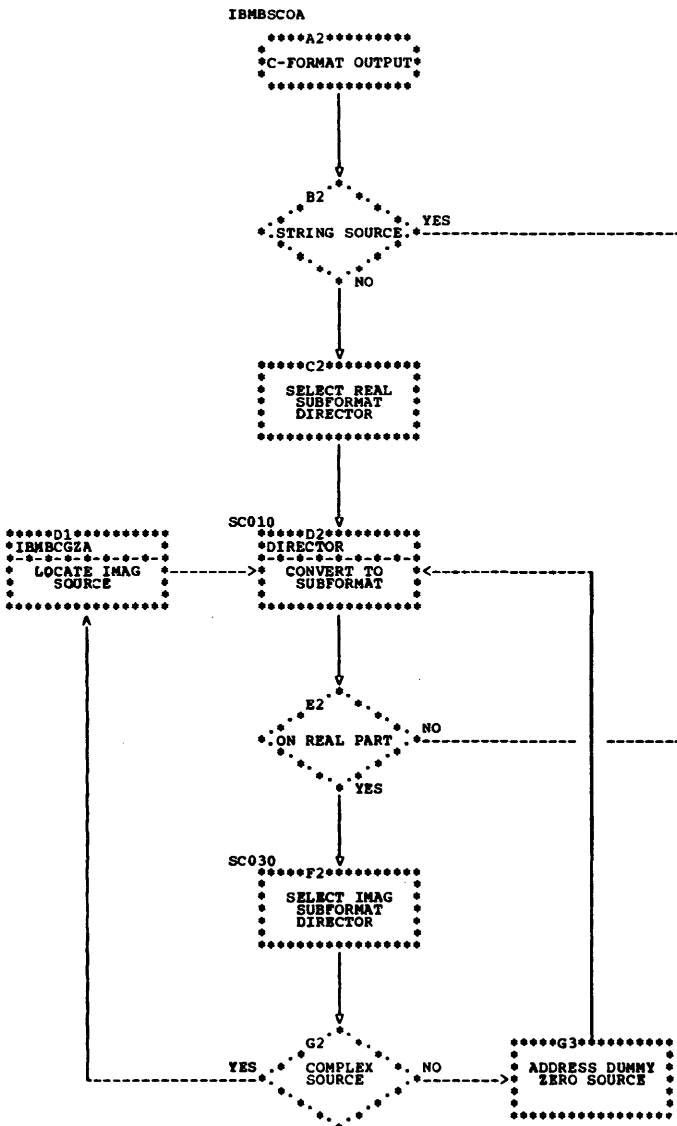


Chart BSCO. Output conversion director (C format)

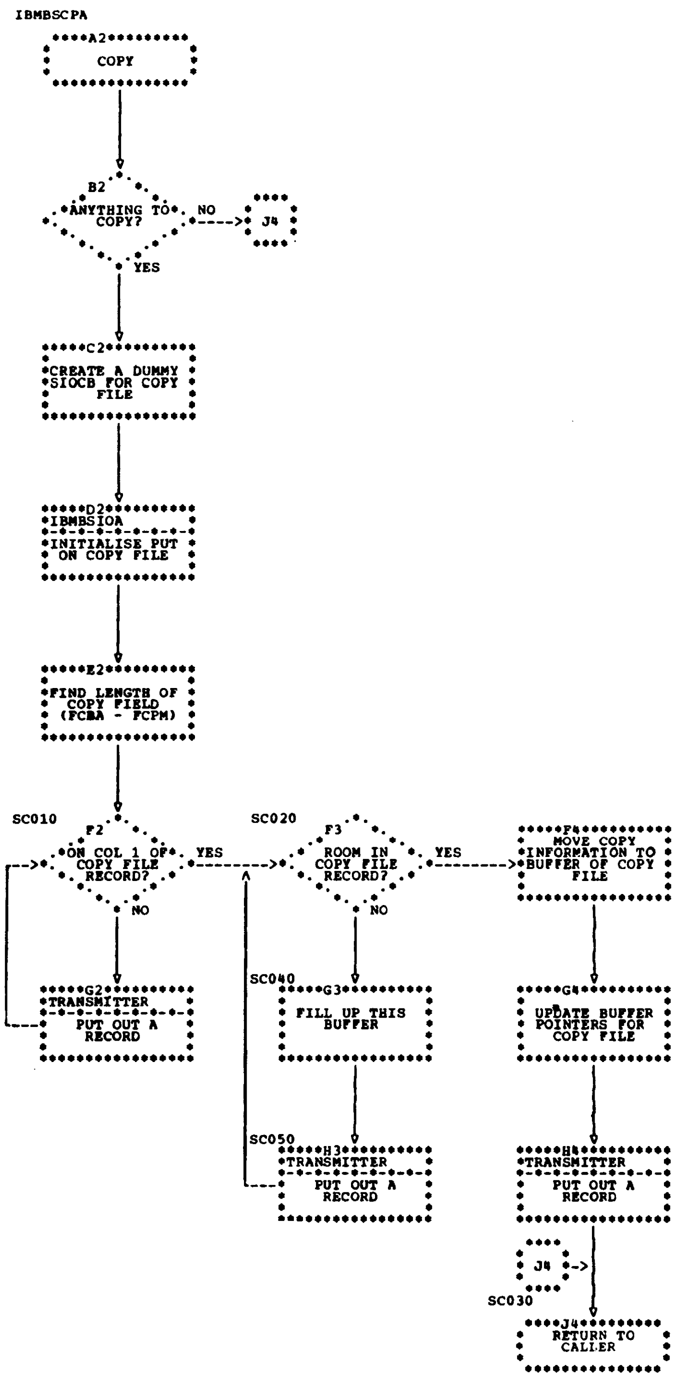


Chart DSCP. COPY option module

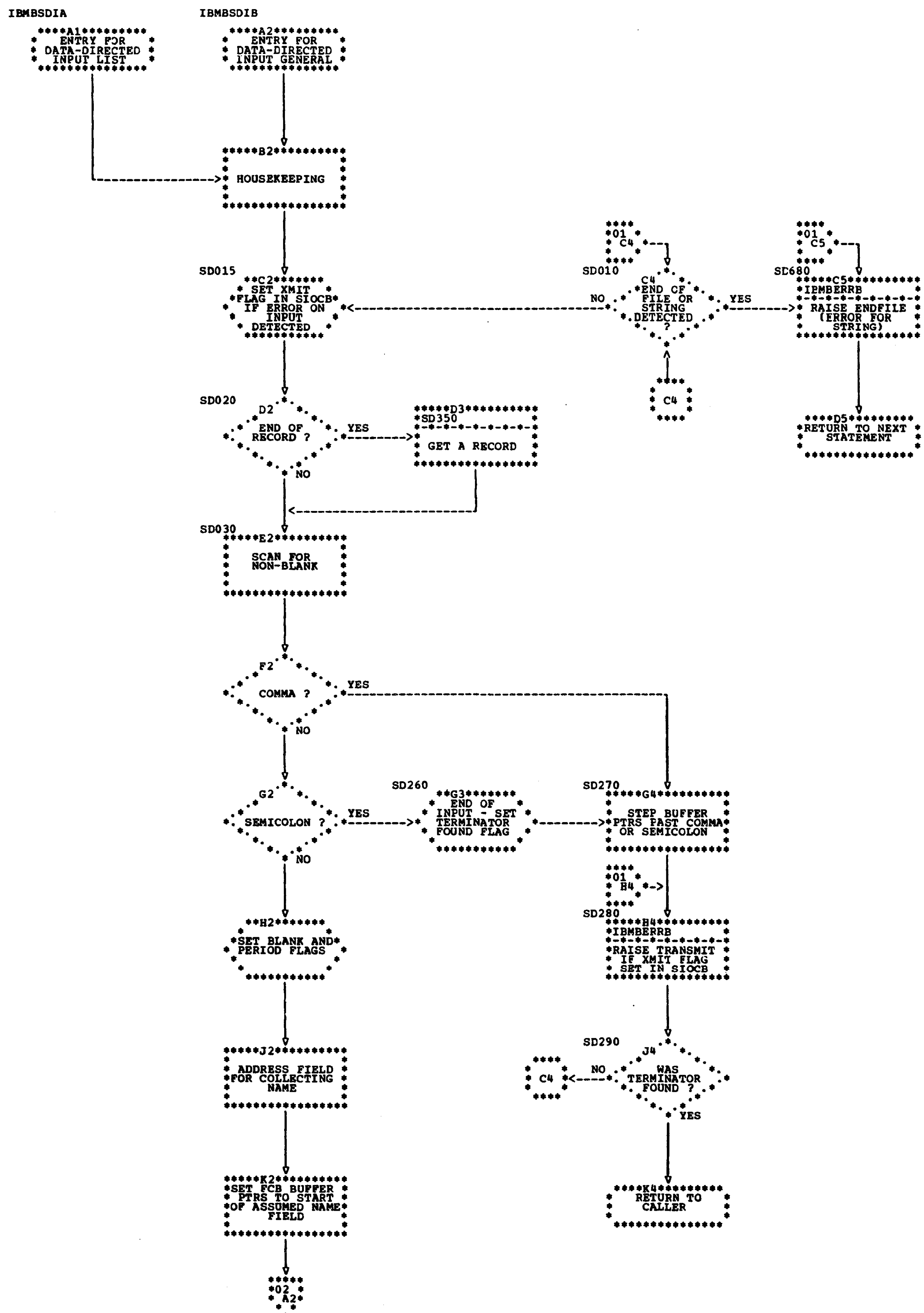


Chart DSDI. Data-directed Input (part 1 of 9)

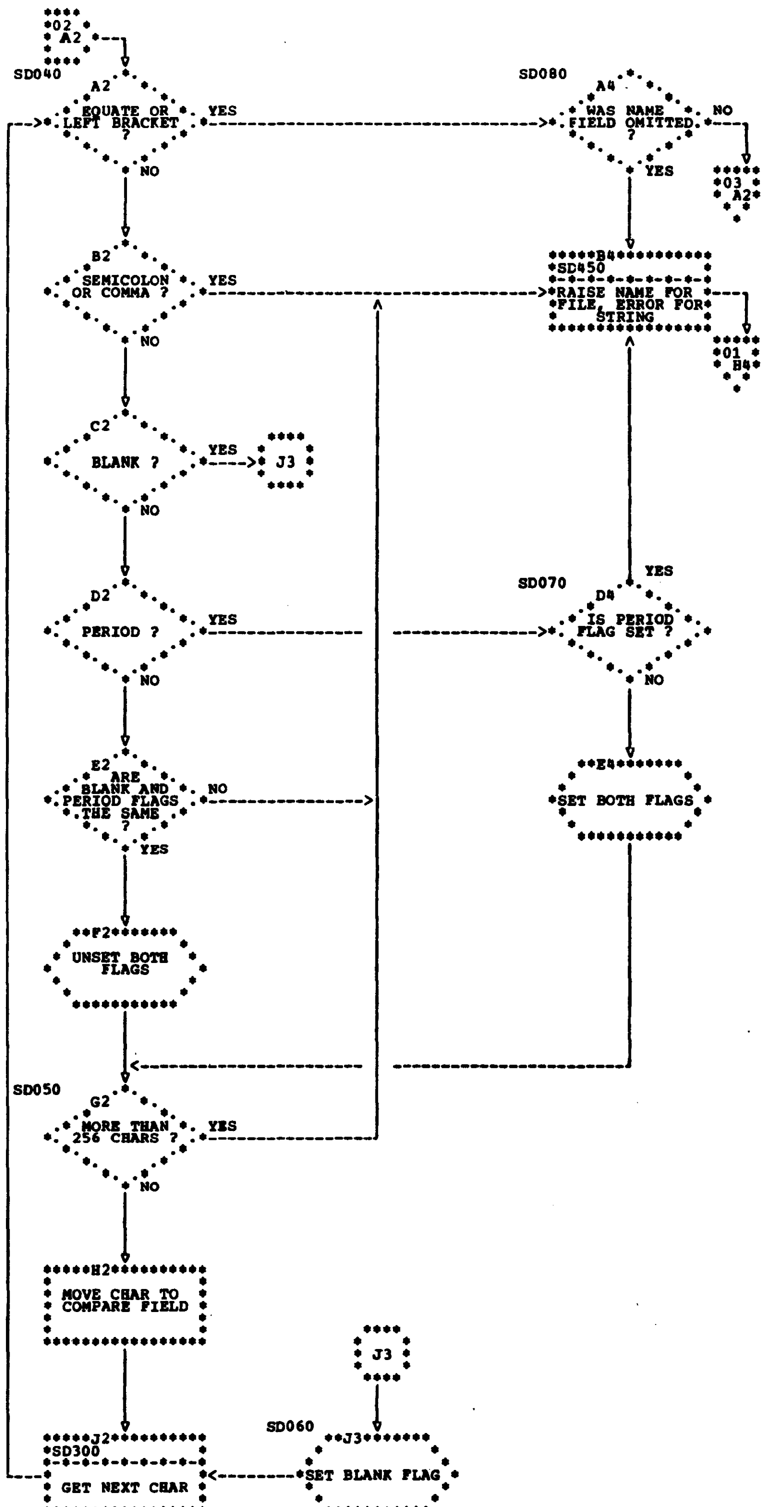


Chart DSDI. Data-directed Input (part 2 of 9)

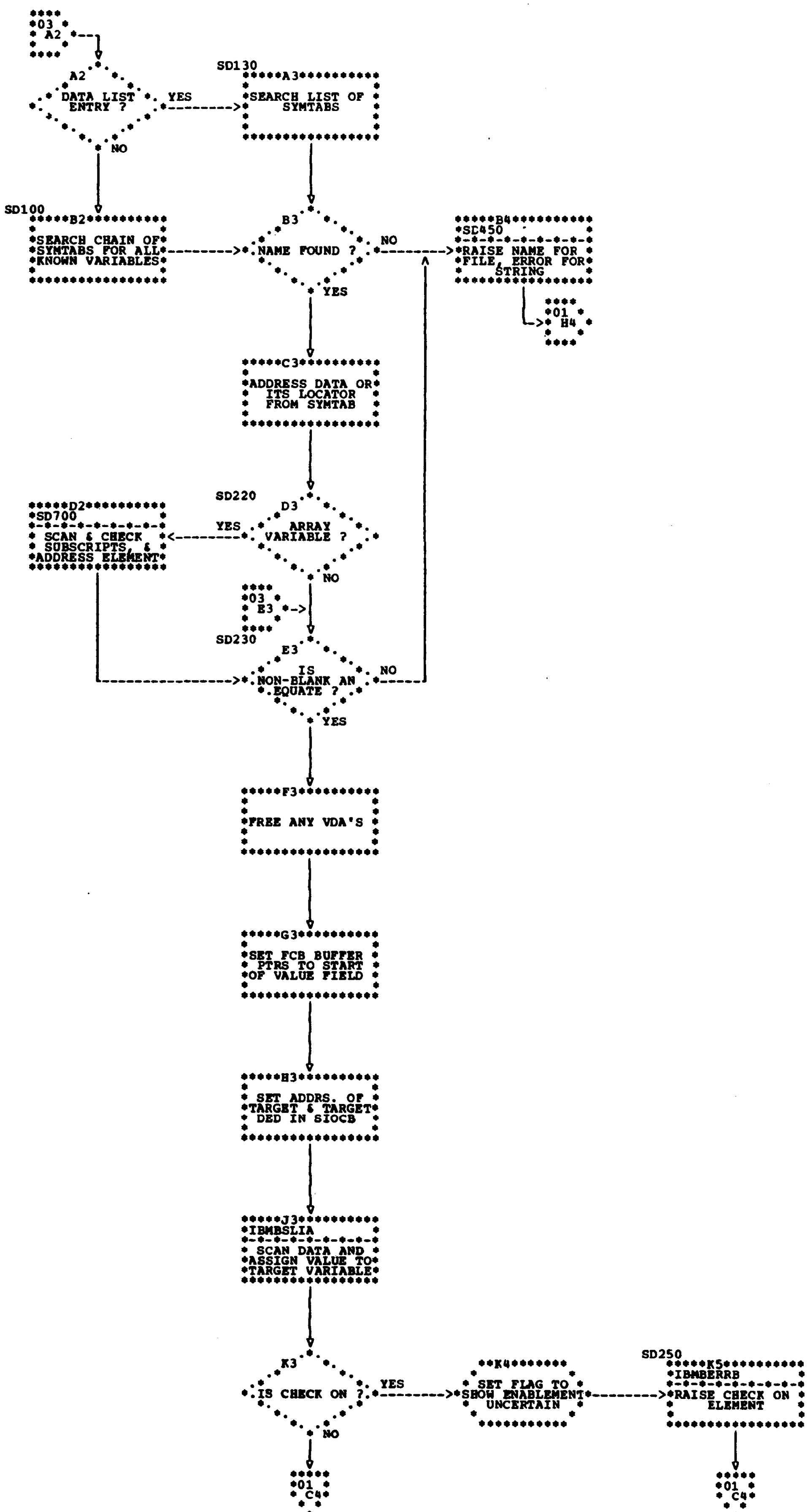


Chart DSDI. Data-directed Input (part 3 of 9)

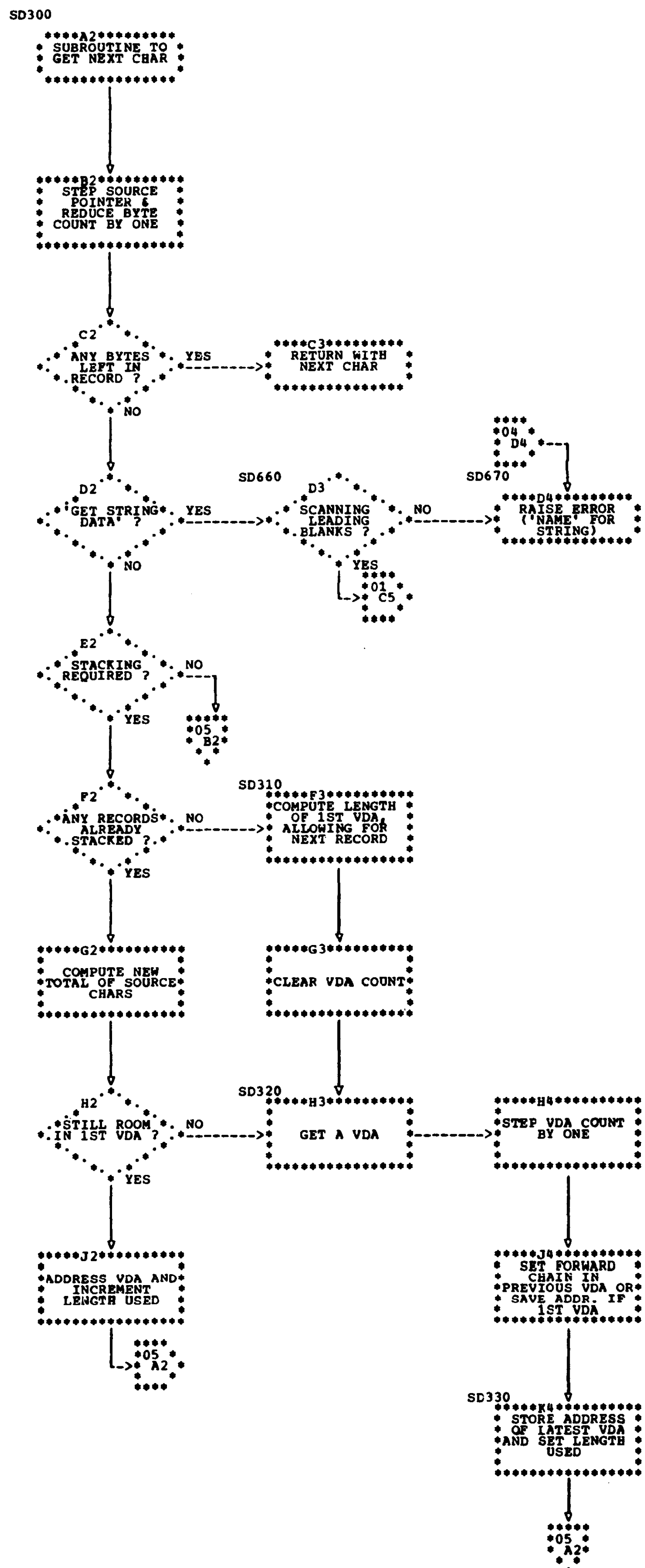


Chart DSDI. Data-directed Input (part 4 of 9)

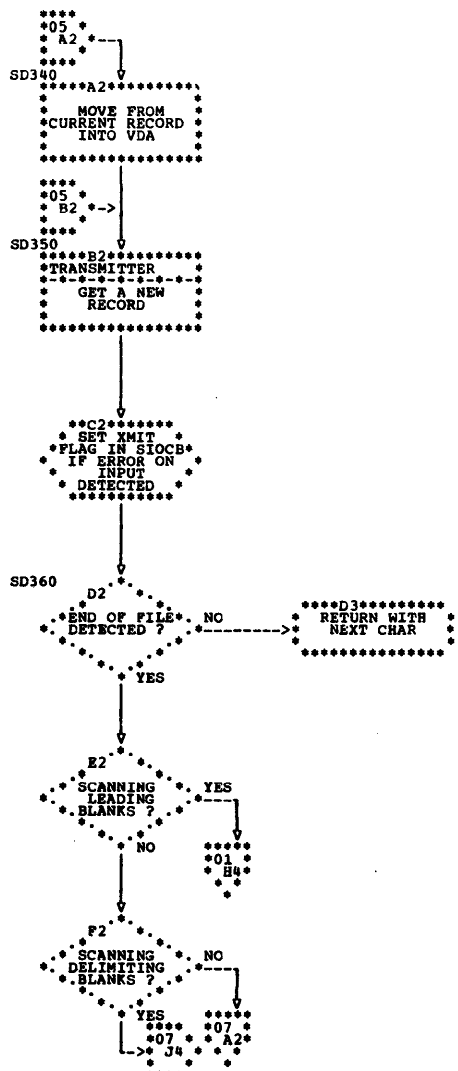


Chart DSDI. Data-directed Input (part 5 of 9)

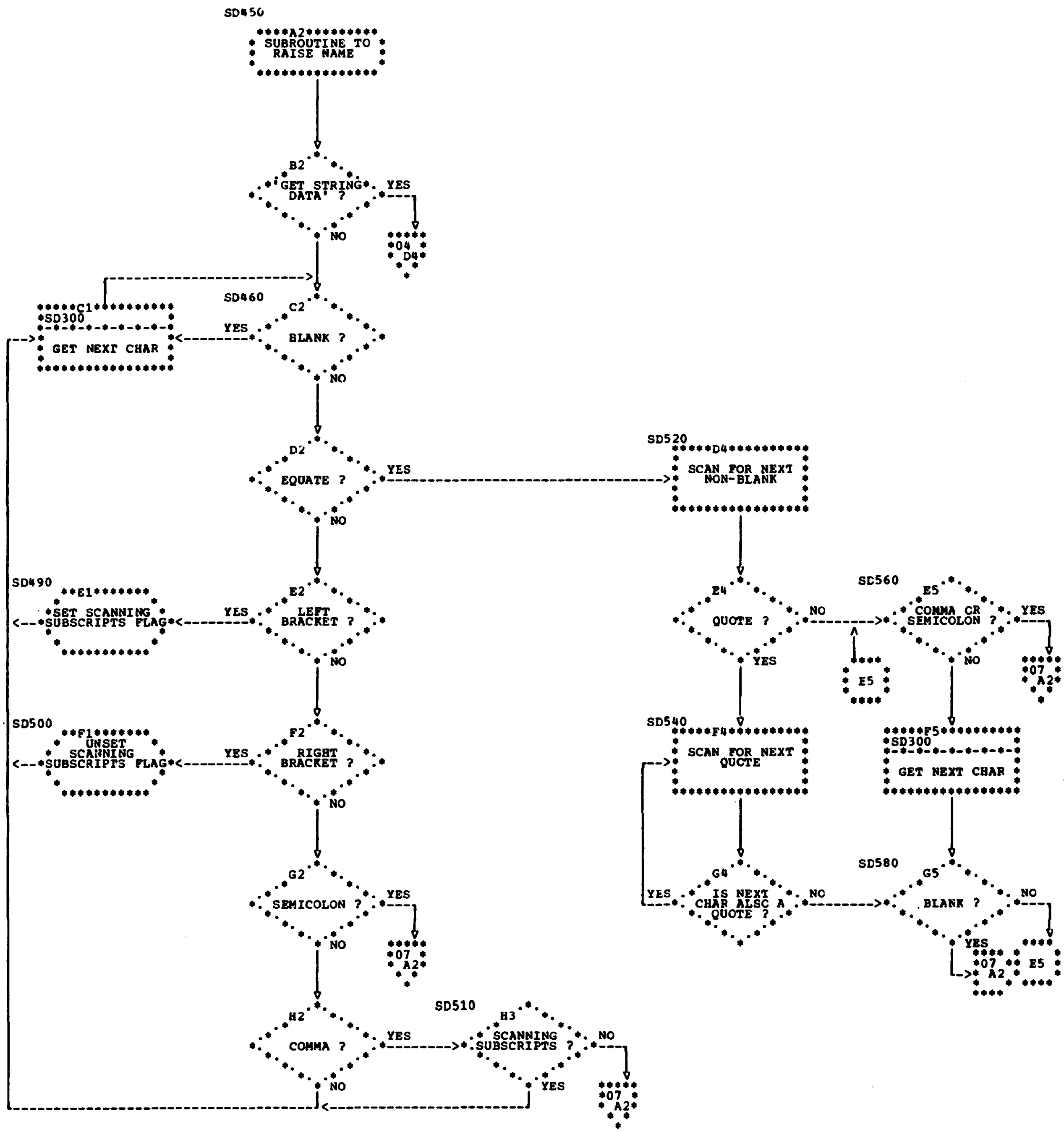


Chart DSDI. Data-directed Input (part 6 of 9)

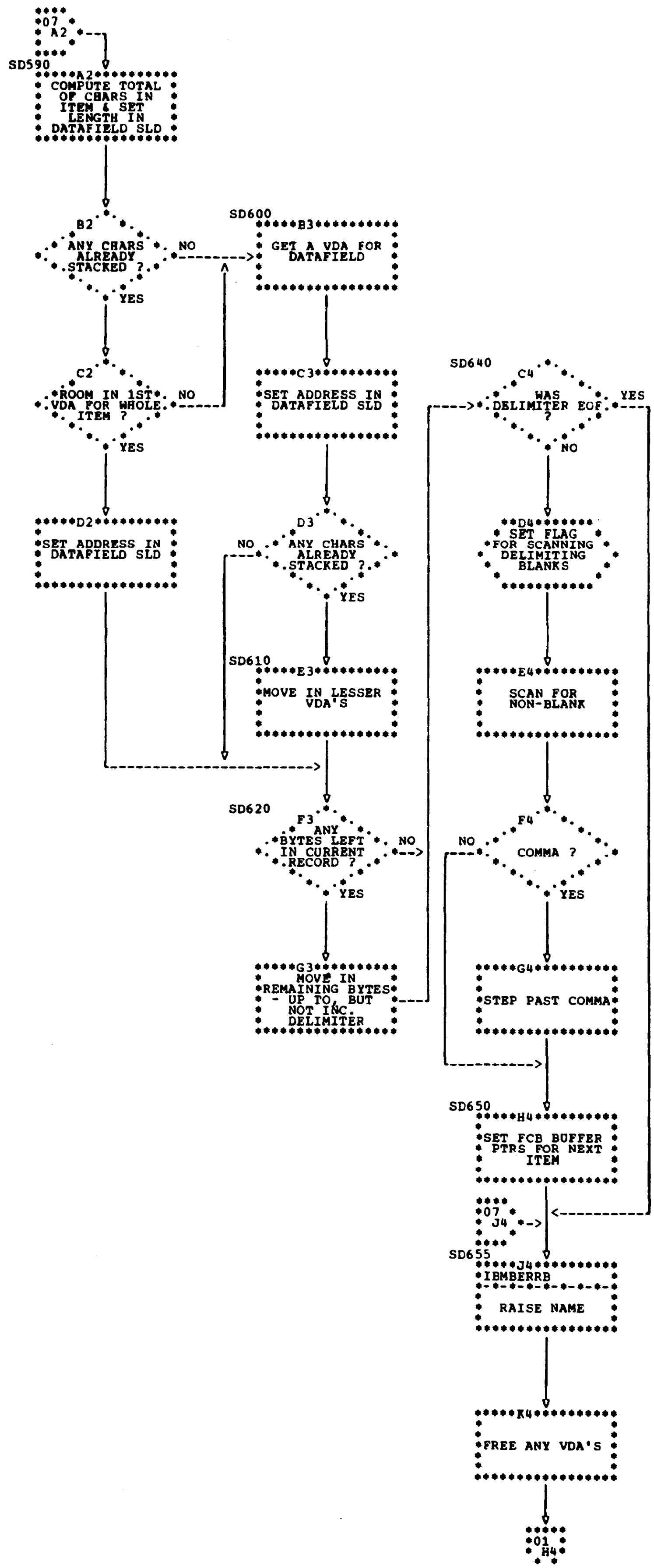


Chart DSDI. Data-directed Input (part 7 of 9)

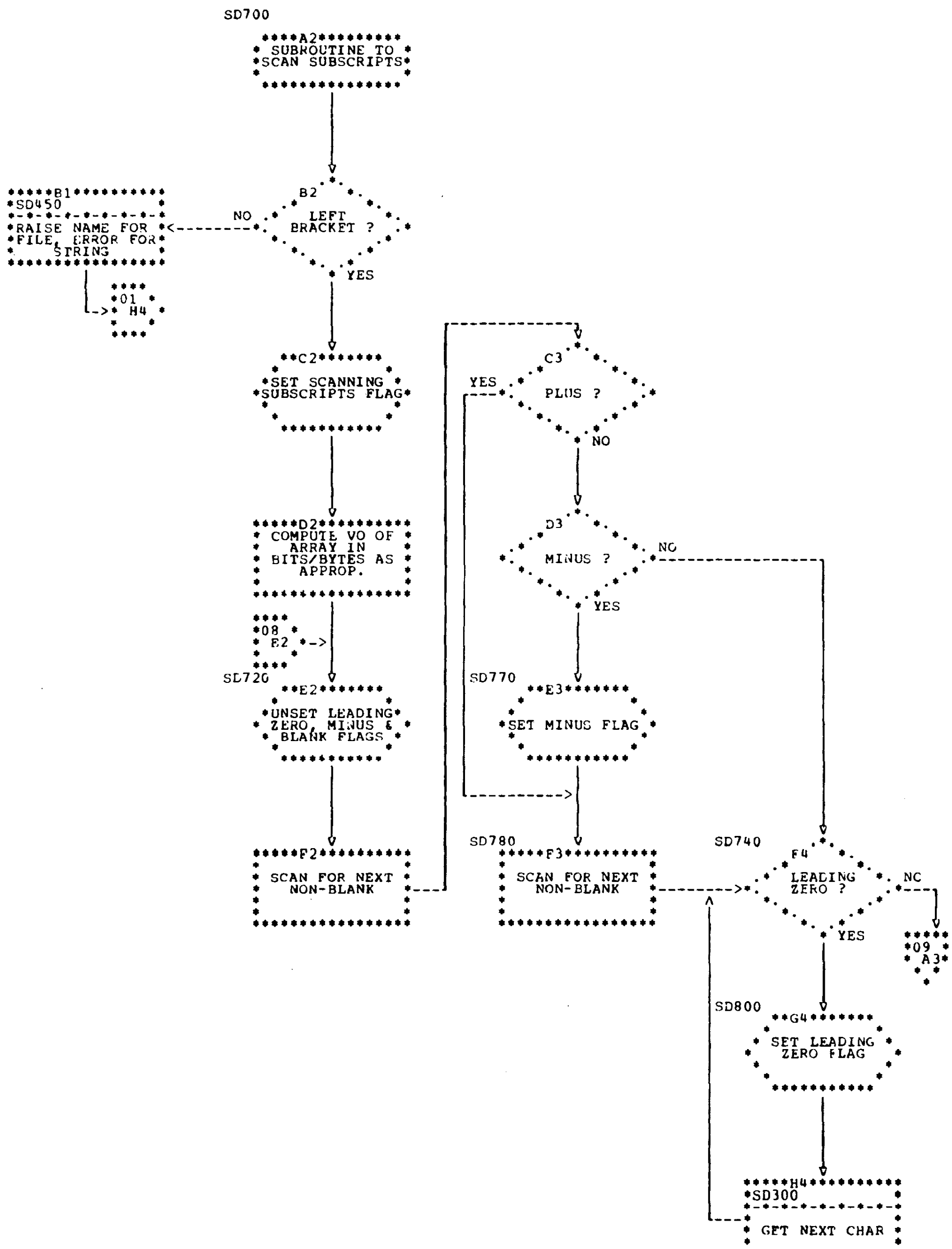


Chart DSDI. Data-directed Input (part 8 of 9)

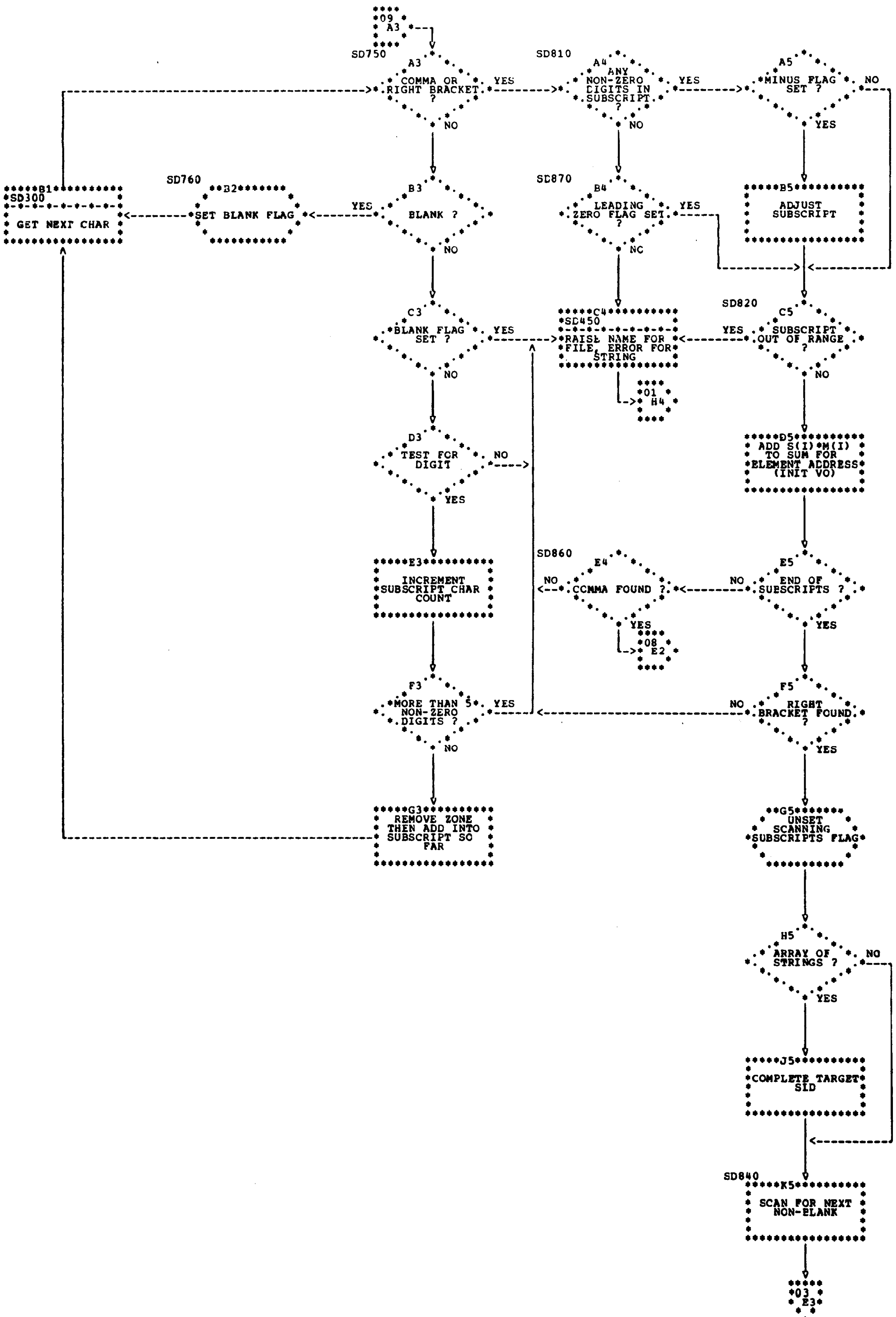


Chart DSDI. Data-directed Input (part 9 of 9)

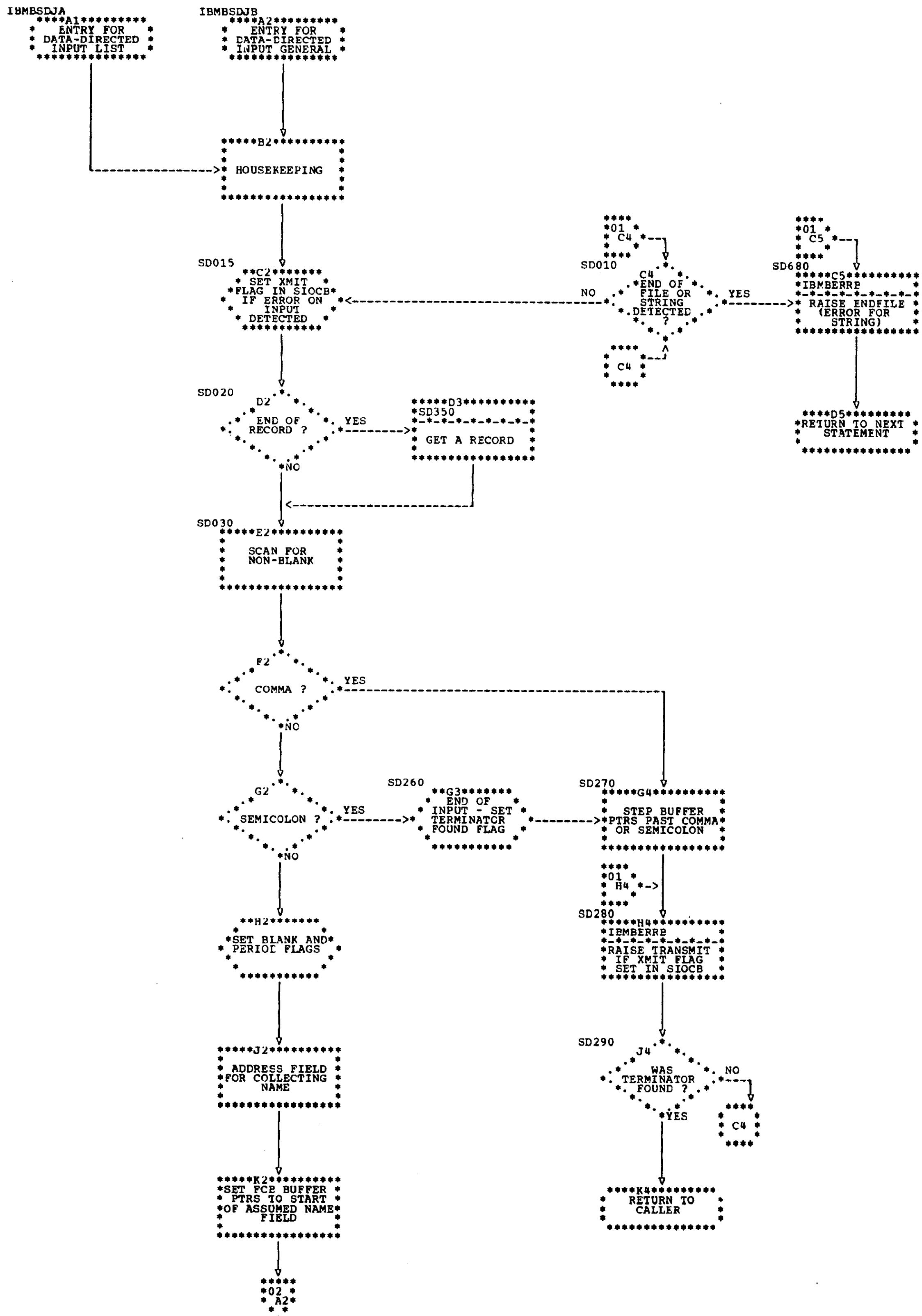


Chart DSDJ. Data-directed Input, restricted conversions (part 1 of 9)

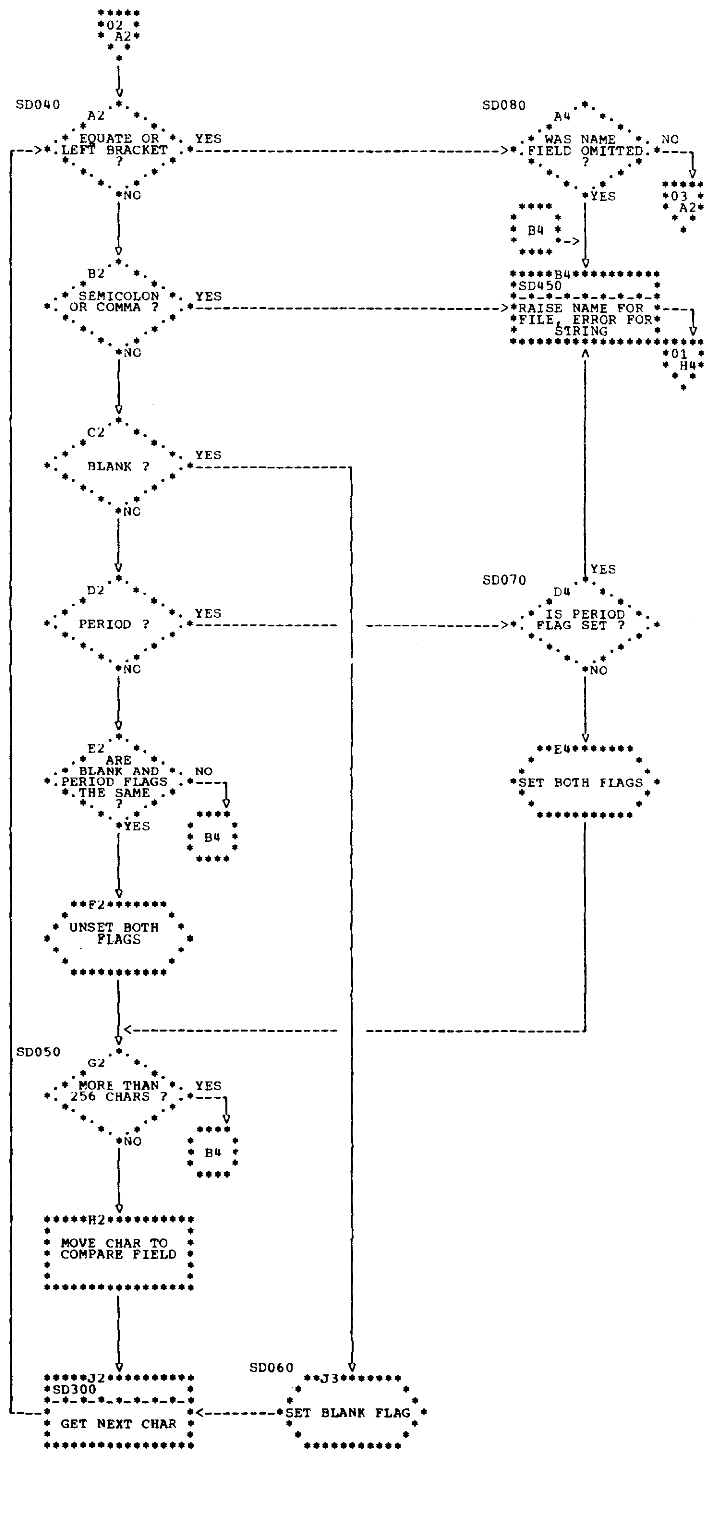


Chart DSDJ. Data-directed Input, restricted conversions (part 2 of 9)

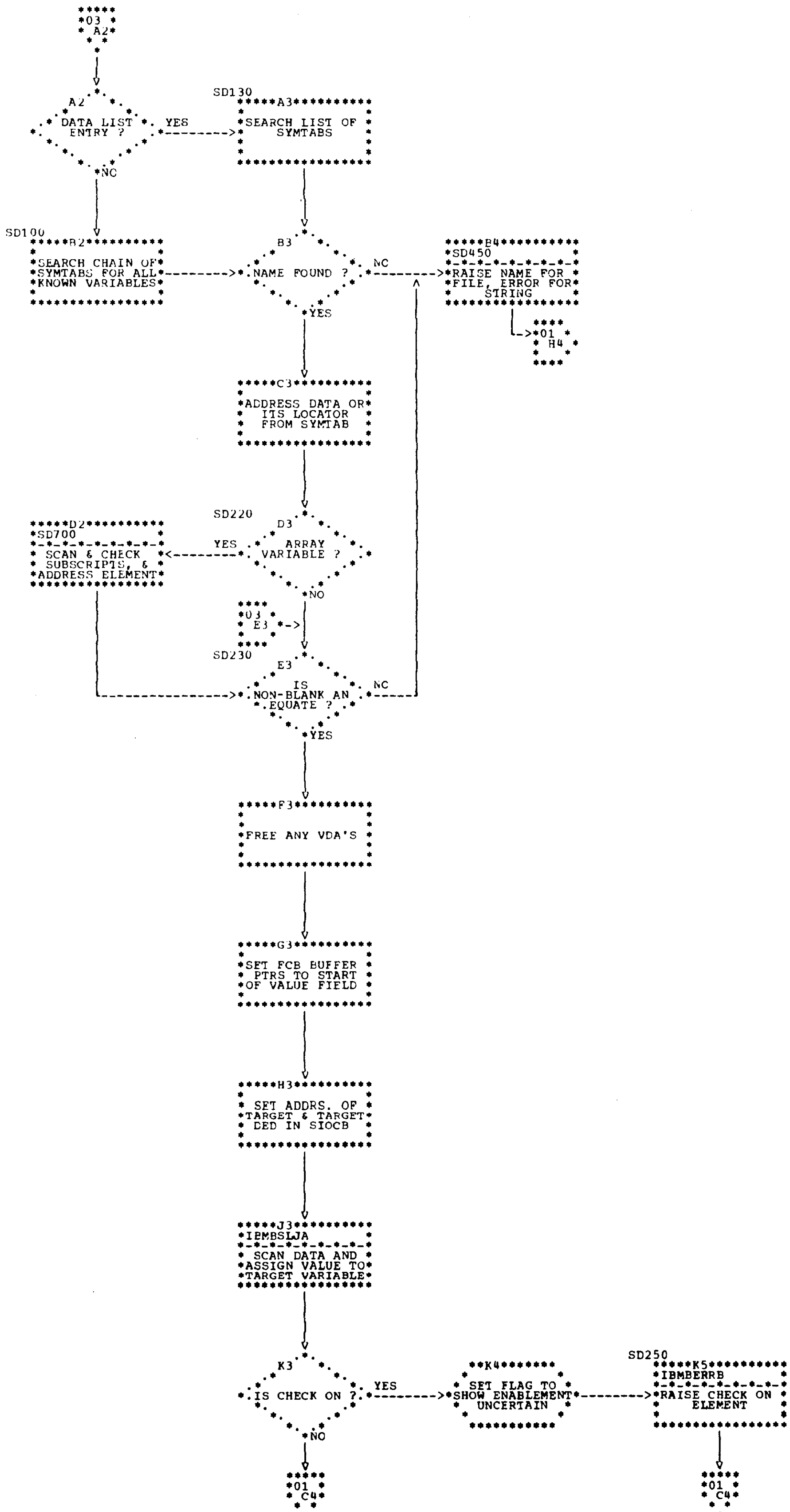


Chart DSDJ. Data-directed Input, restricted conversions (part 3 of 9)

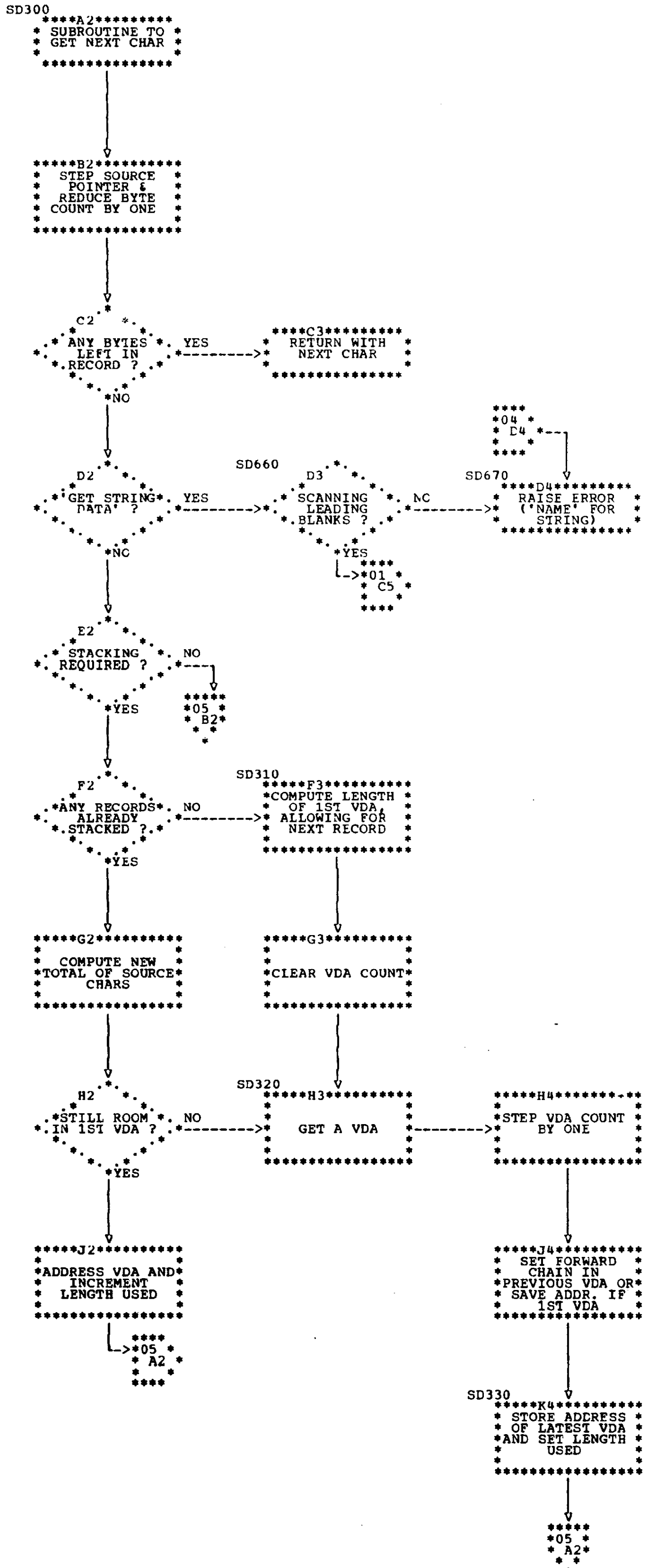


Chart DSDJ. Data-directed Input, restricted conversions (part 4 of 9)

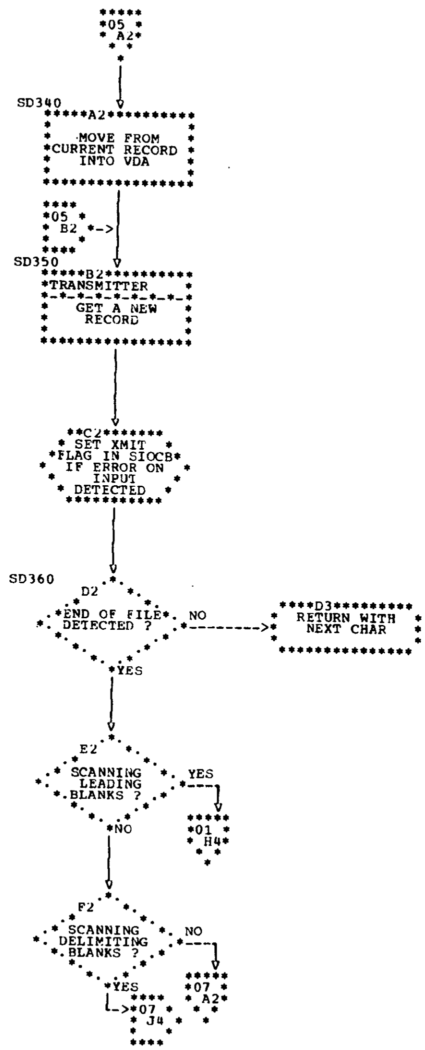


Chart DSDJ. Data-directed Input, restricted conversions (part 5 of 9)

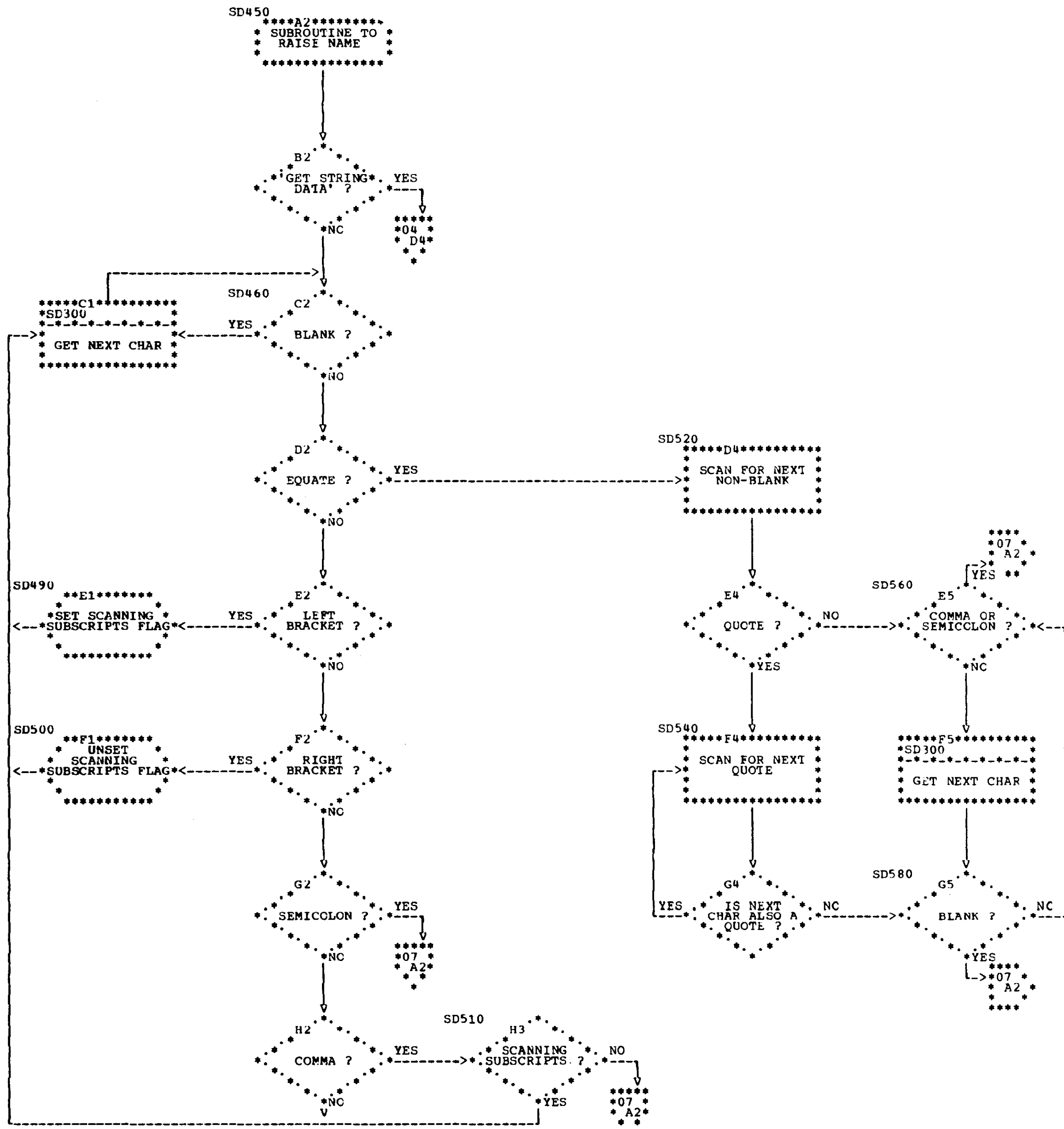


Chart DSDJ. Data-directed Input, restricted conversions (part 6 of 9)

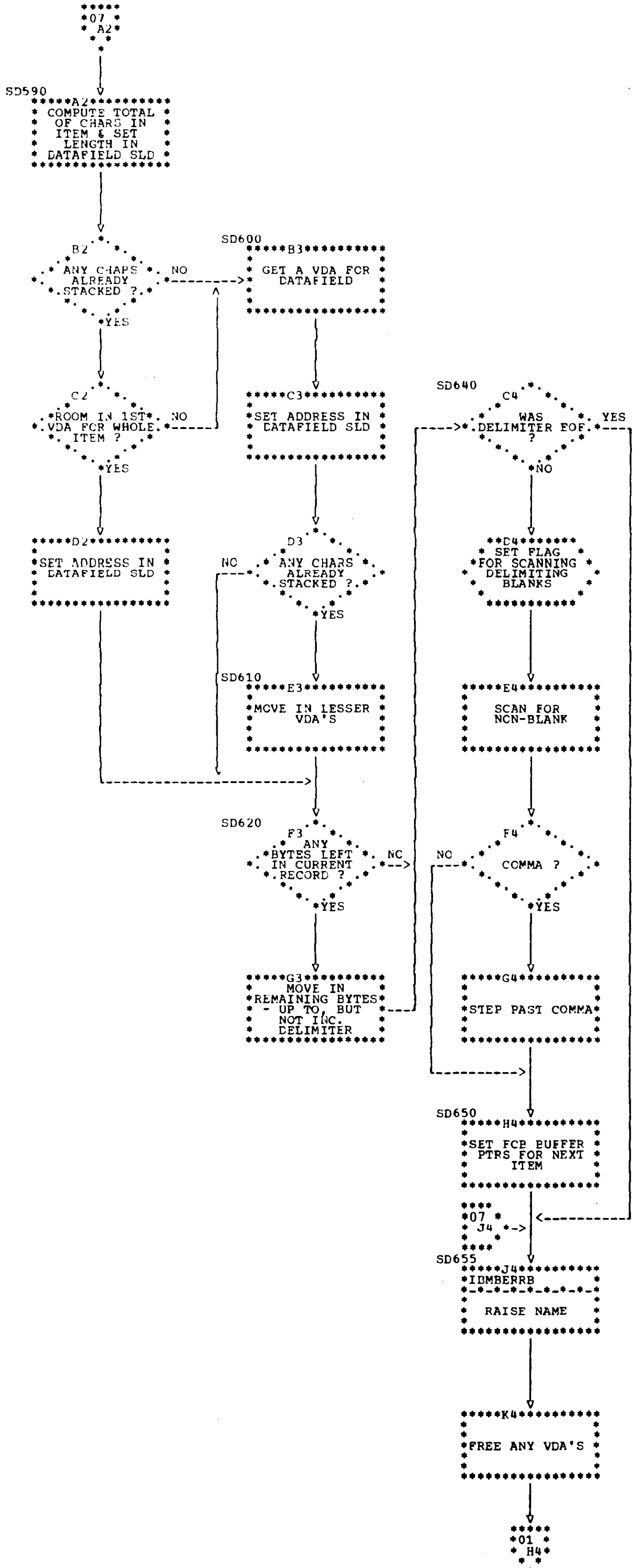


Chart DSDJ. Data-directed Input, restricted conversions (part 7 of 9)

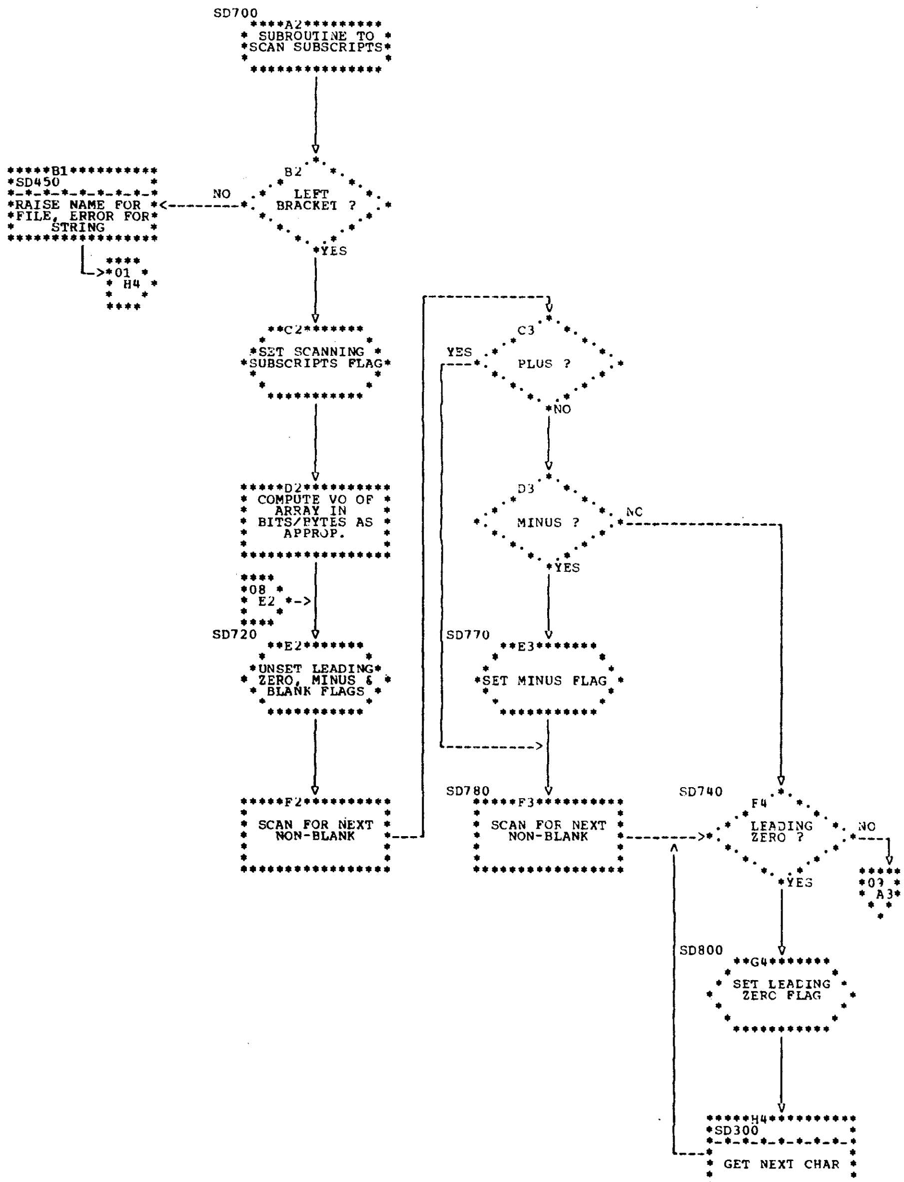


Chart DSDJ. Data-directed Input, restricted conversions (part 8 of 9)

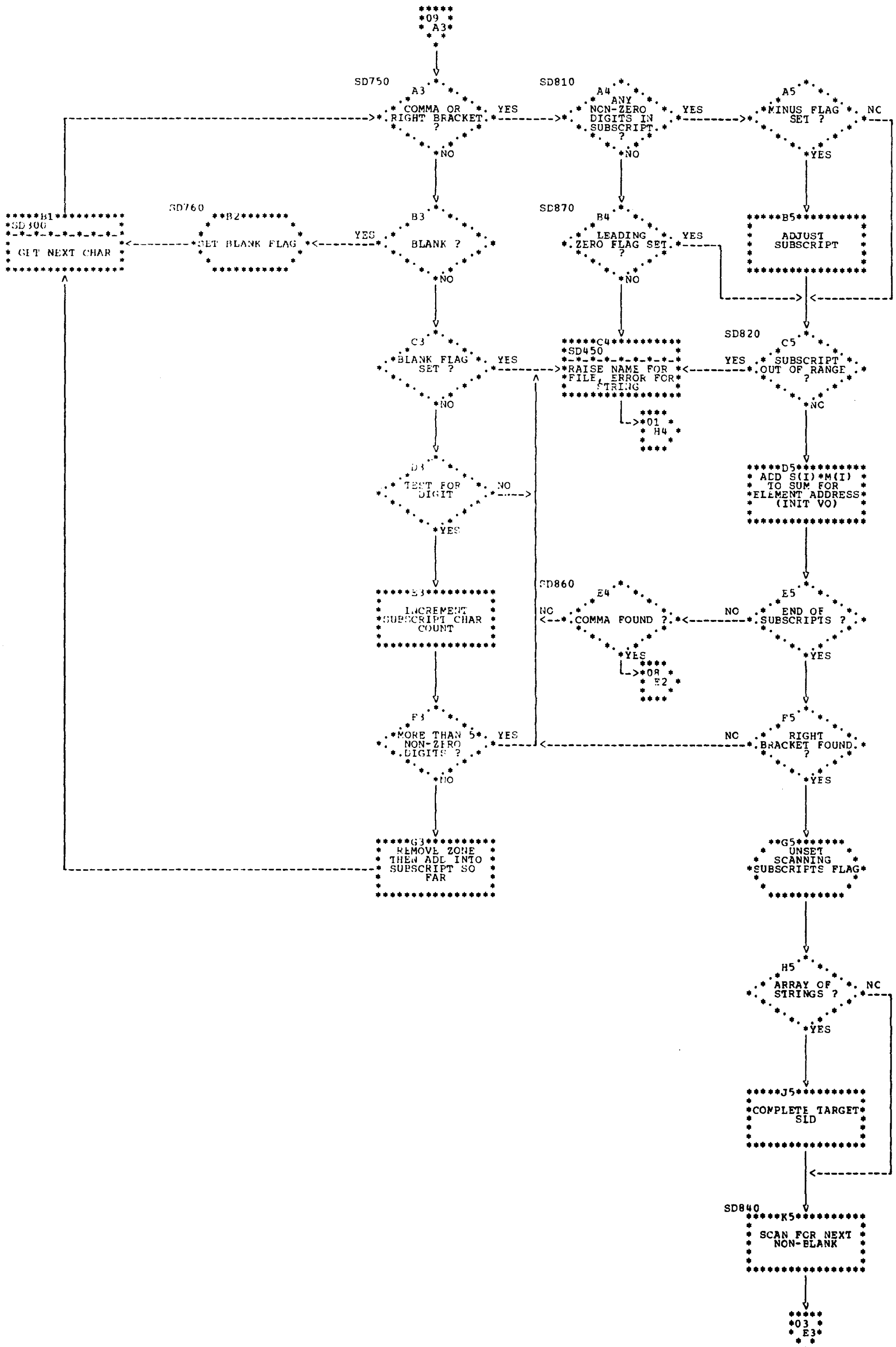


Chart DSDJ. Data-directed Input, restricted conversions (part 9 of 9)

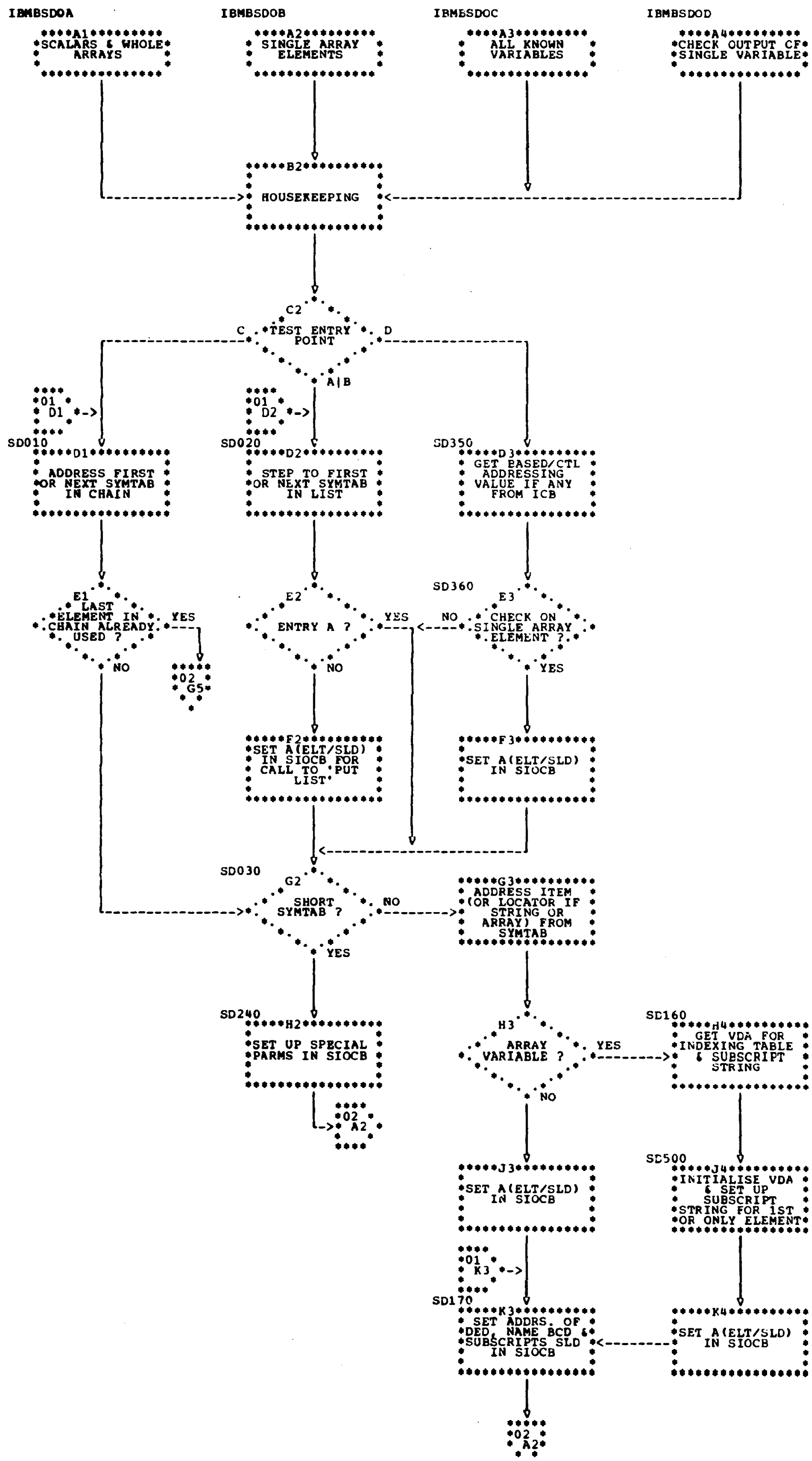


Chart DSDO. Data-directed Output (part 1 of 2)

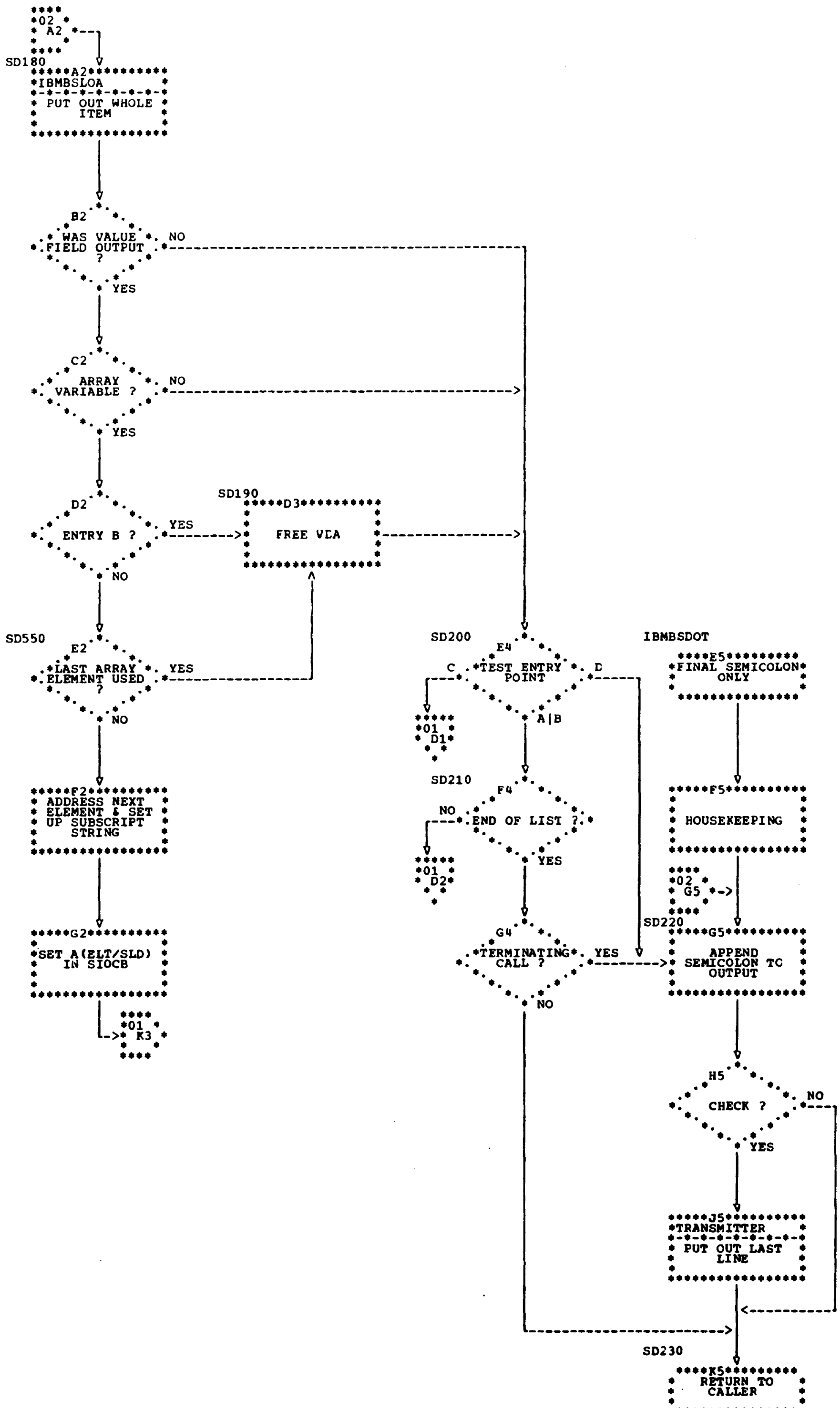


Chart DSDO. Data-directed Output (part 2 of 2)

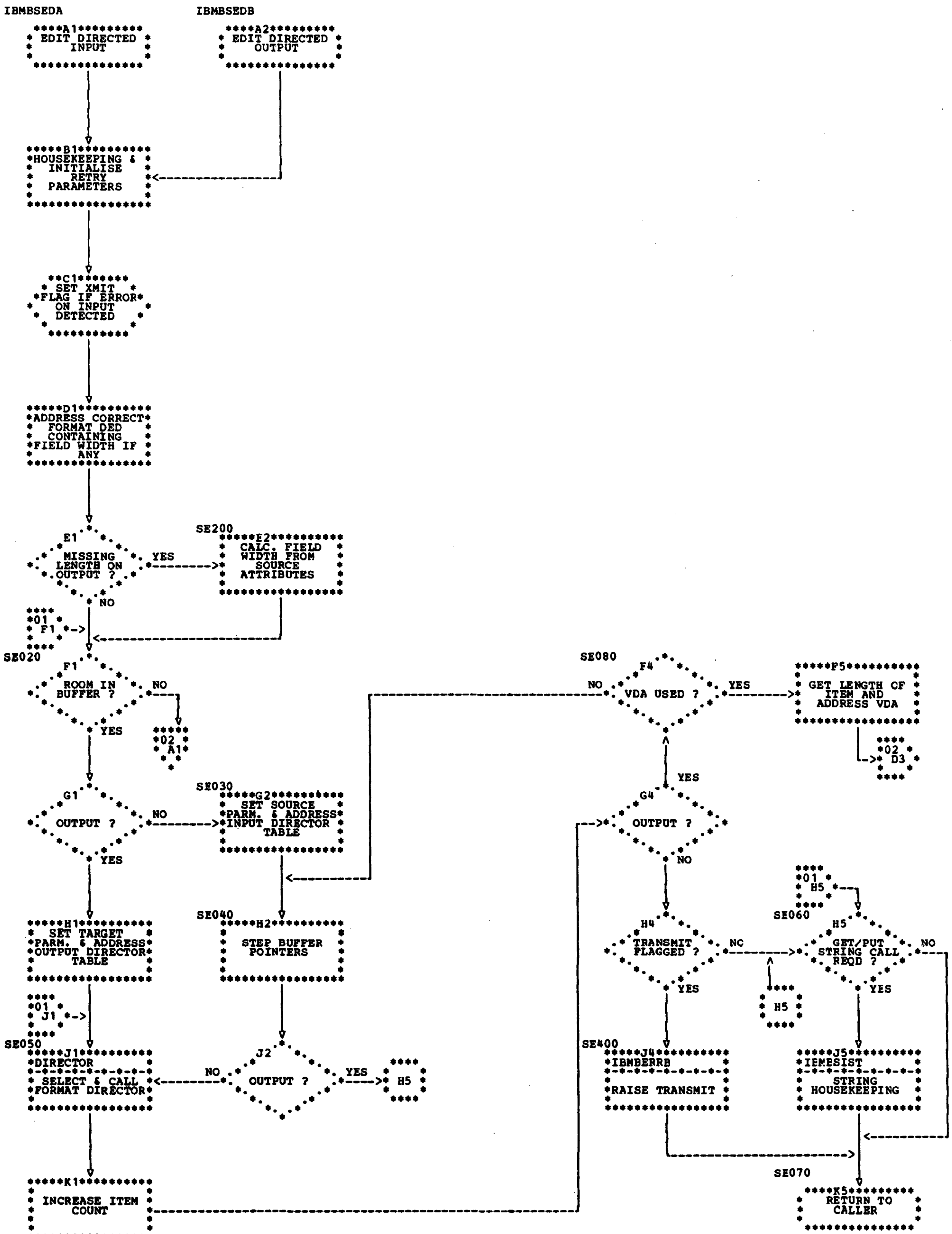


Chart DSED. Edit-directed I/O housekeeping (part 1 of 2)

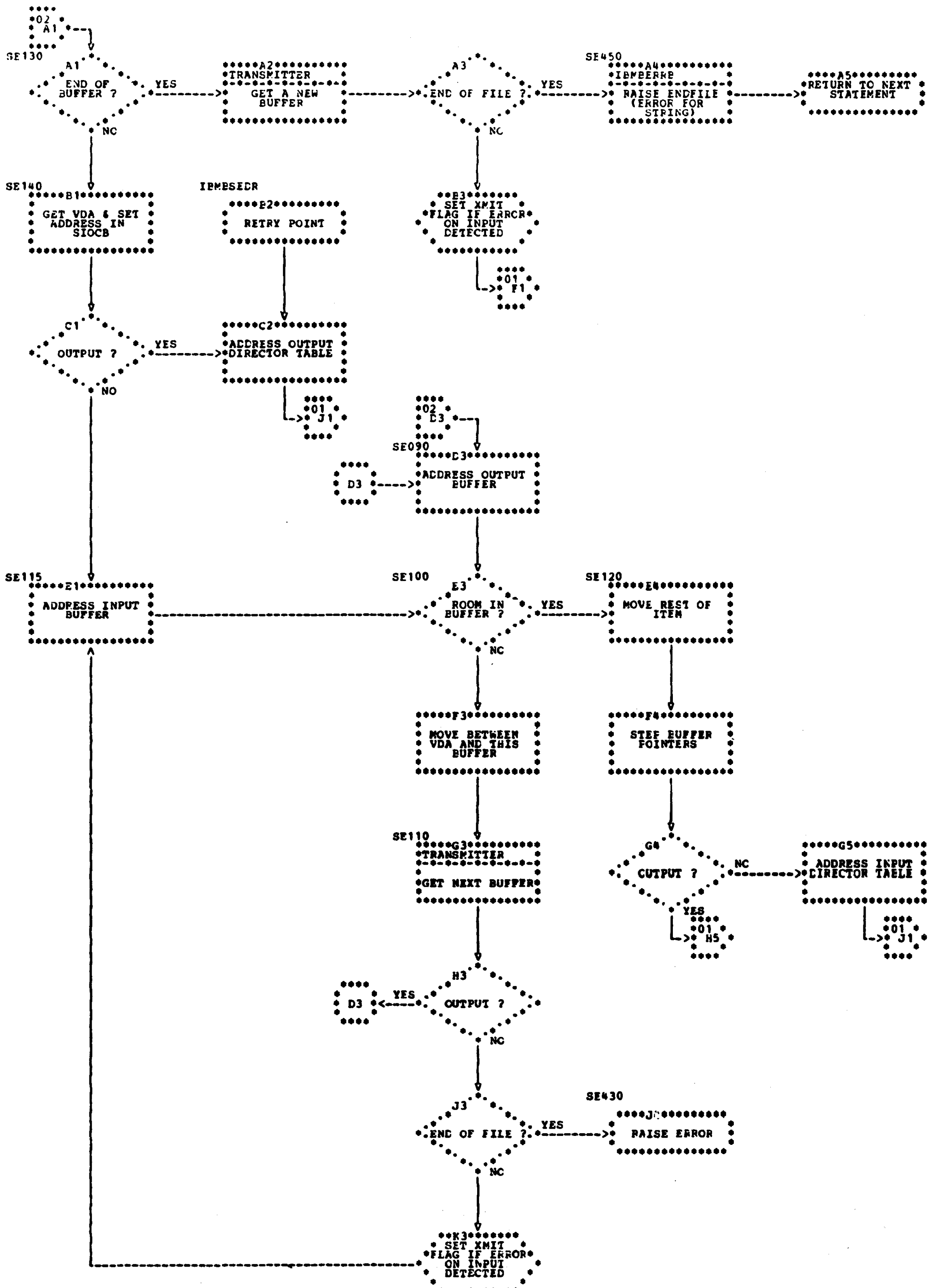


Chart DSED. Edit-directed I/O housekeeping (part 2 of 2)

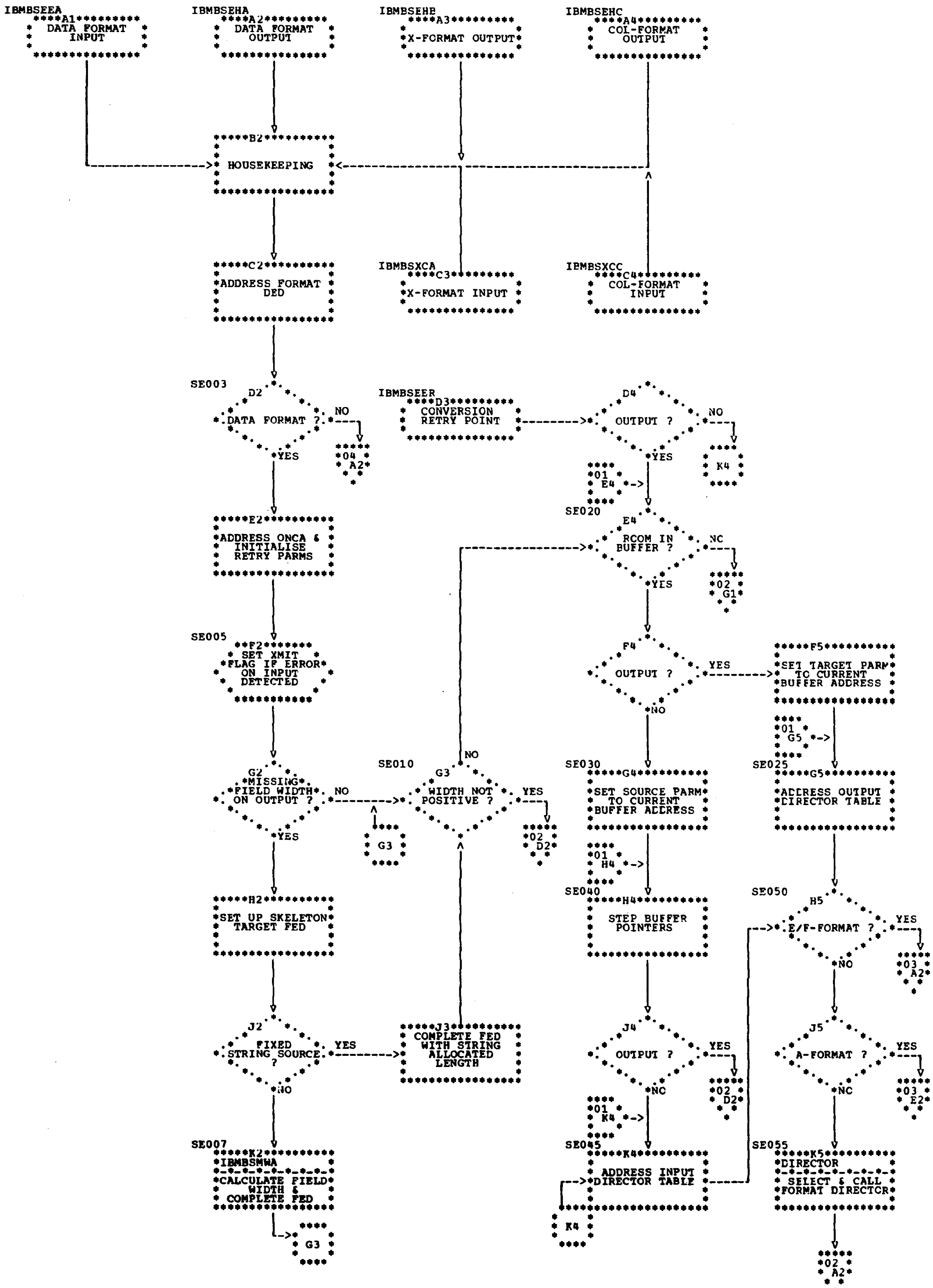


Chart DSEE. Edit-directed combination module (part 1 of 4)

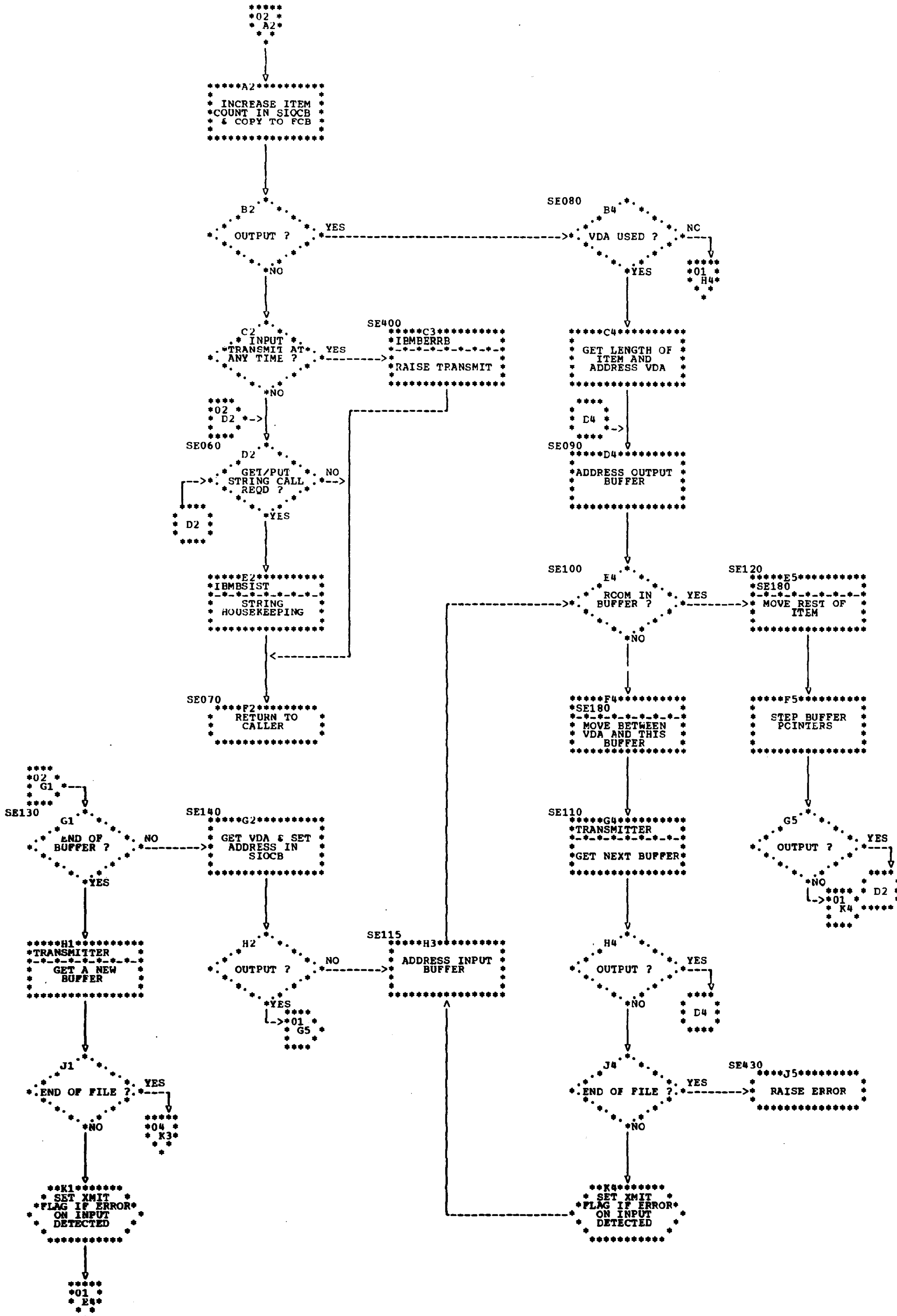


Chart DSEE. Edit-directed combination module (part 2 of 4)

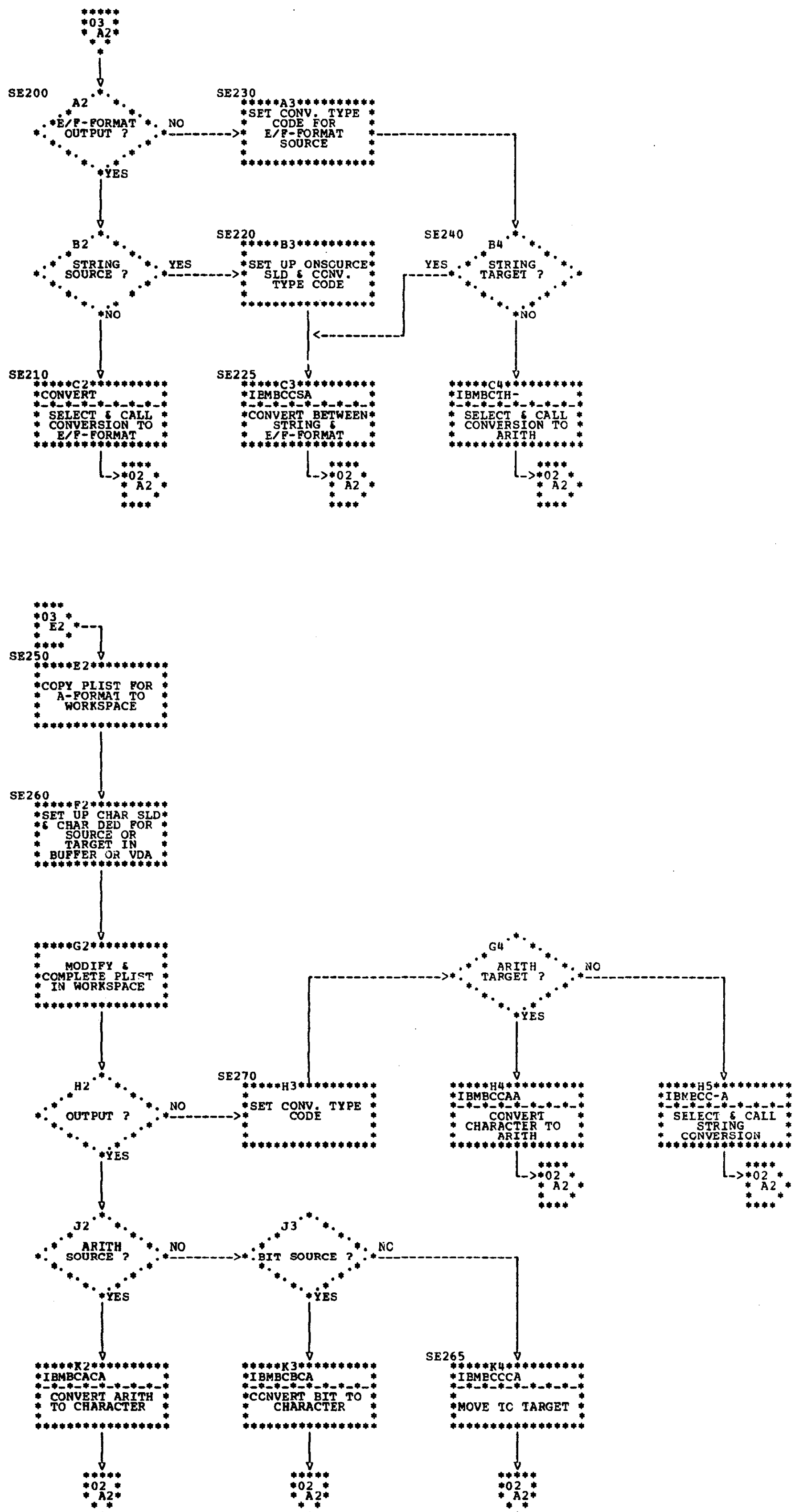


Chart DSEE. Edit-directed combination module (part 3 of 4)

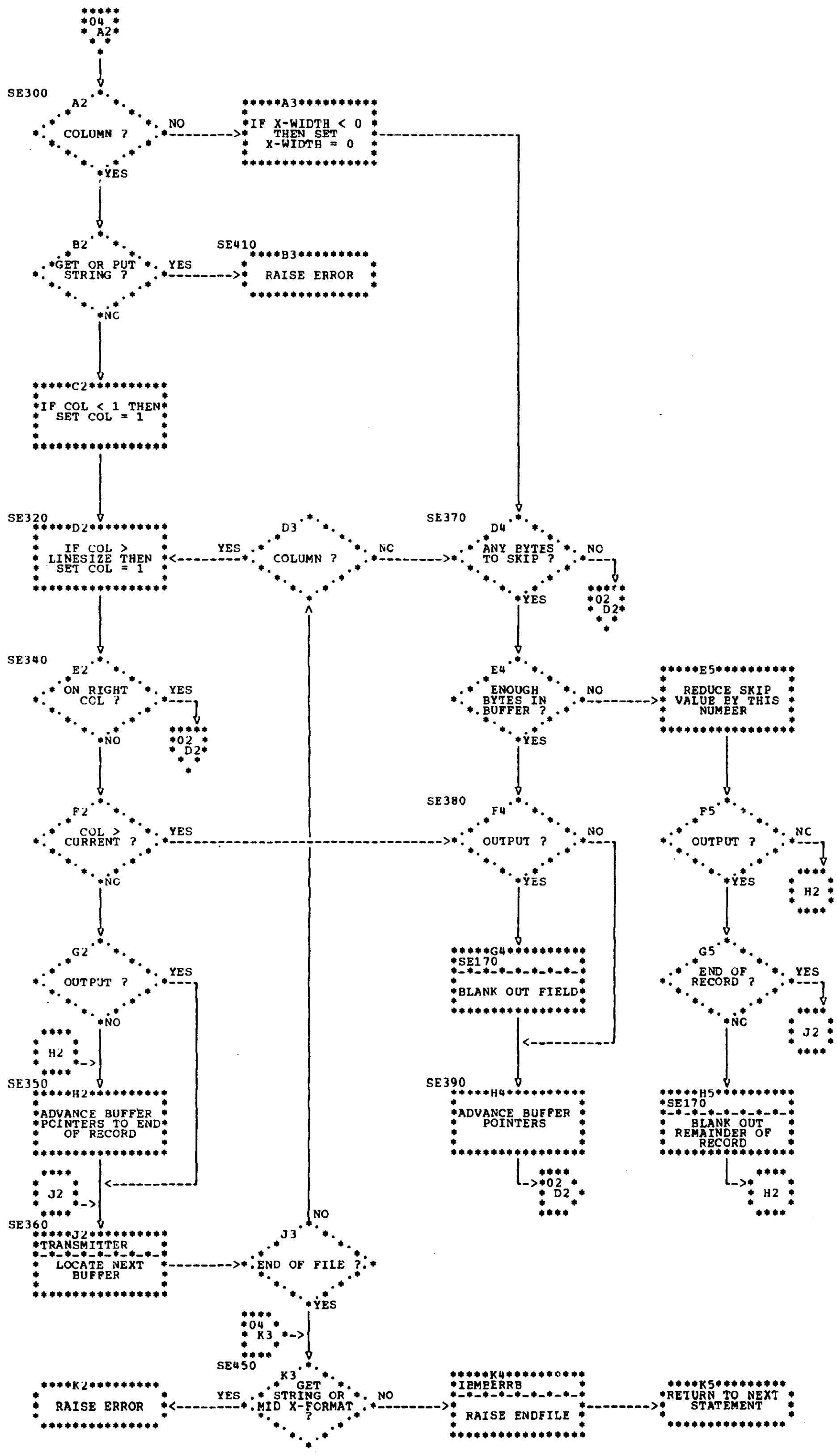


Chart DSEE. Edit-directed combination module (part 4 of 4)

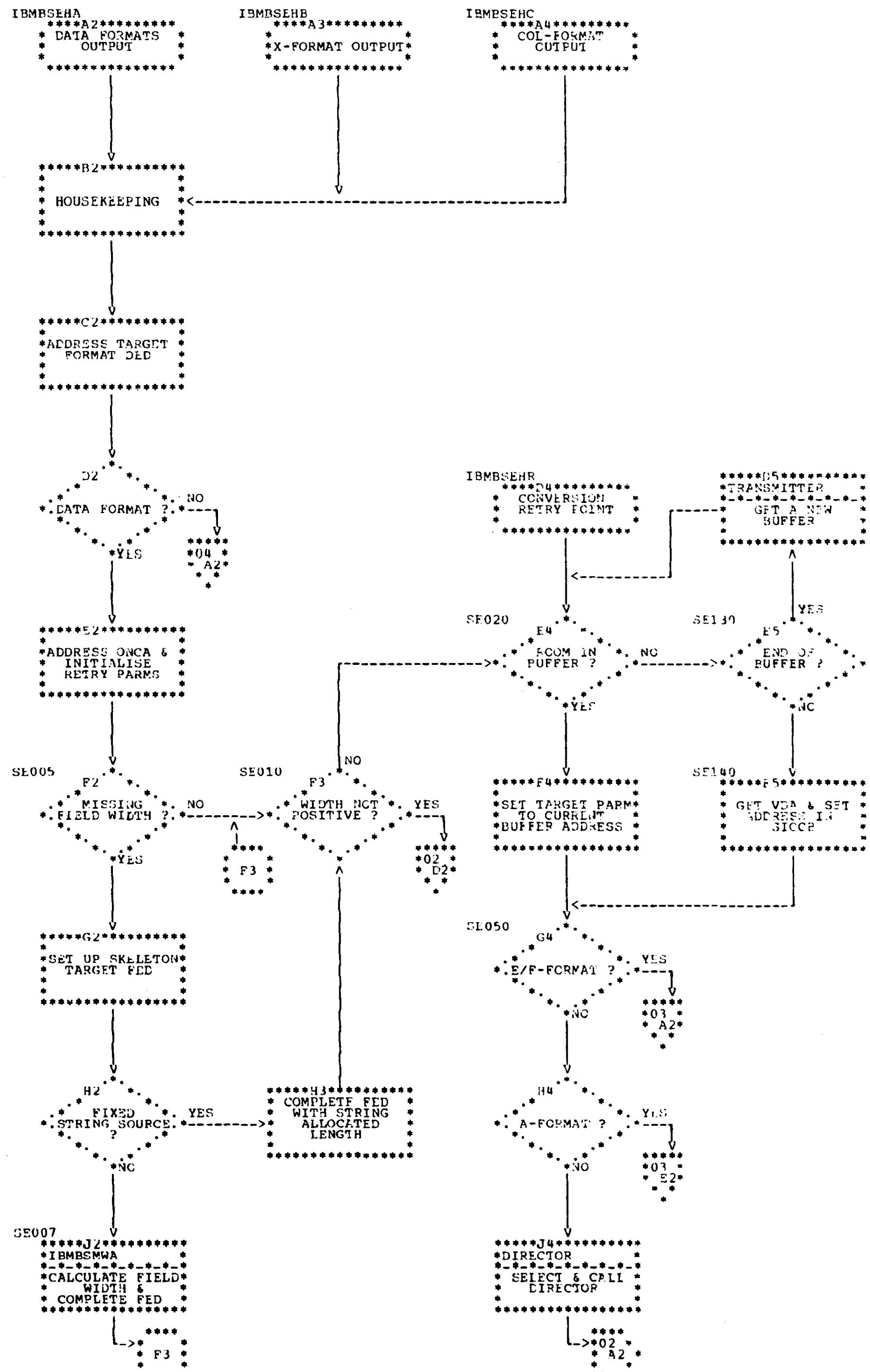


Chart DSEH. Edit-directed combination subset module (part 1 of 4)

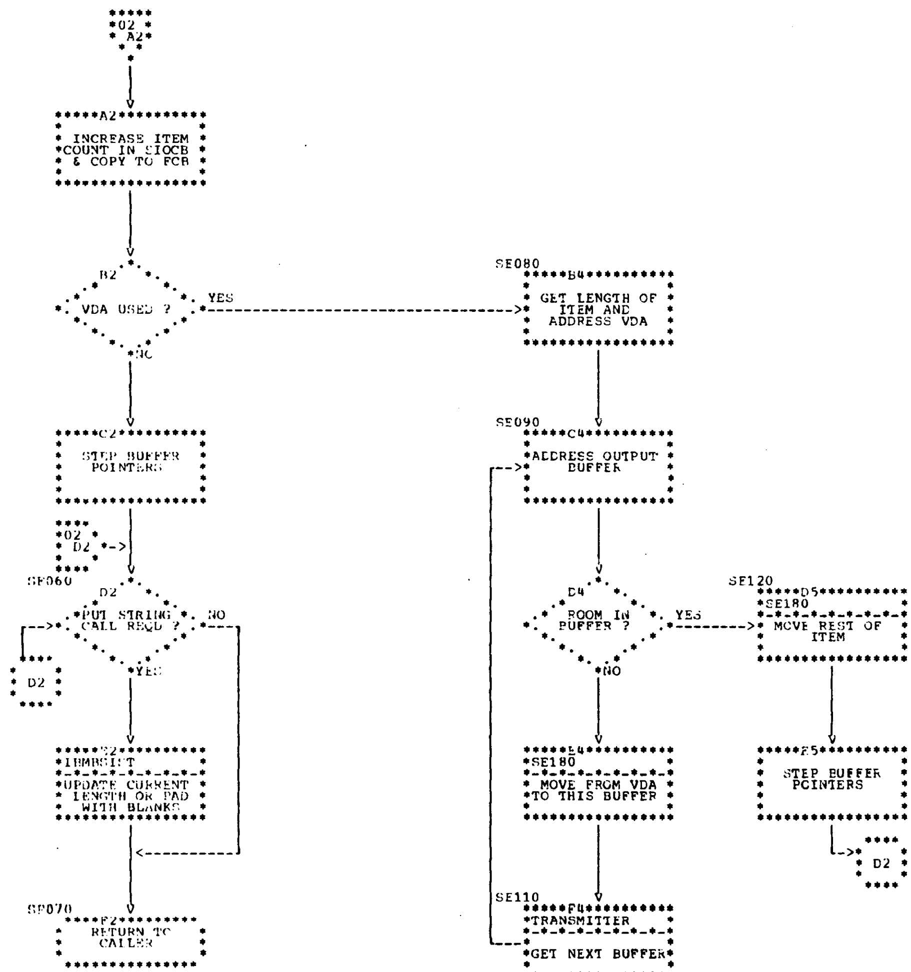


Chart DSEH. Edit-directed combination subset module (part 2 of 4)

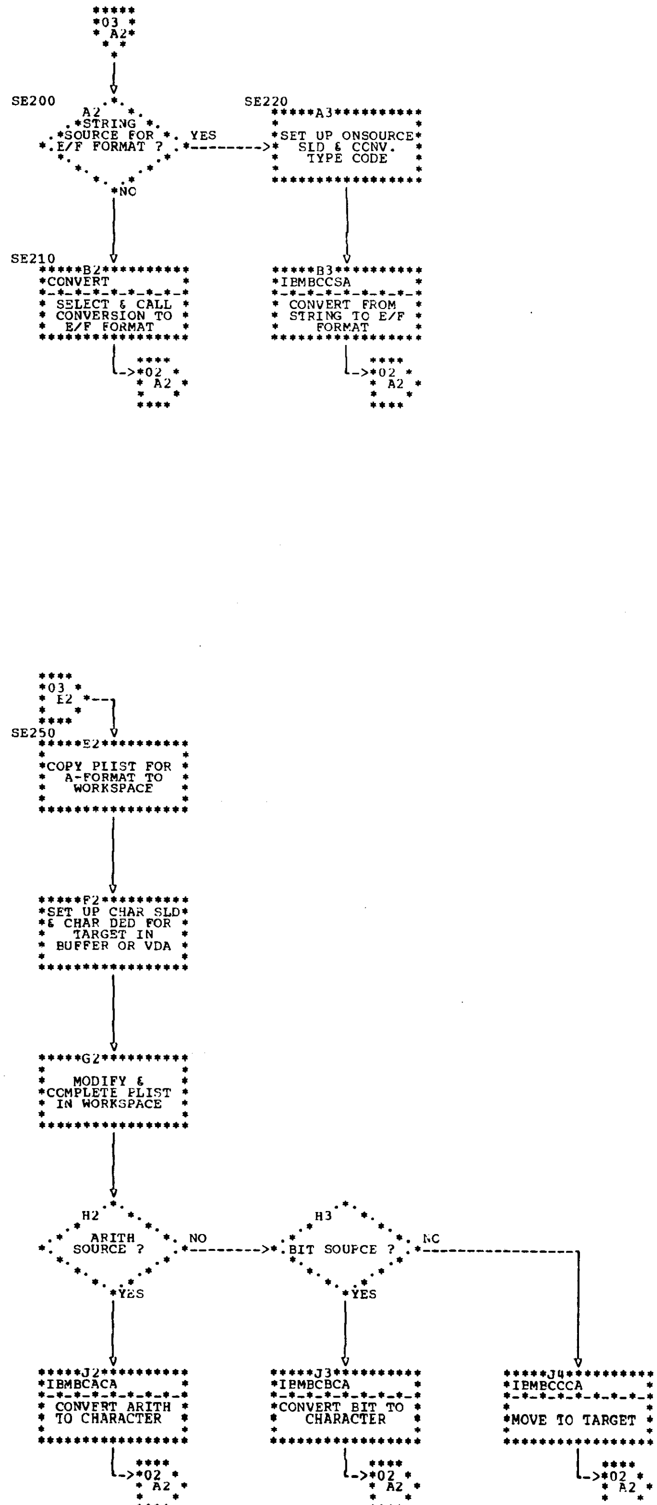


Chart DSEH. Edit-directed combination subset module (part 3 of 4)

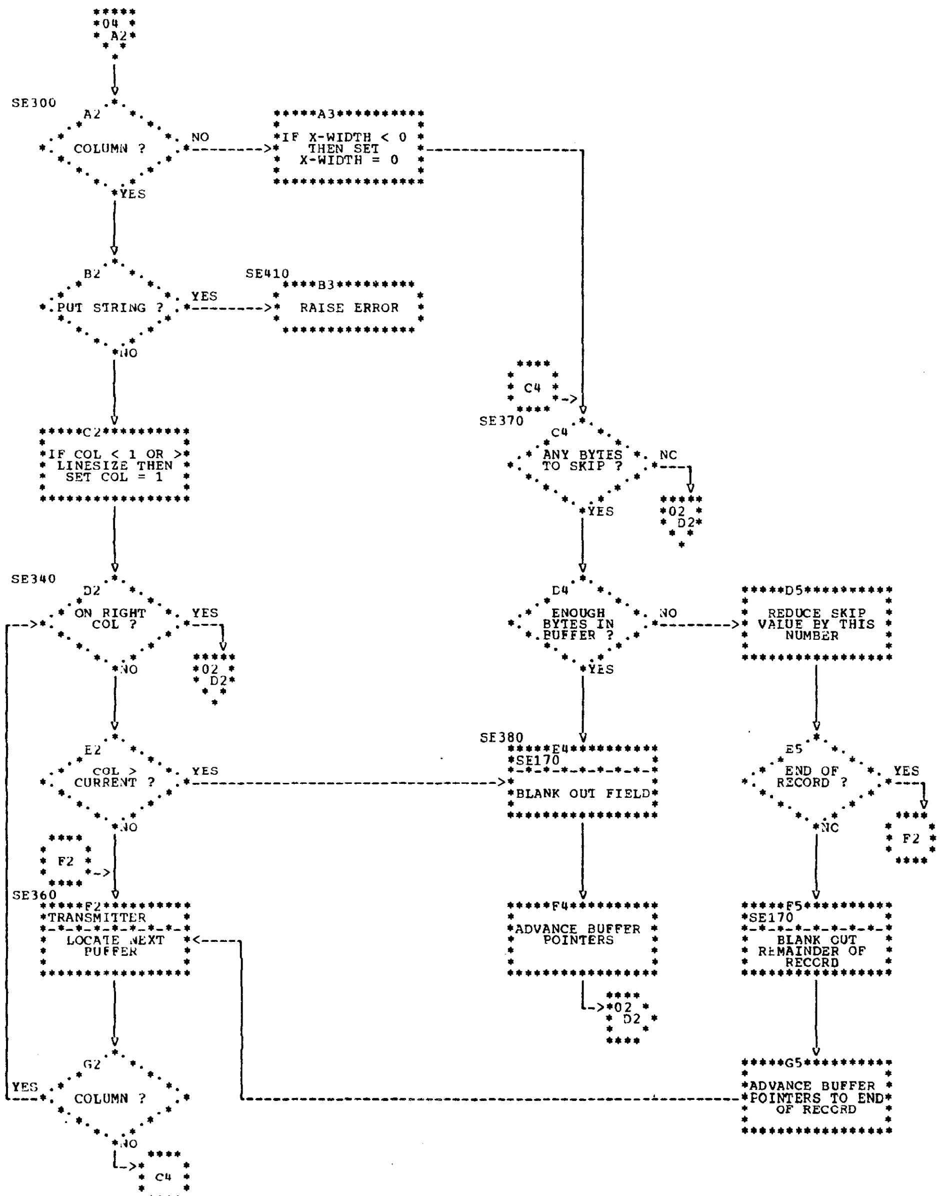


Chart DSEH. Edit-directed combination subset module (part 4 of 4)

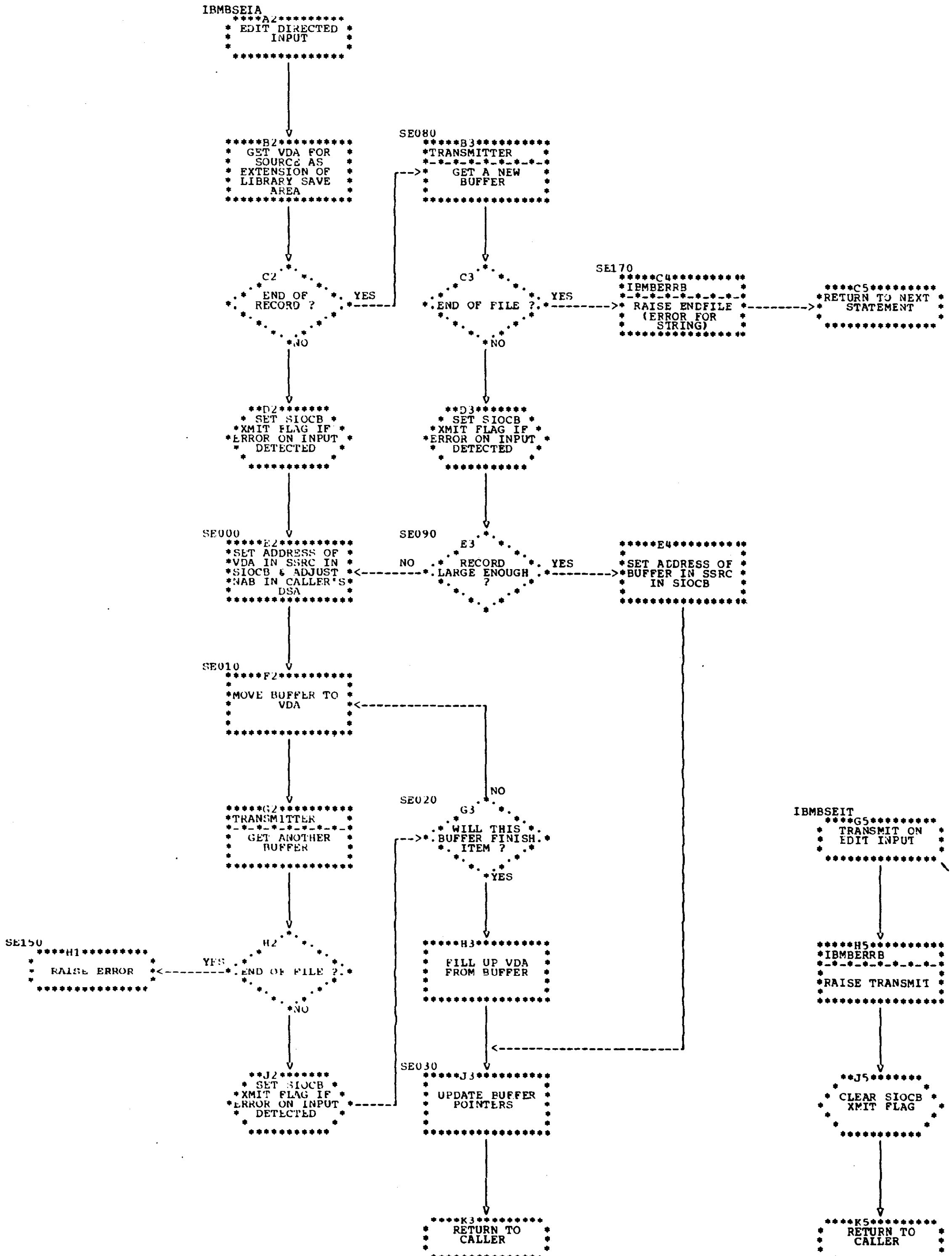


Chart DSEI. Edit-directed Input

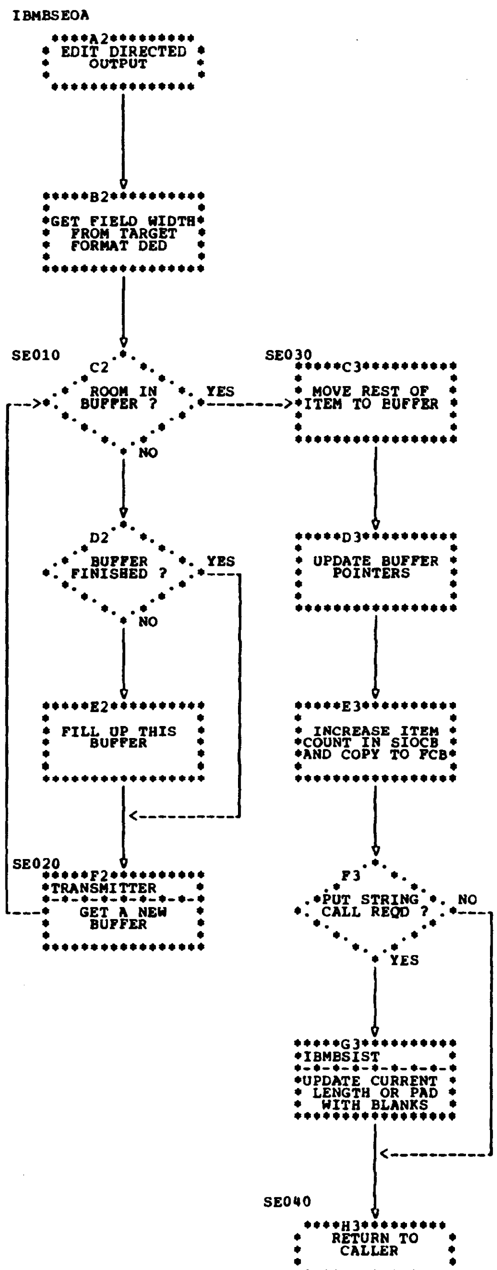


Chart DSEO. Edit-directed Output

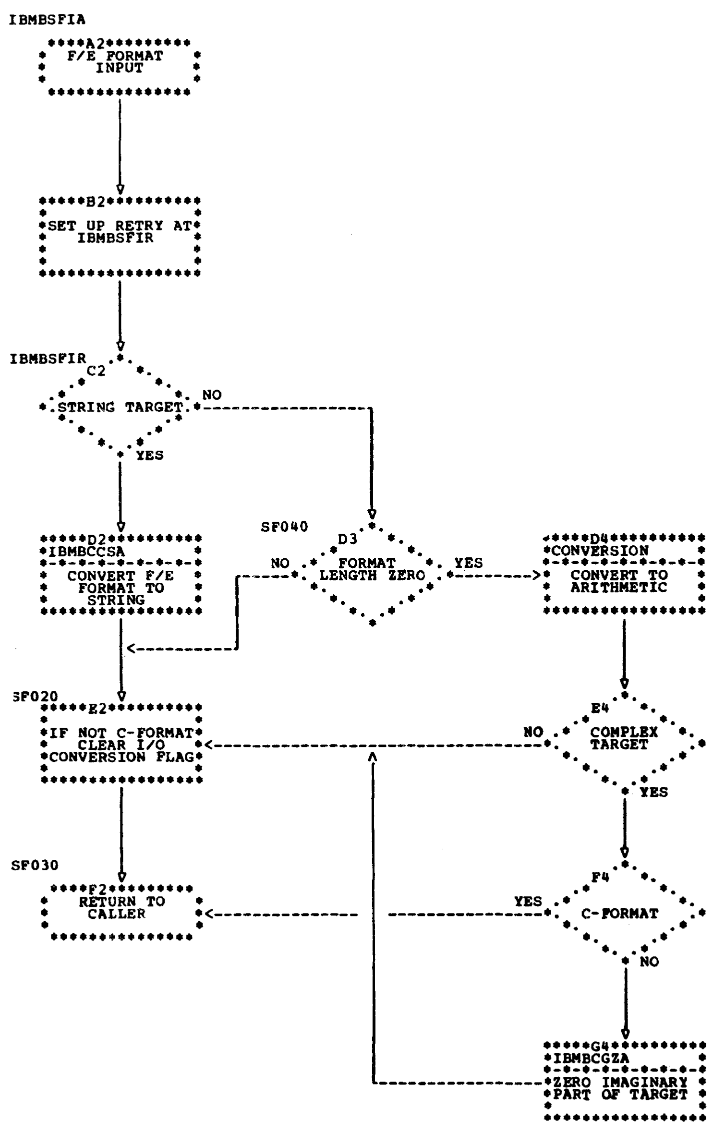


Chart BSFI. Input conversion director (F and E format)

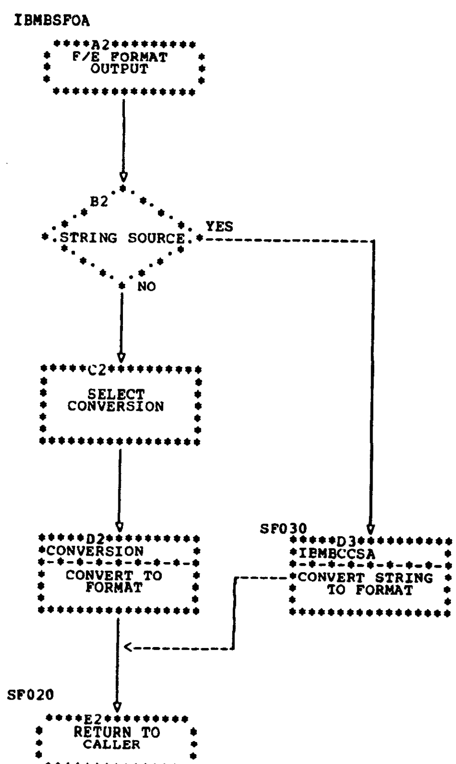


Chart BSFO. Output conversion director (F and E format)

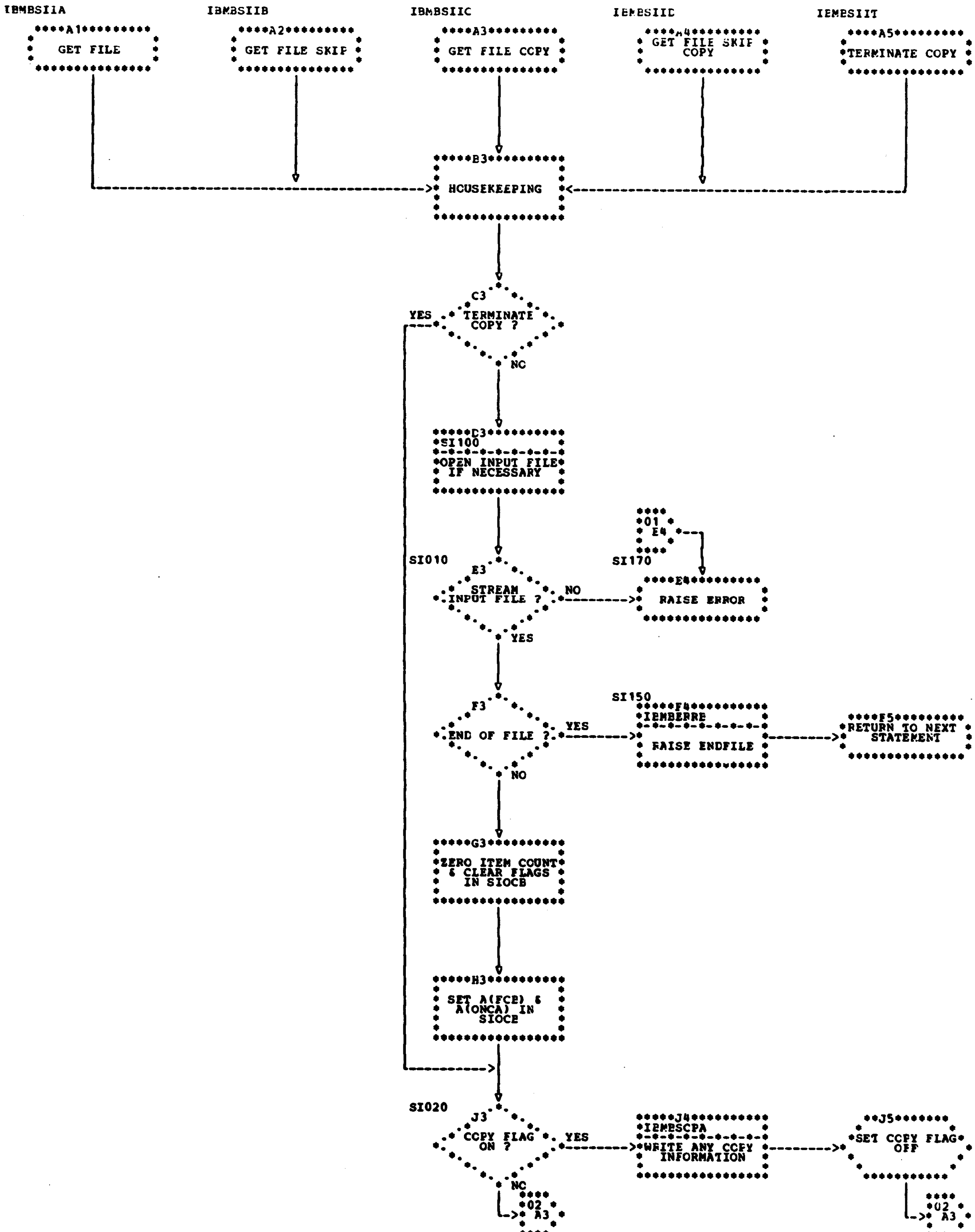


Chart DSII. GET FILE initialization (part 1 of 2)

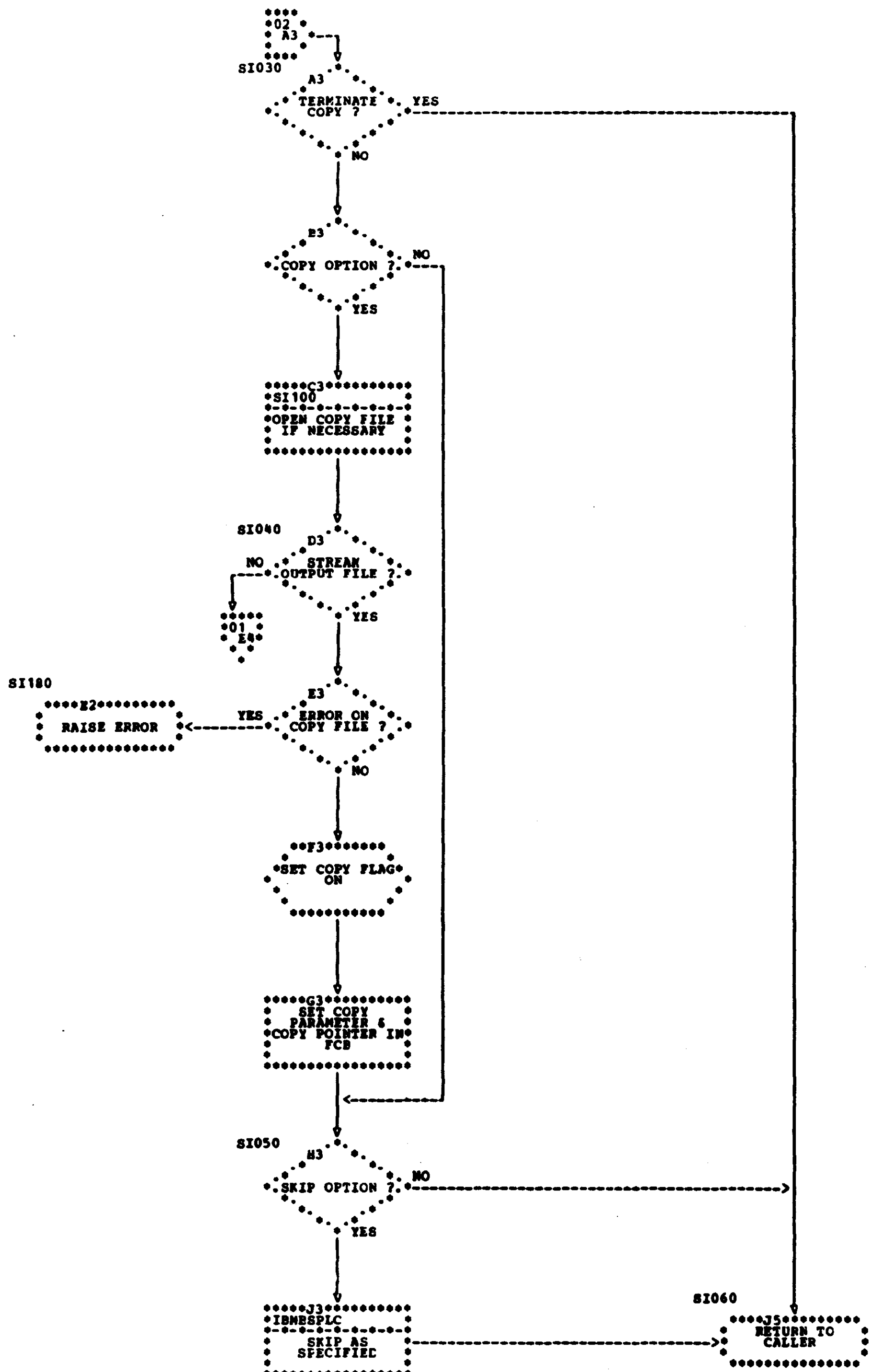


Chart DSII. GET FILE initialization (part 2 of 2)

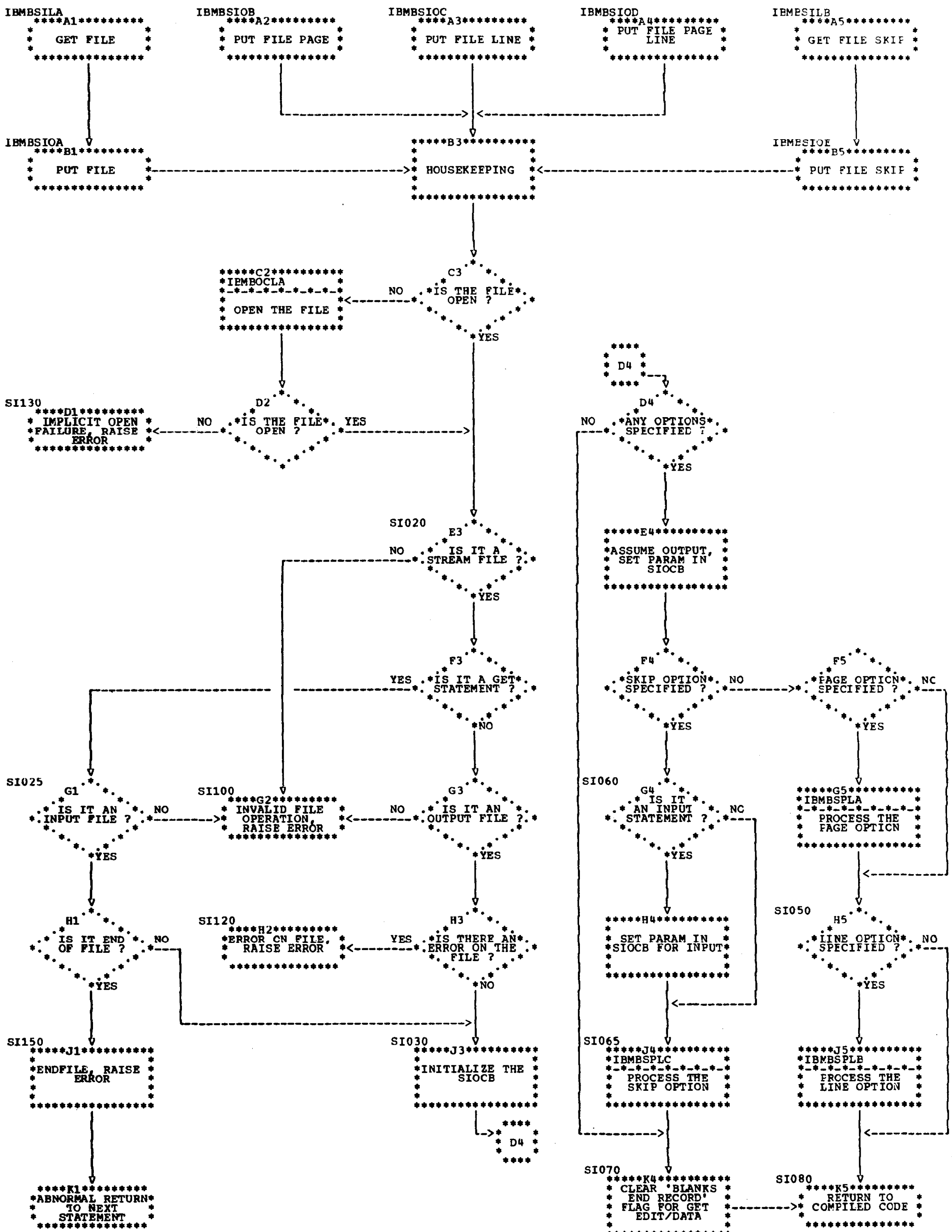
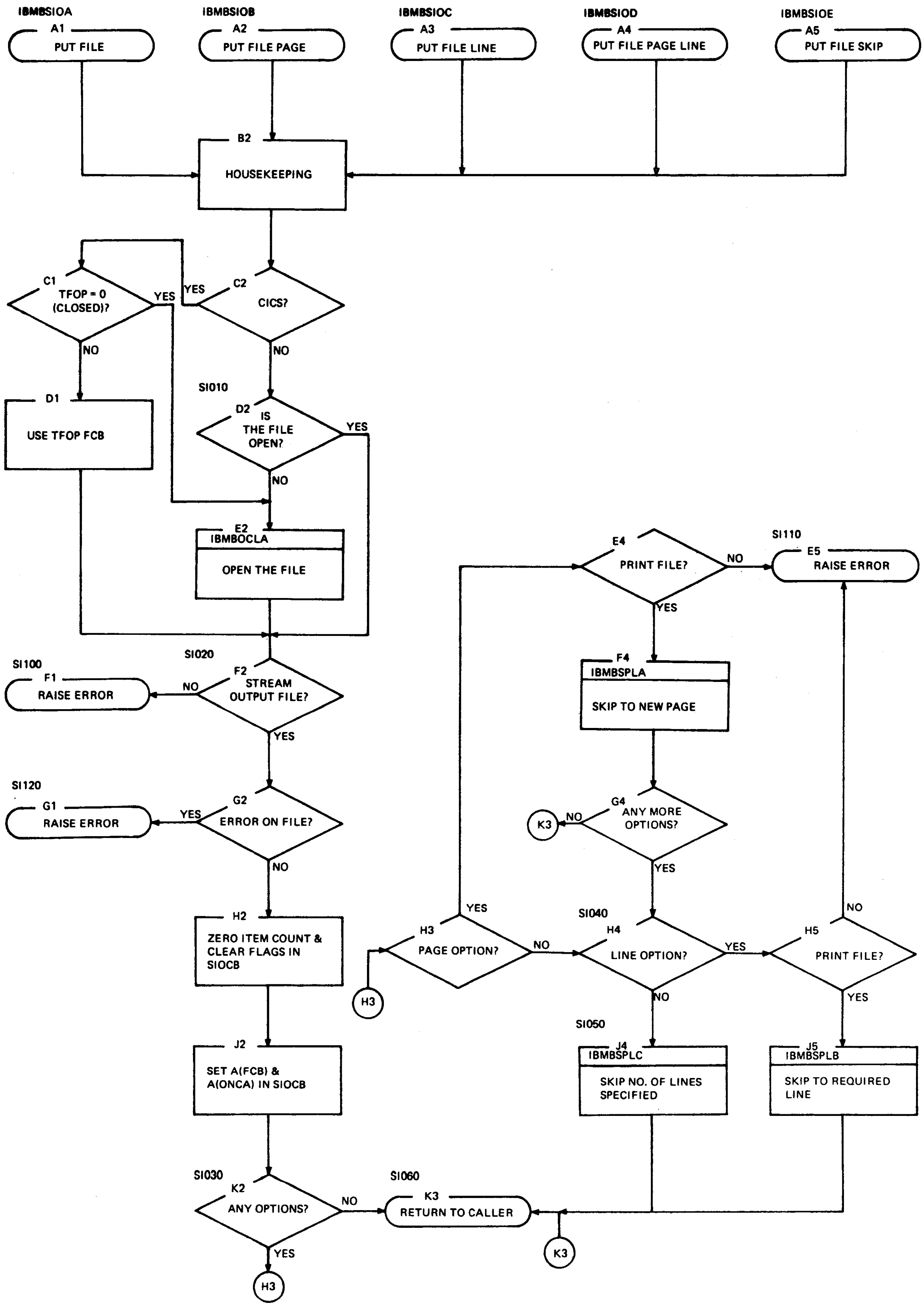


Chart DSIL. FILE initialization



|Chart DSIO. PUT FILE initialization

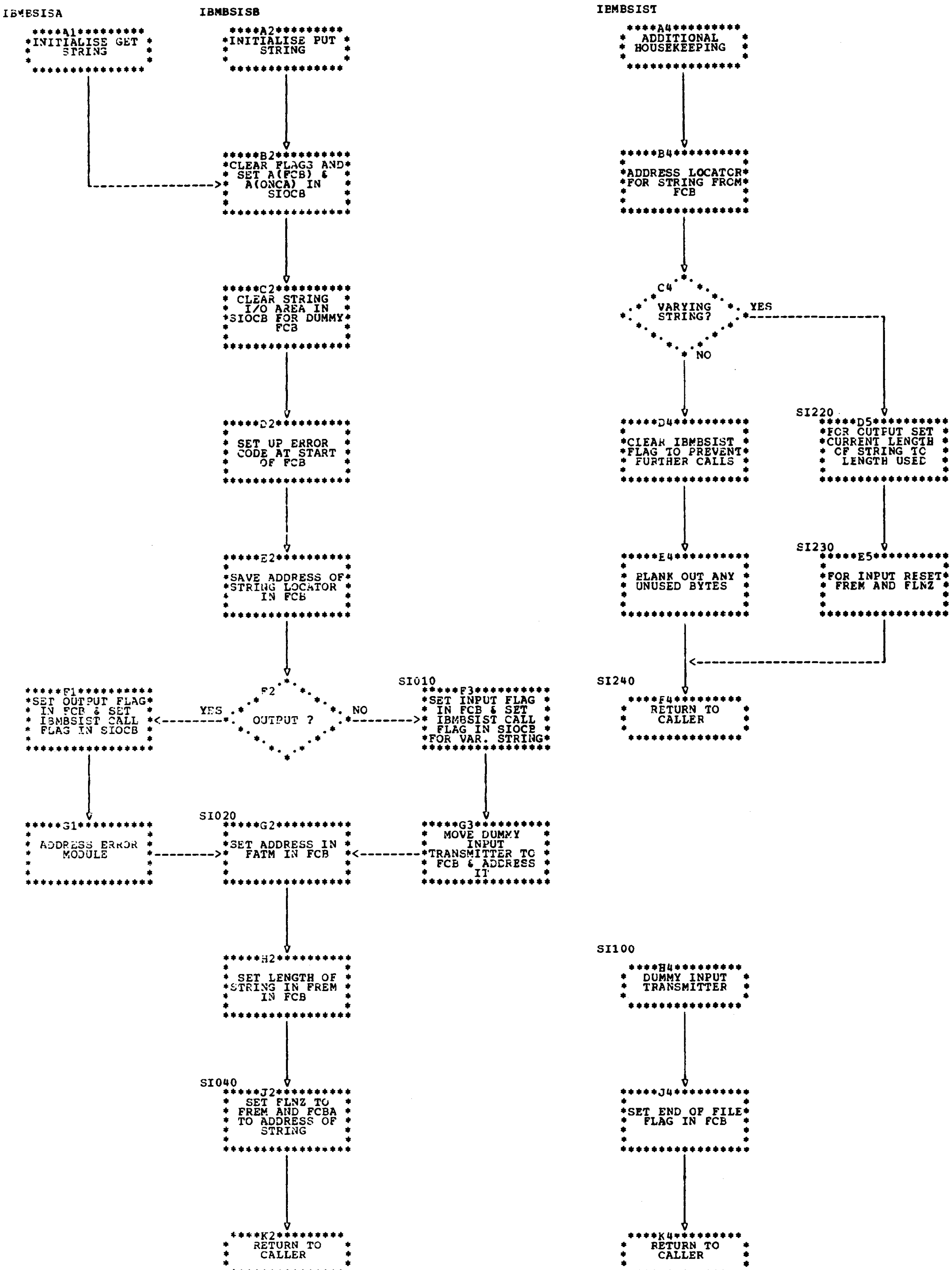


Chart DSIS. GET or PUT STRING

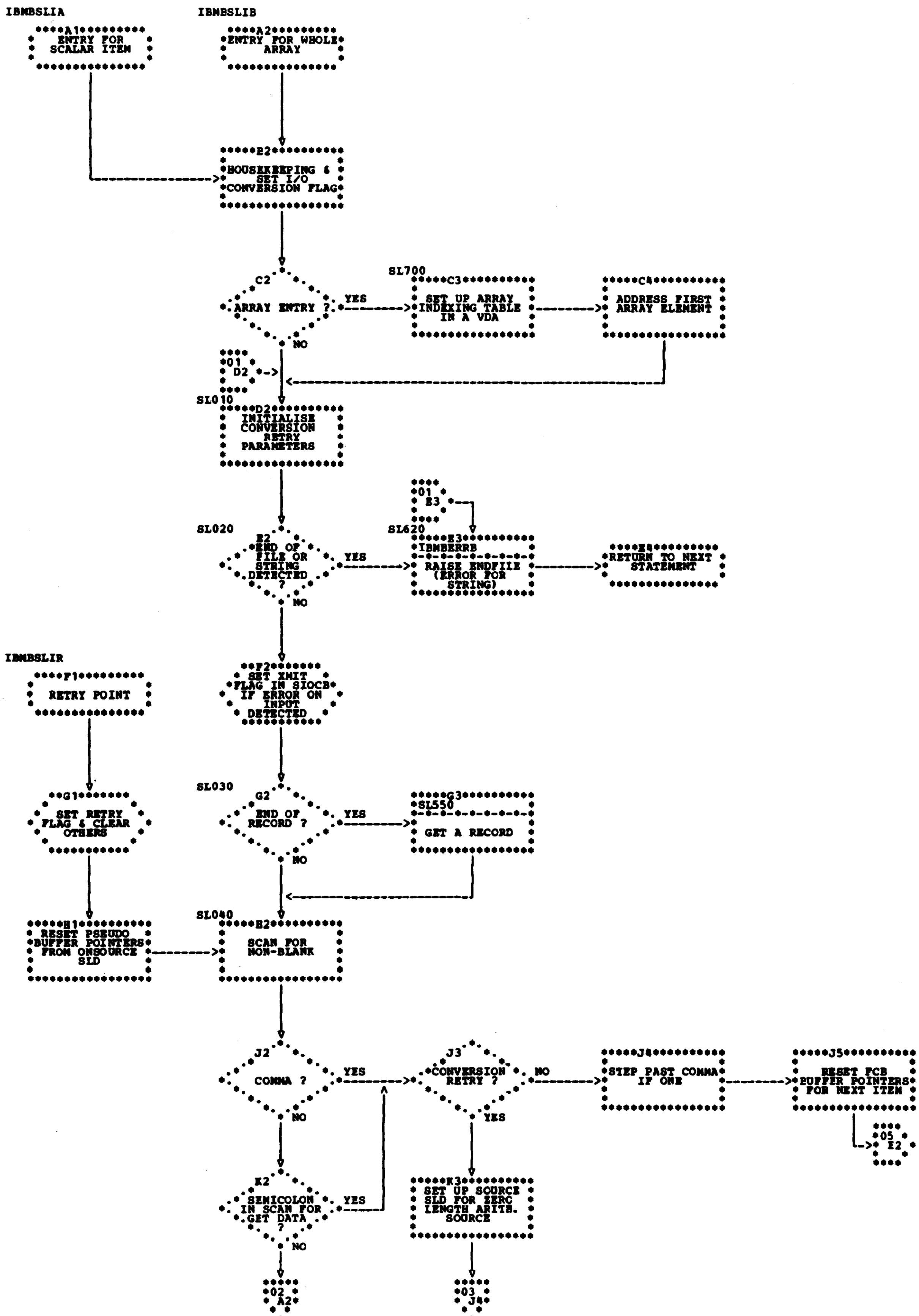


Chart DSLI. List-directed Input (part 1 of 8)

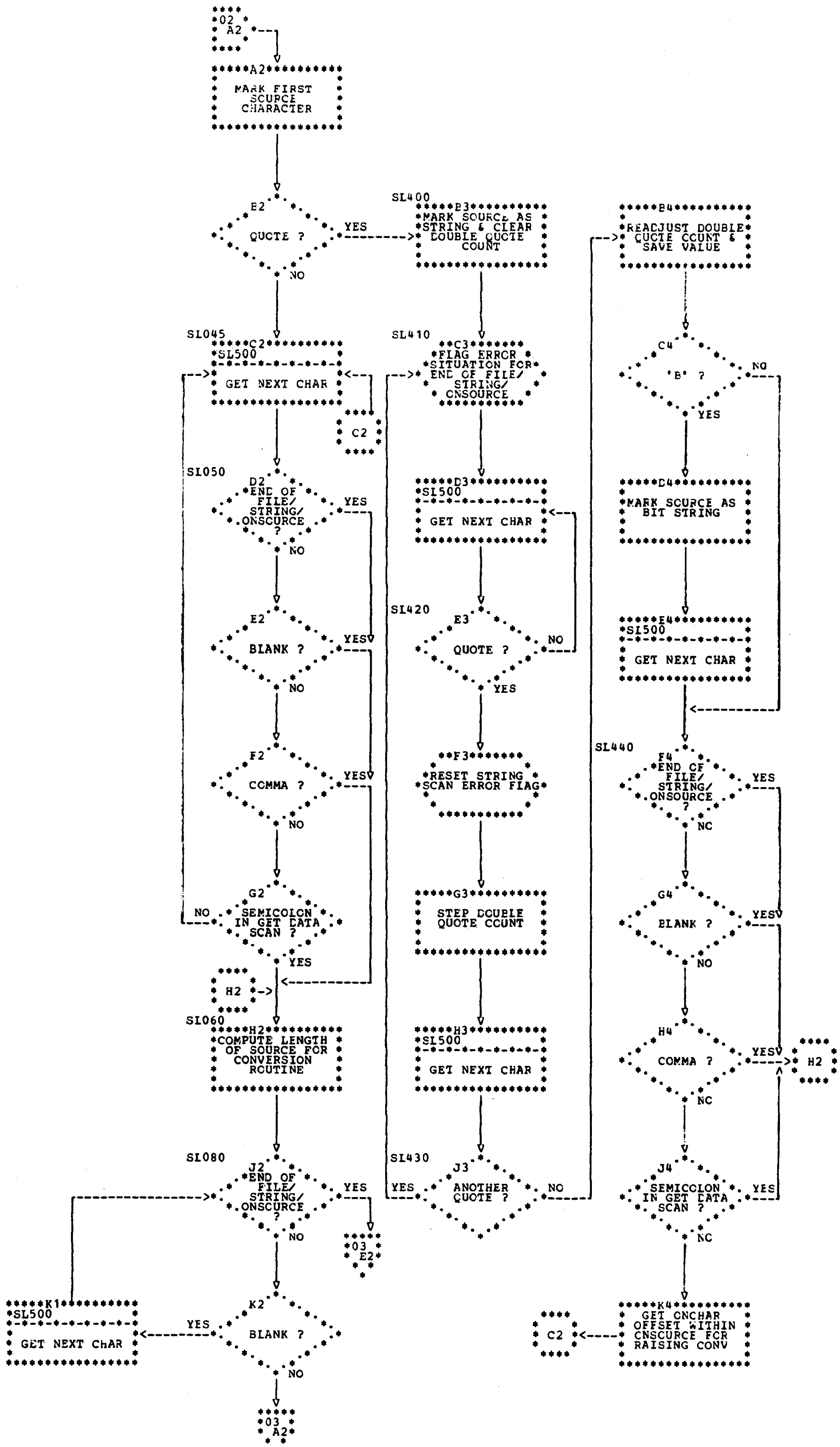


Chart DSLI. List-directed Input (part 2 of 8)

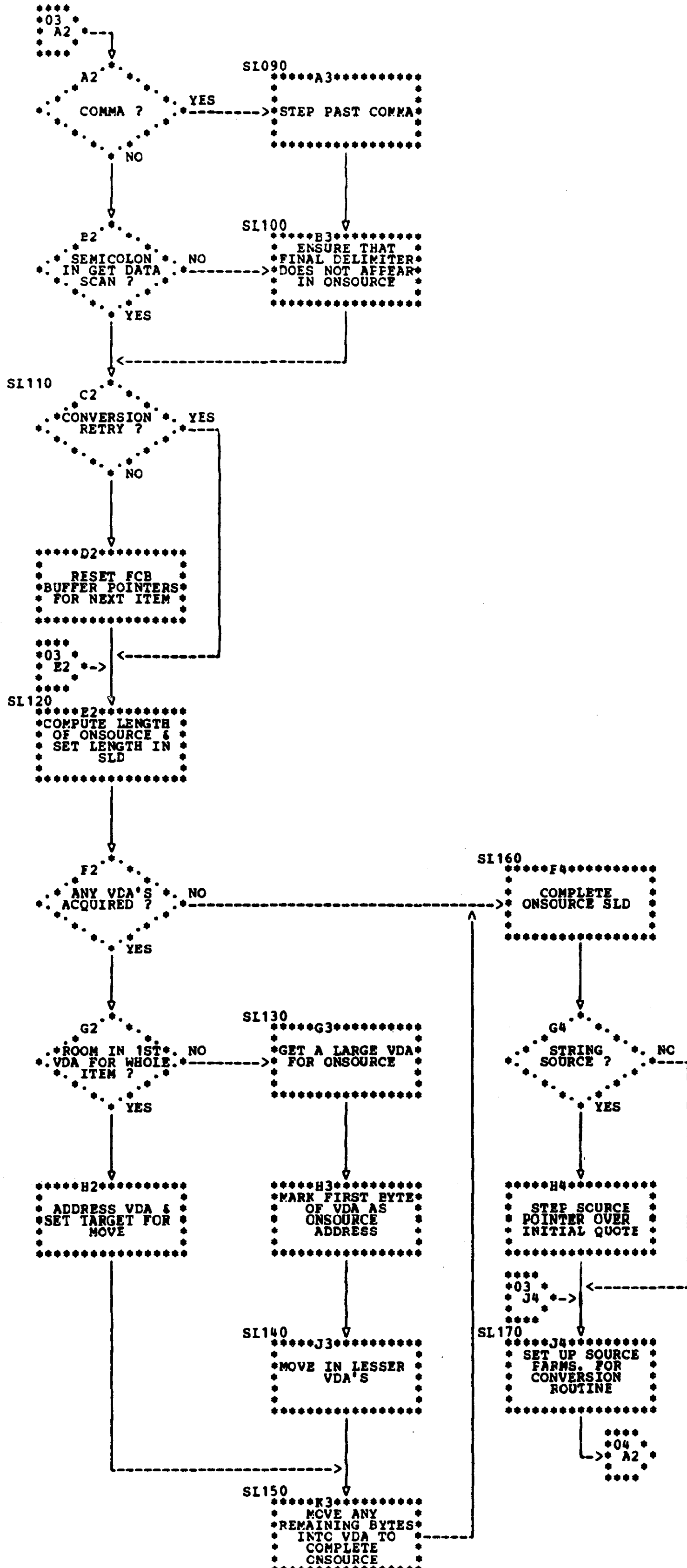


Chart DSLI. List-directed Input (part 3 of 8)

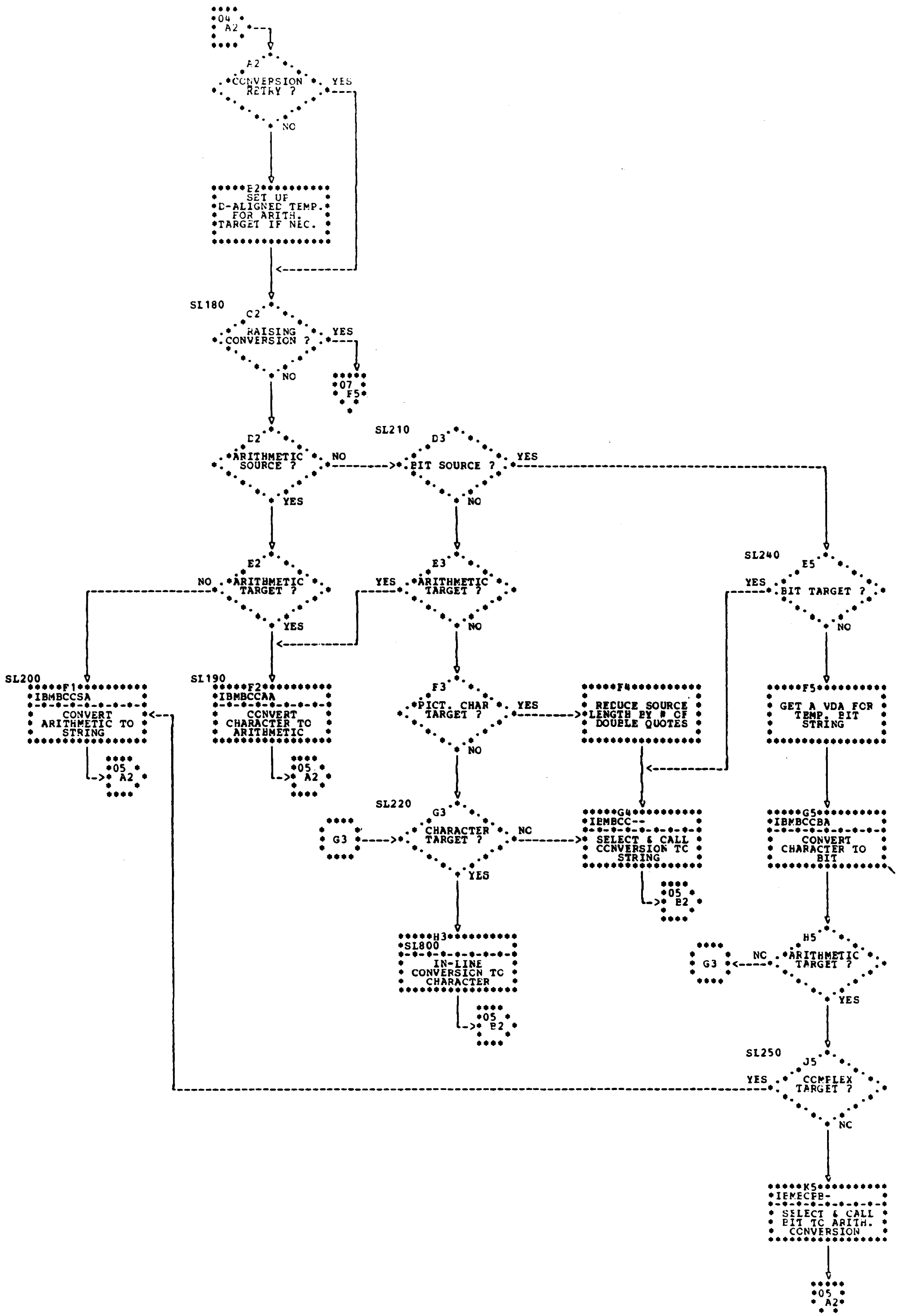


Chart DSLI. List-directed Input (part 4 of 8)

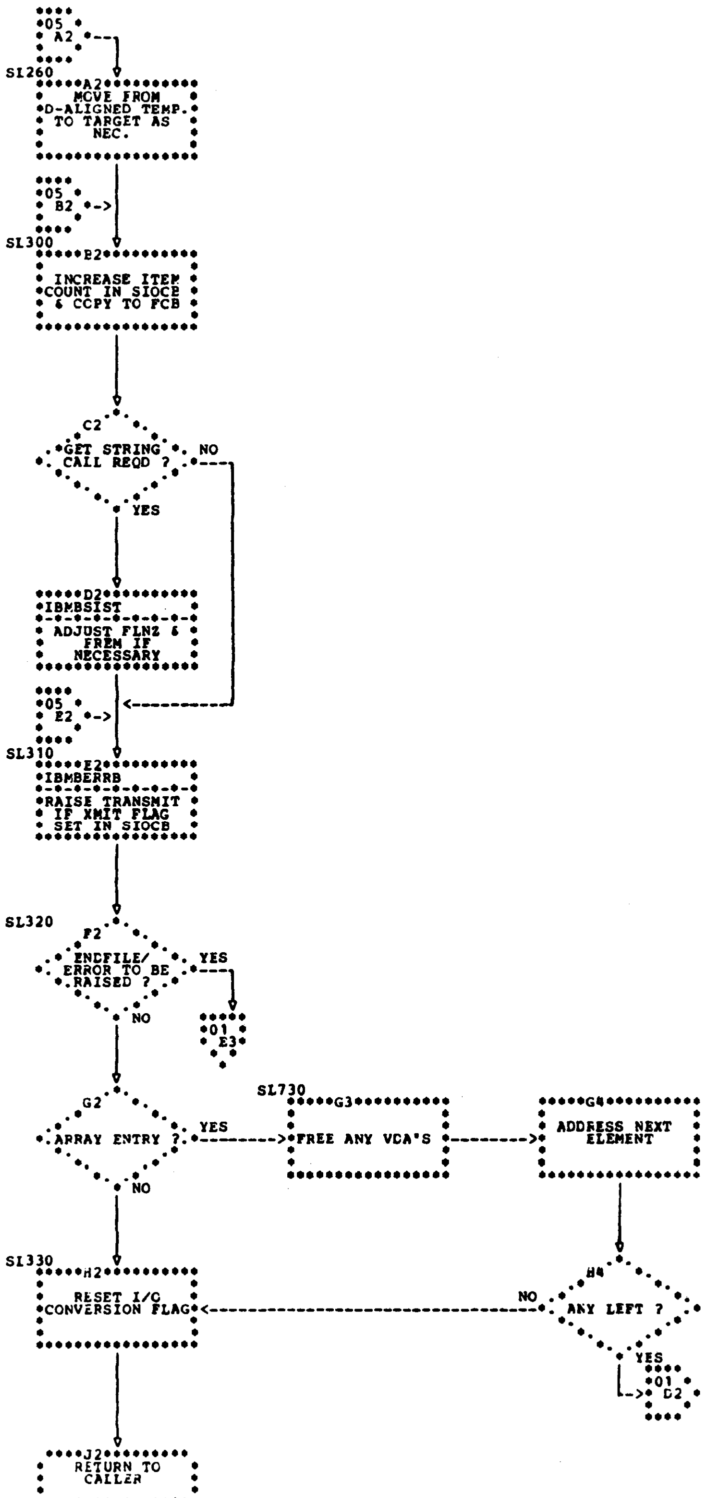


Chart DSLI. List-directed Input (part 5 of 8)

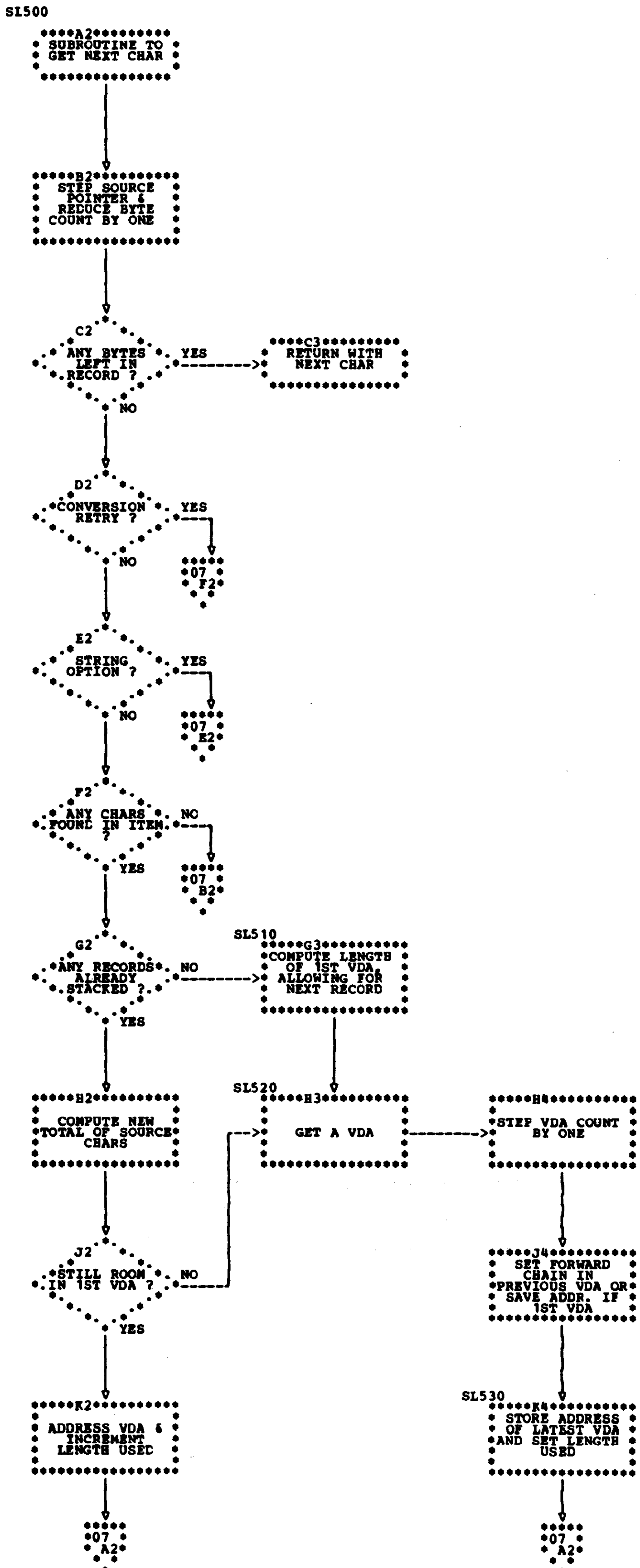


Chart DSLI. List-directed Input (part 6 of 8)

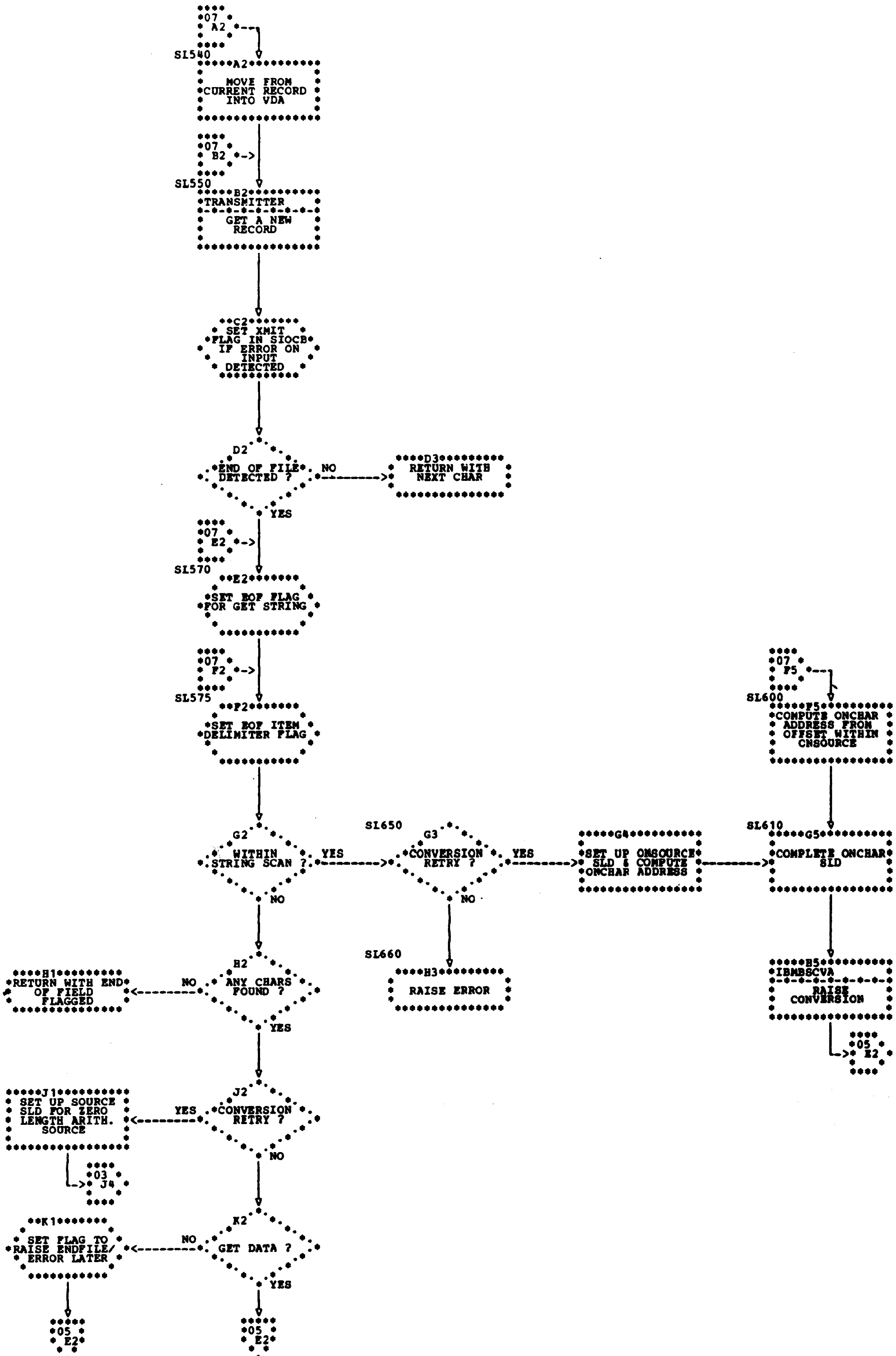


Chart DSLI. List-directed Input (part 7 of 8)

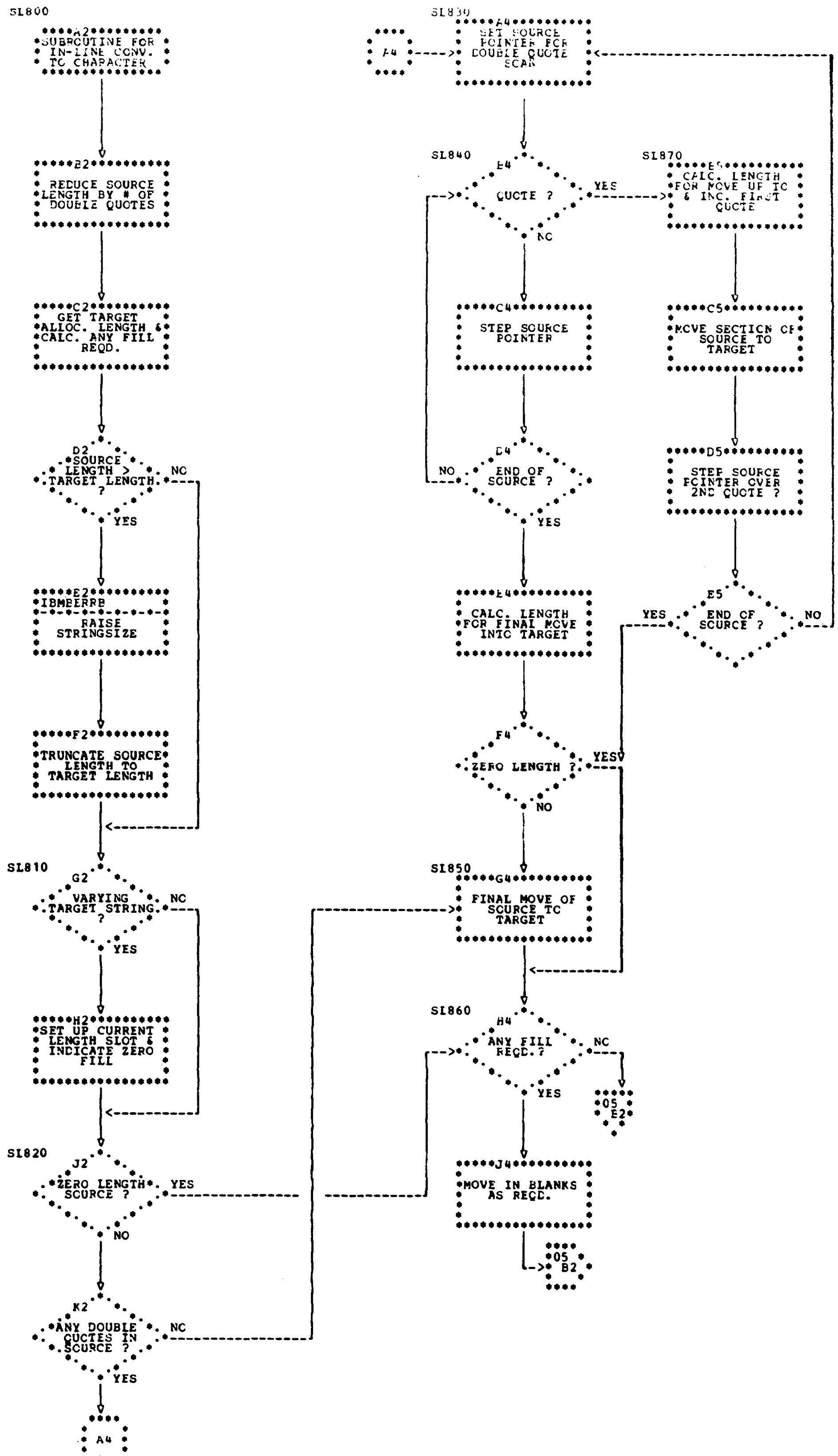


Chart DSLI. List-directed Input (part 8 of 8)

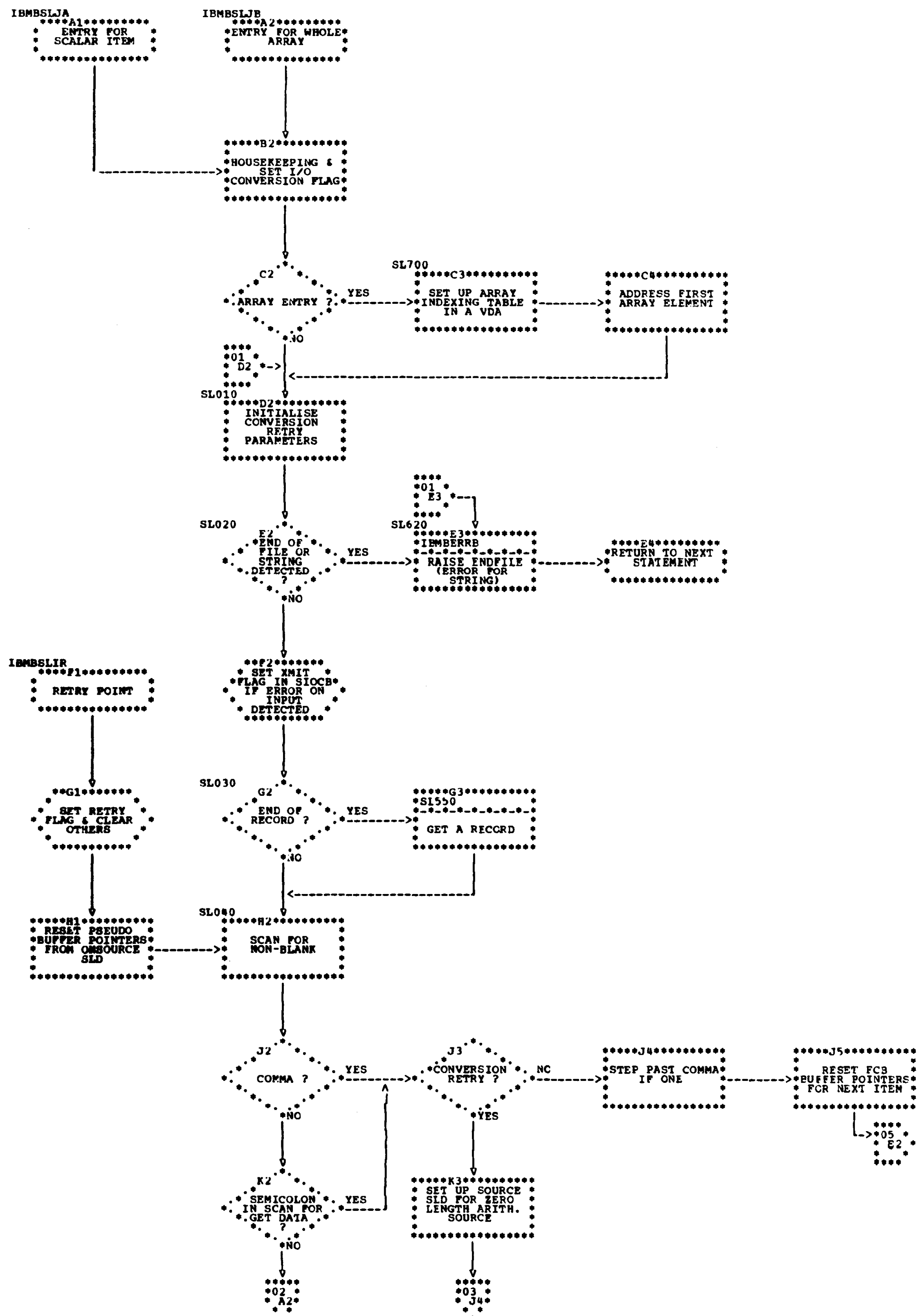


Chart DSLJ. List-directed Input, restricted conversions (part 1 of 8)

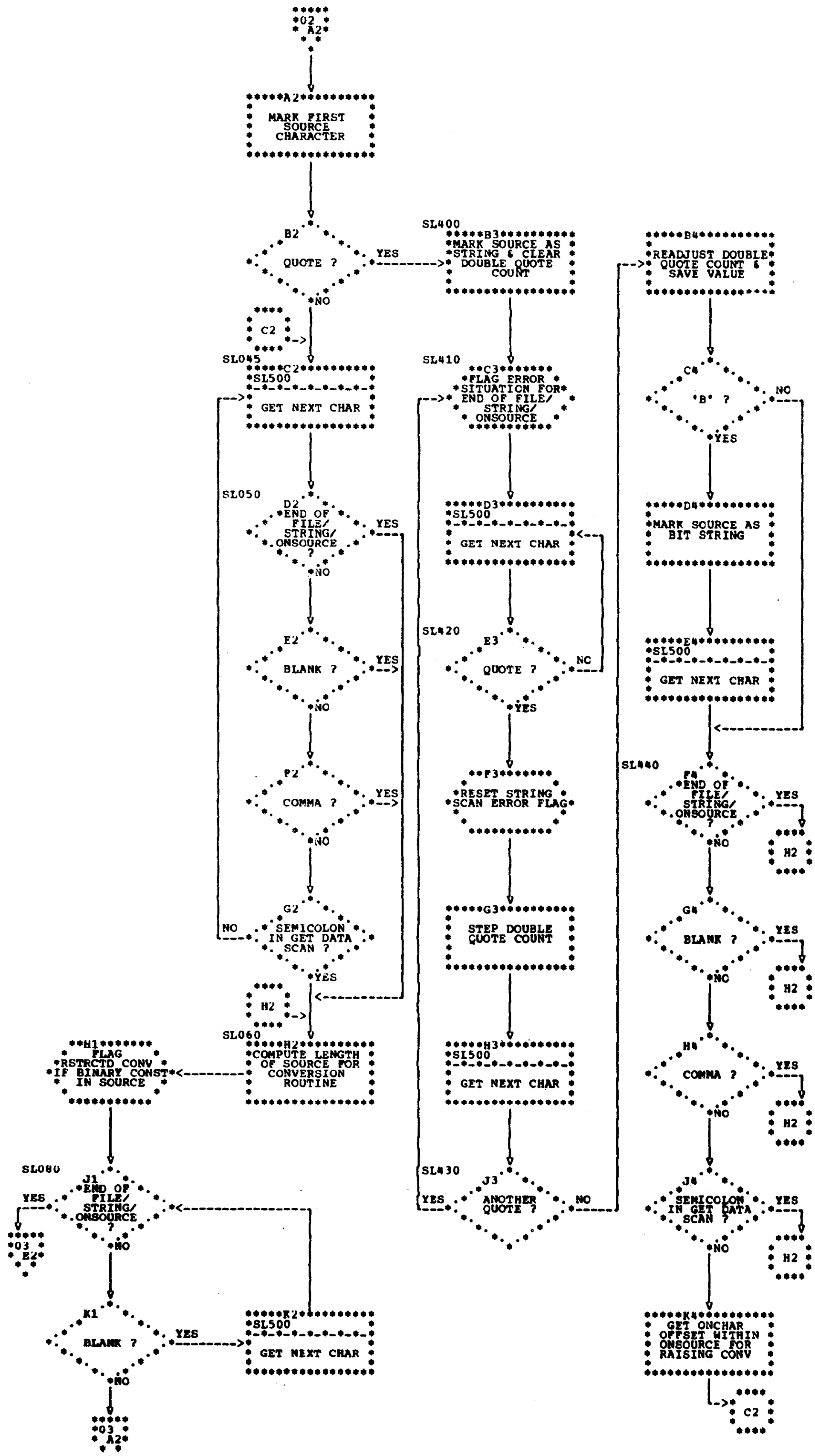


Chart DSLJ. List-directed Input, restricted conversions (part 2 of 8)

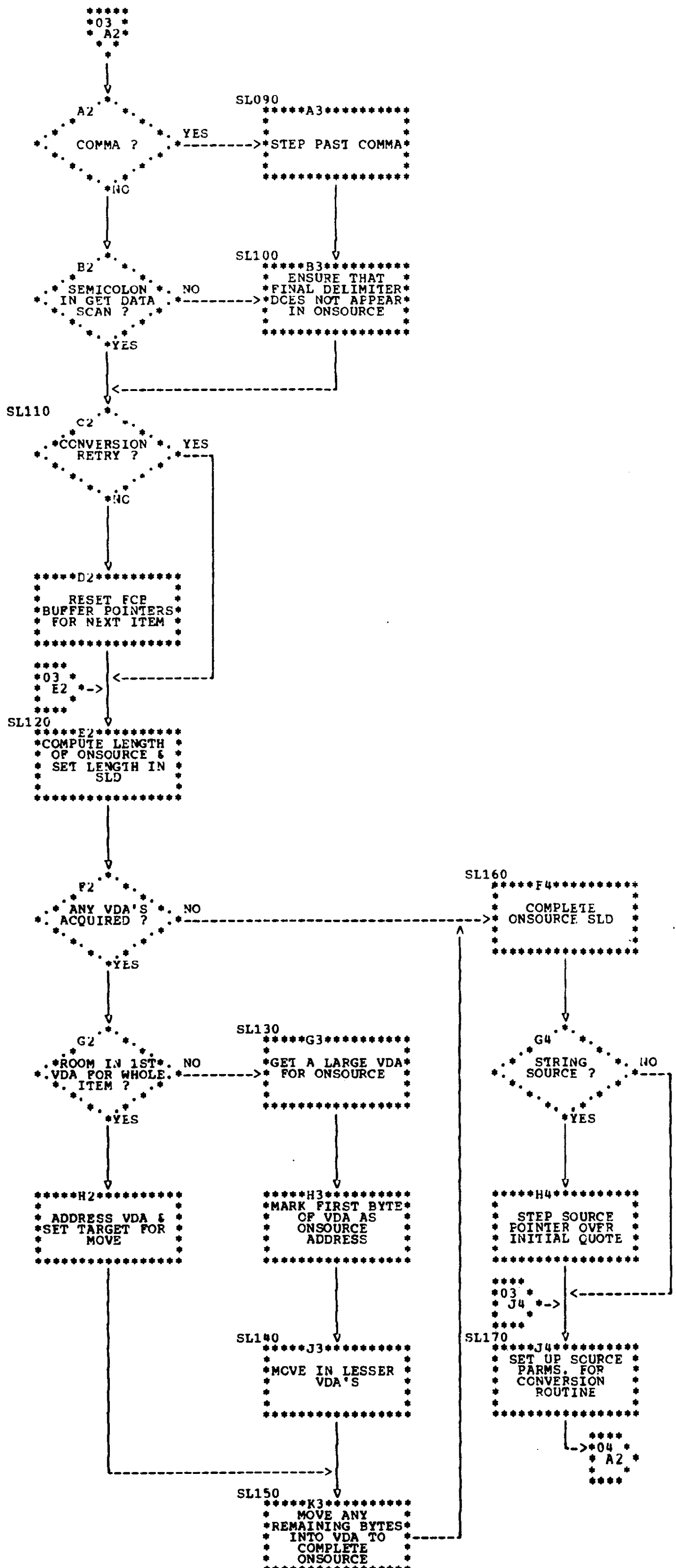


Chart DSLJ. List-directed Input, restricted conversions (part 3 of 8)

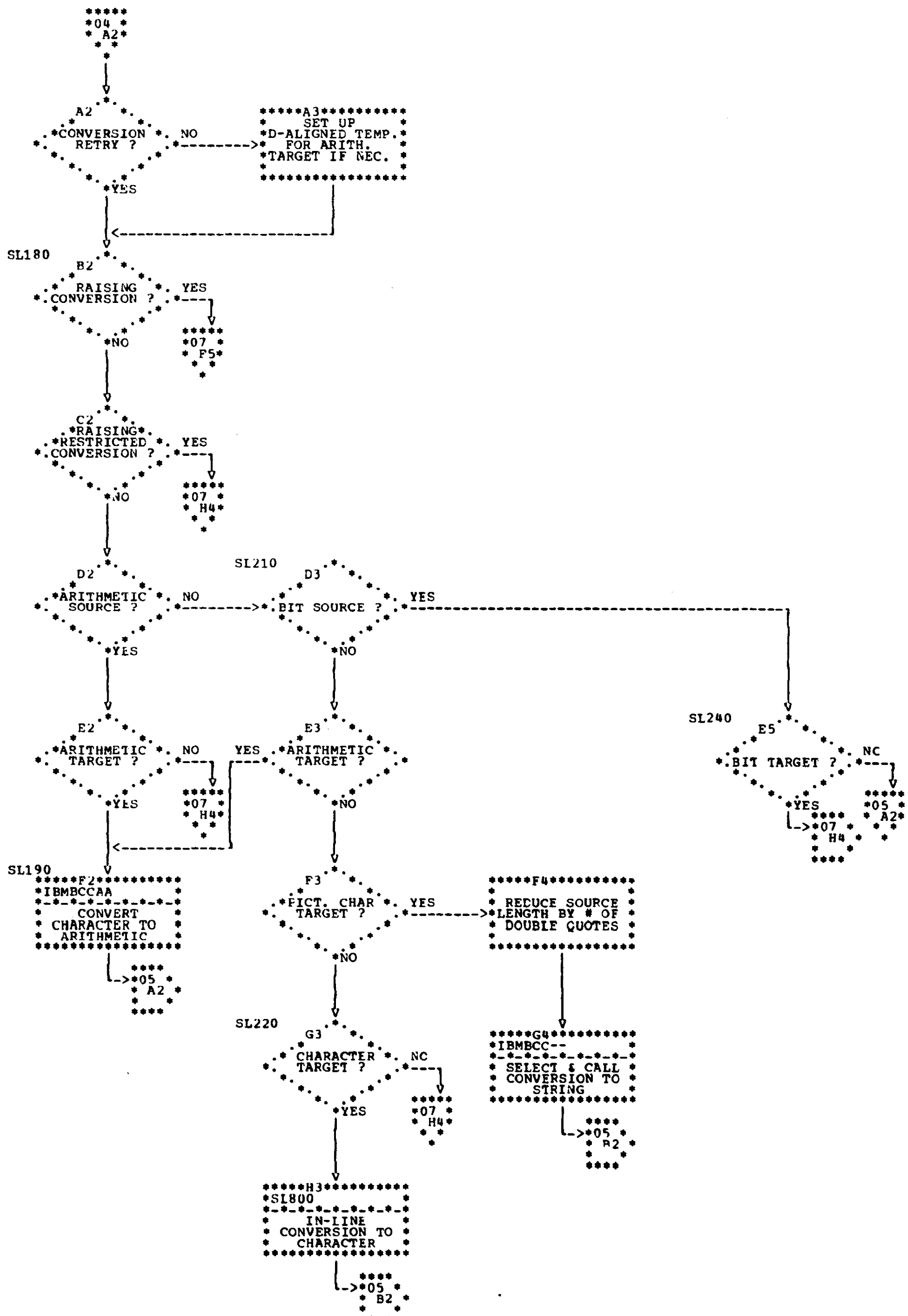


Chart DSLJ. List-directed Input, restricted conversions (part 4 of 8)

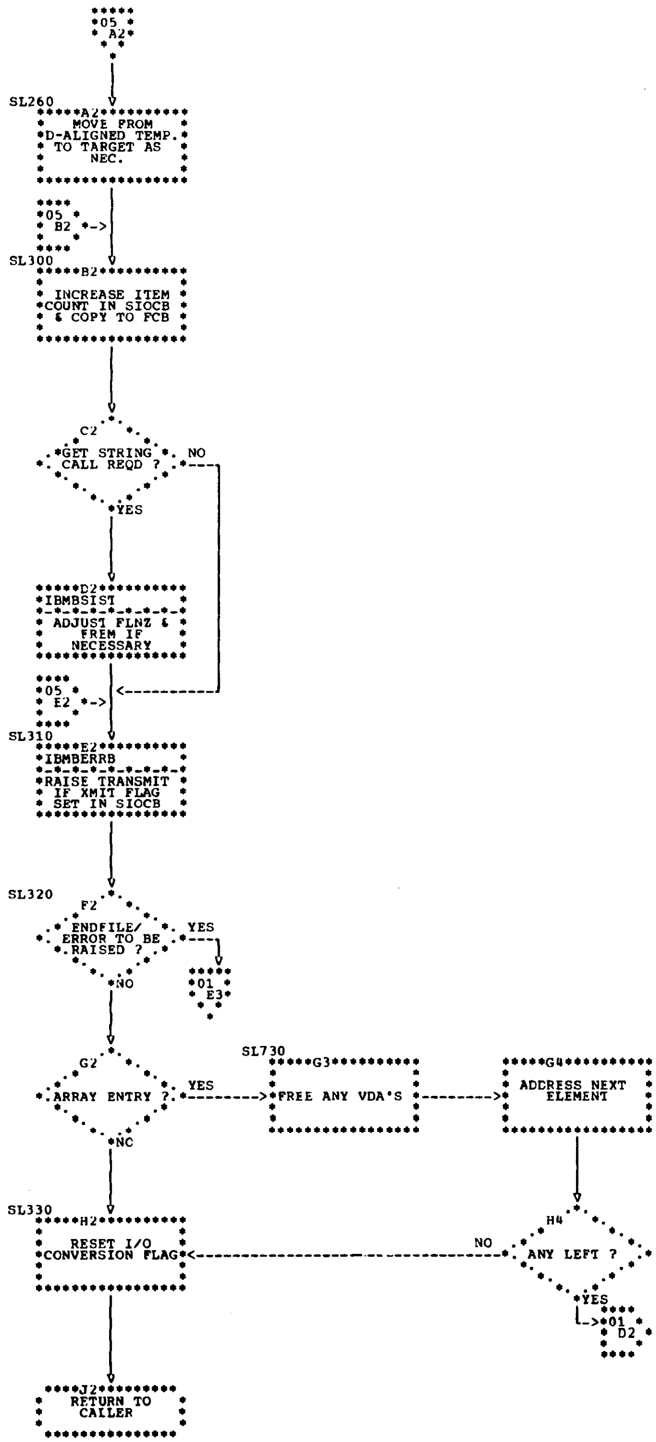


Chart DSLJ. List-directed Input, restricted conversions (part 5 of 8)

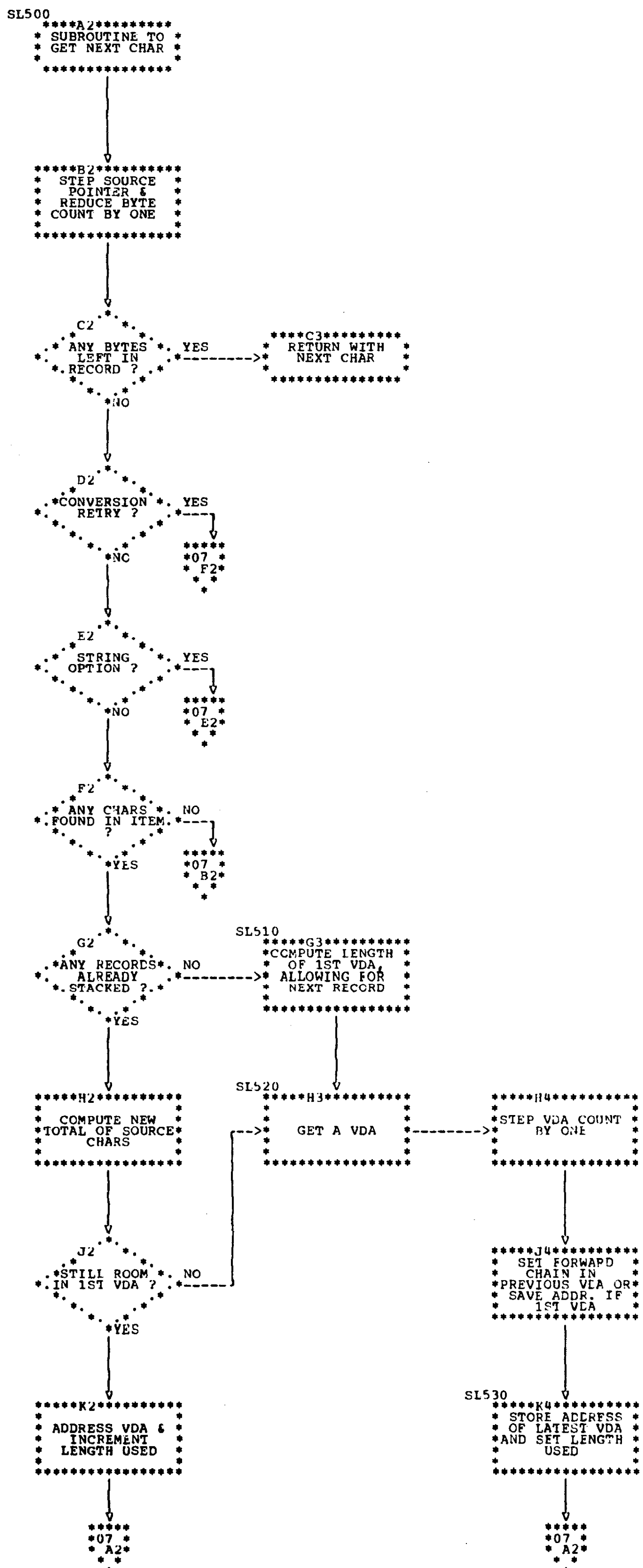


Chart DSLJ. List-directed Input, restricted conversions (part 6 of 8)

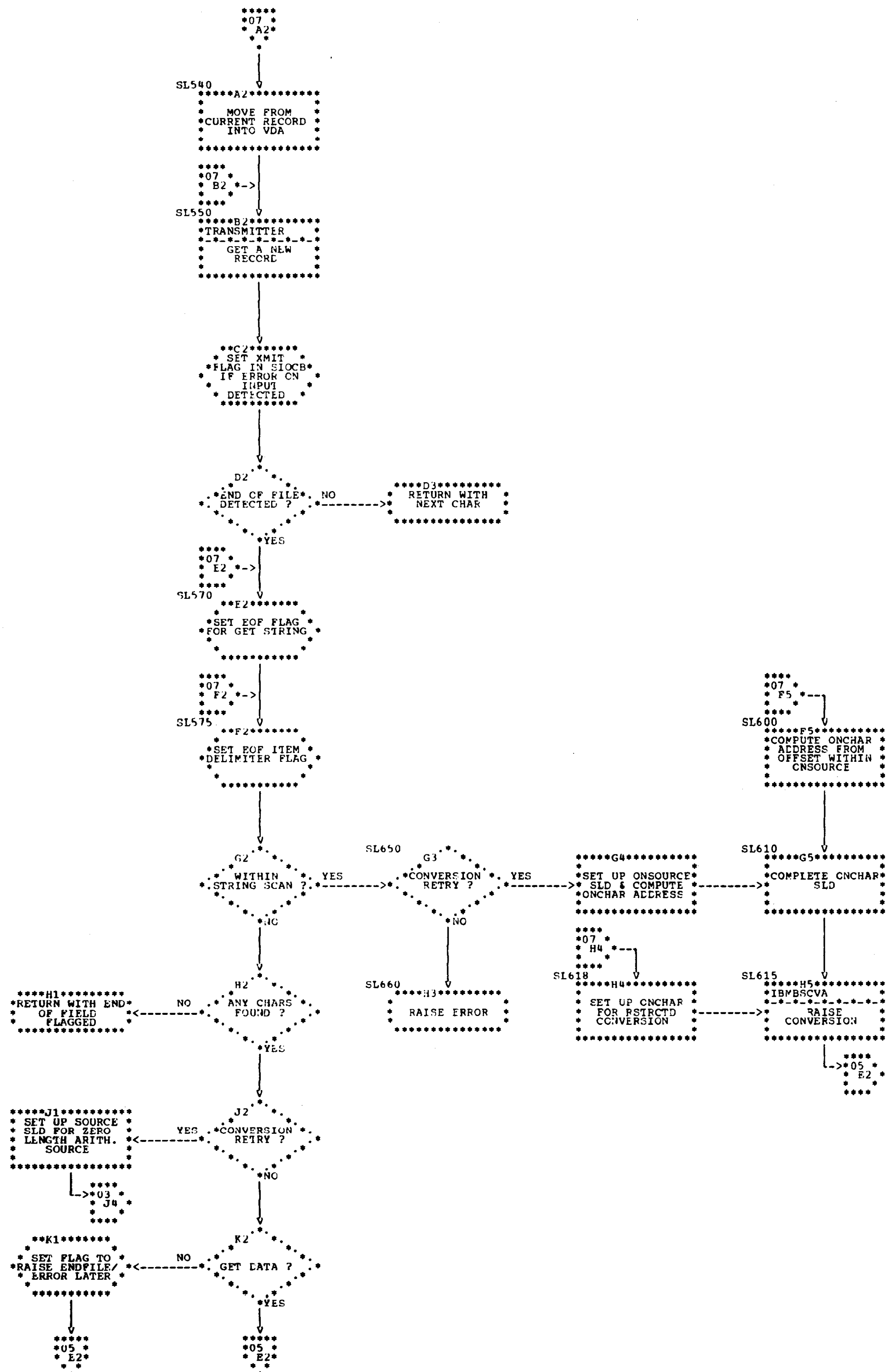


Chart DSLJ. List-directed Input, restricted conversions (part 7 of 8)

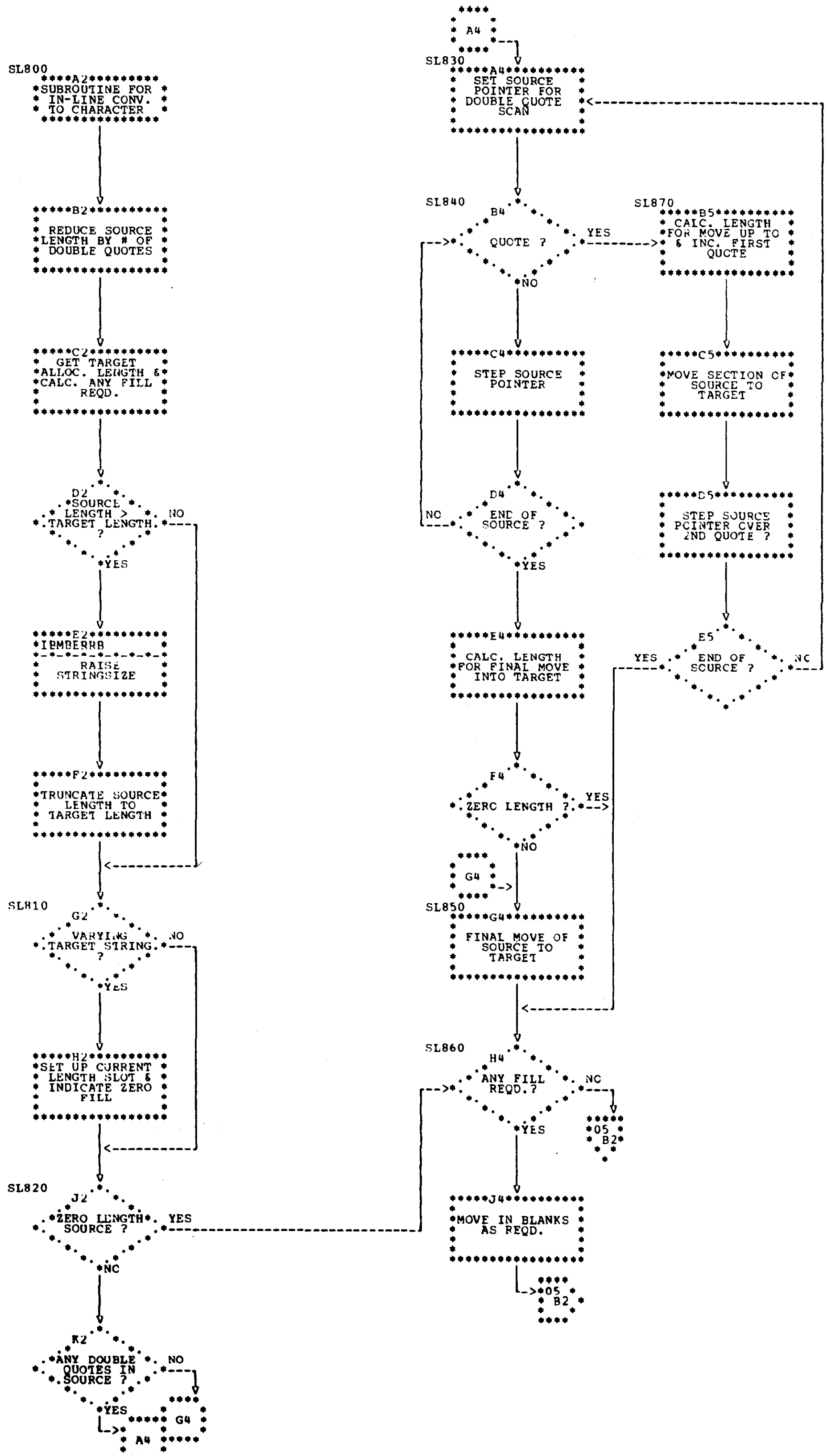


Chart DSLJ. List-directed Input, restricted conversions (part 8 of 8)

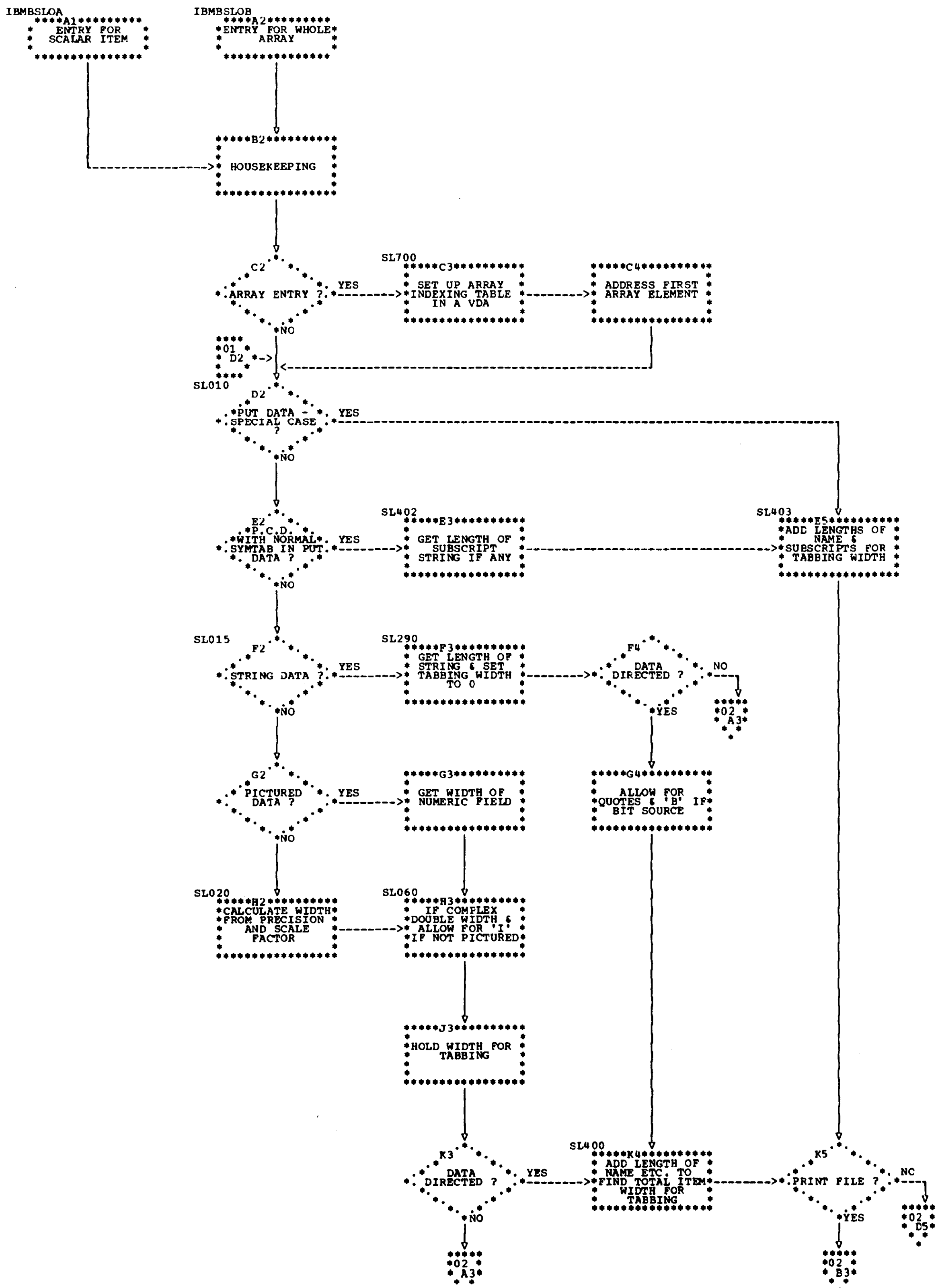


Chart DSLO. List-directed Output (part 1 of 3)

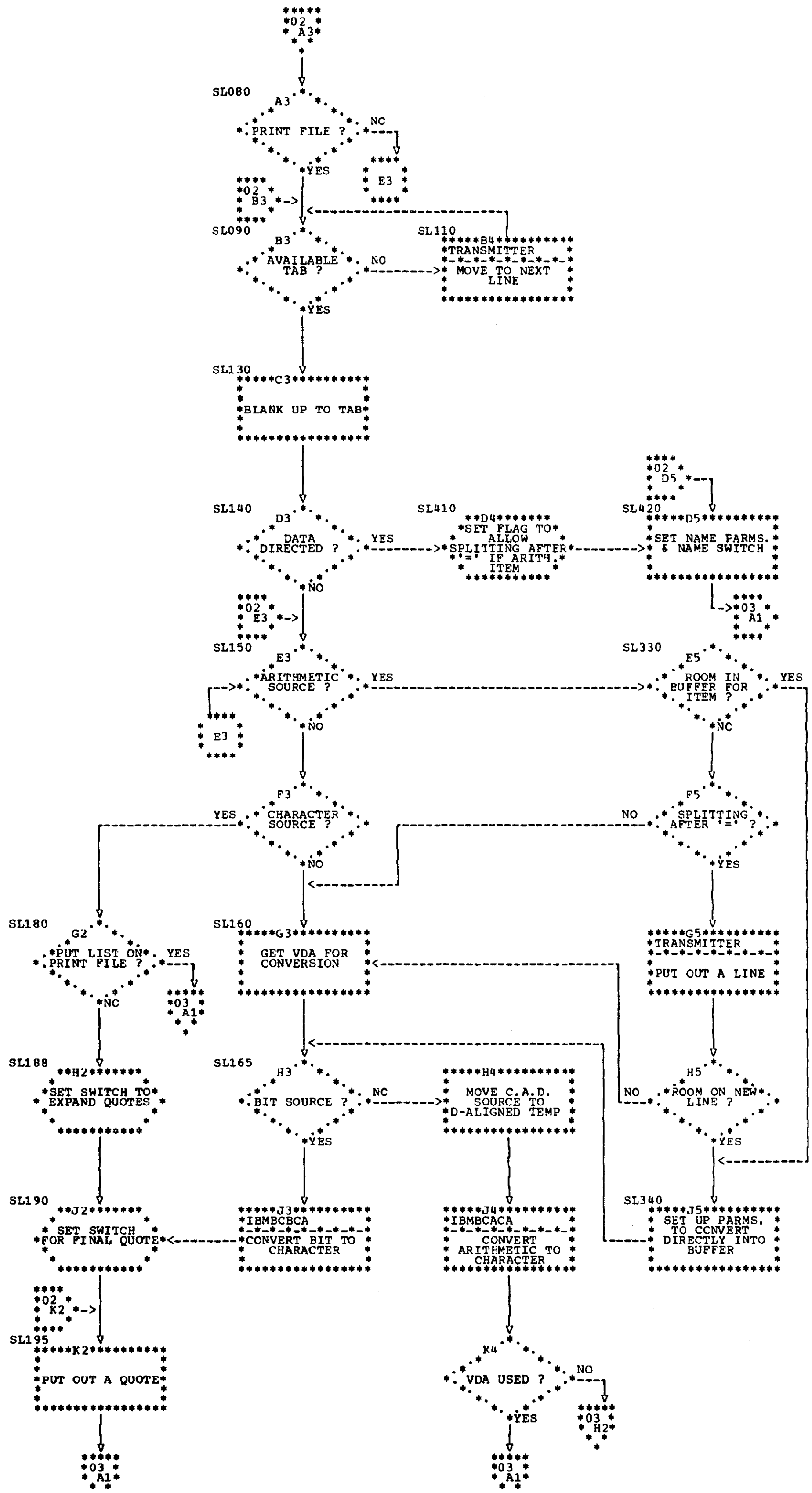


Chart DSLO. List-directed Output (part 2 of 3)

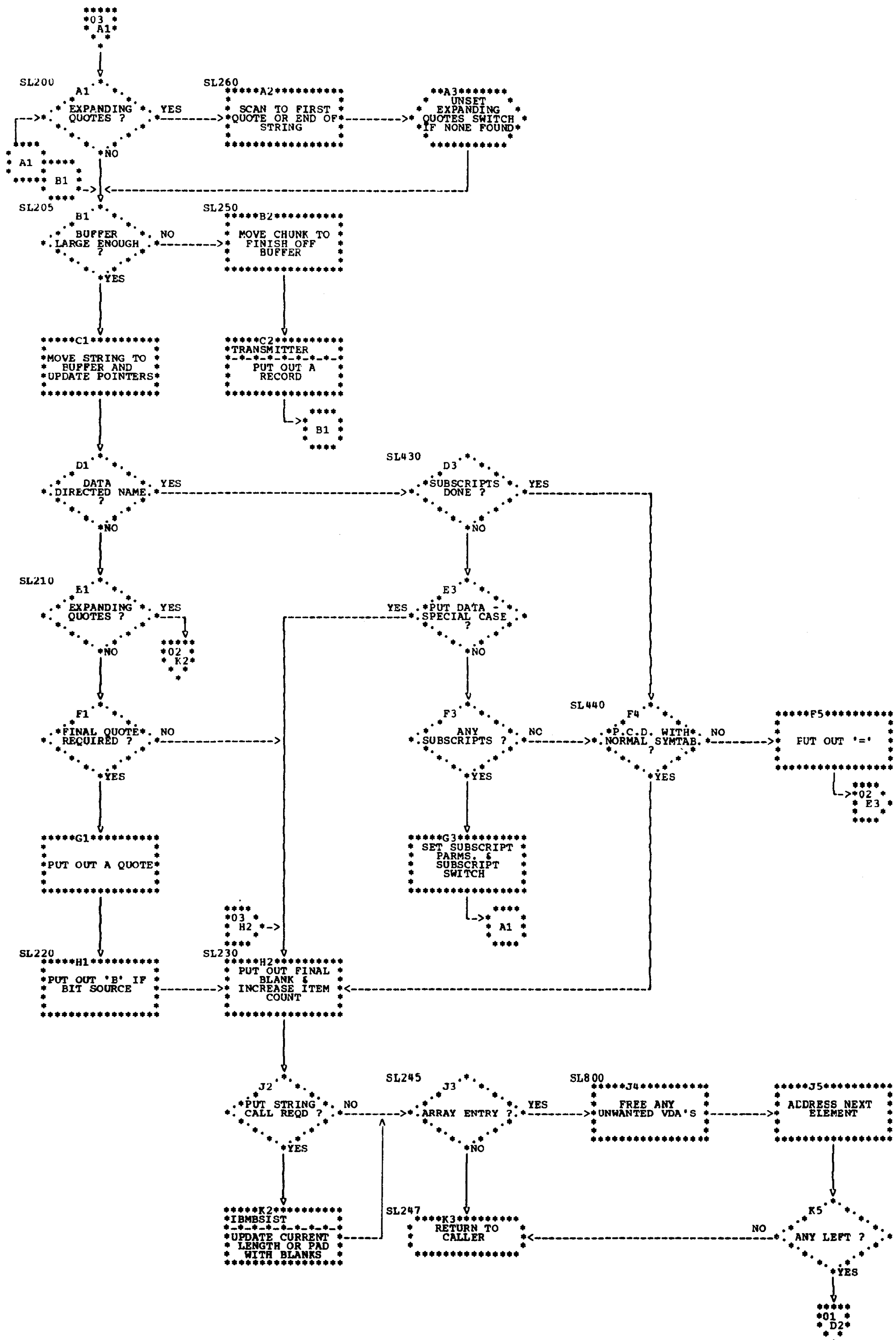


Chart DSLO. List-directed Output (part 3 of 3)

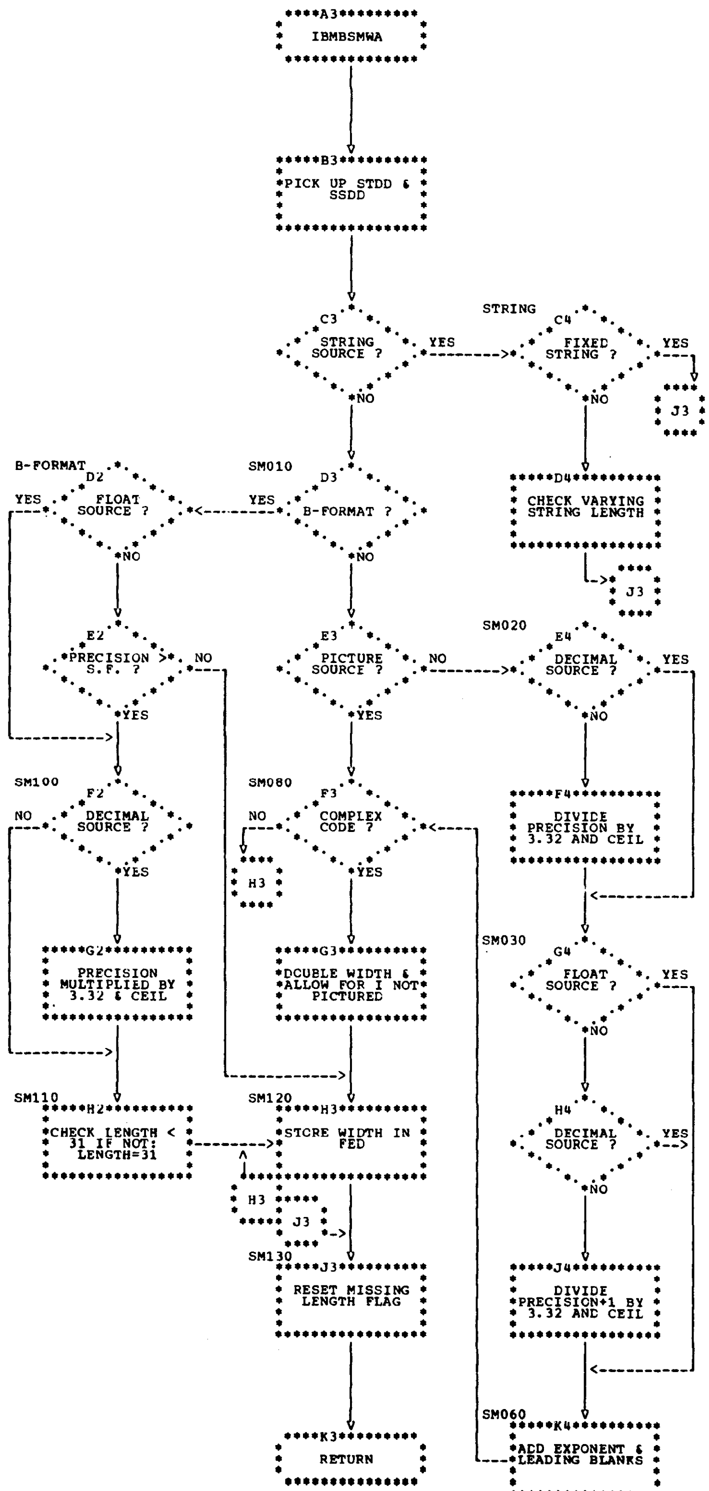


Chart BSMW. Missing output width module

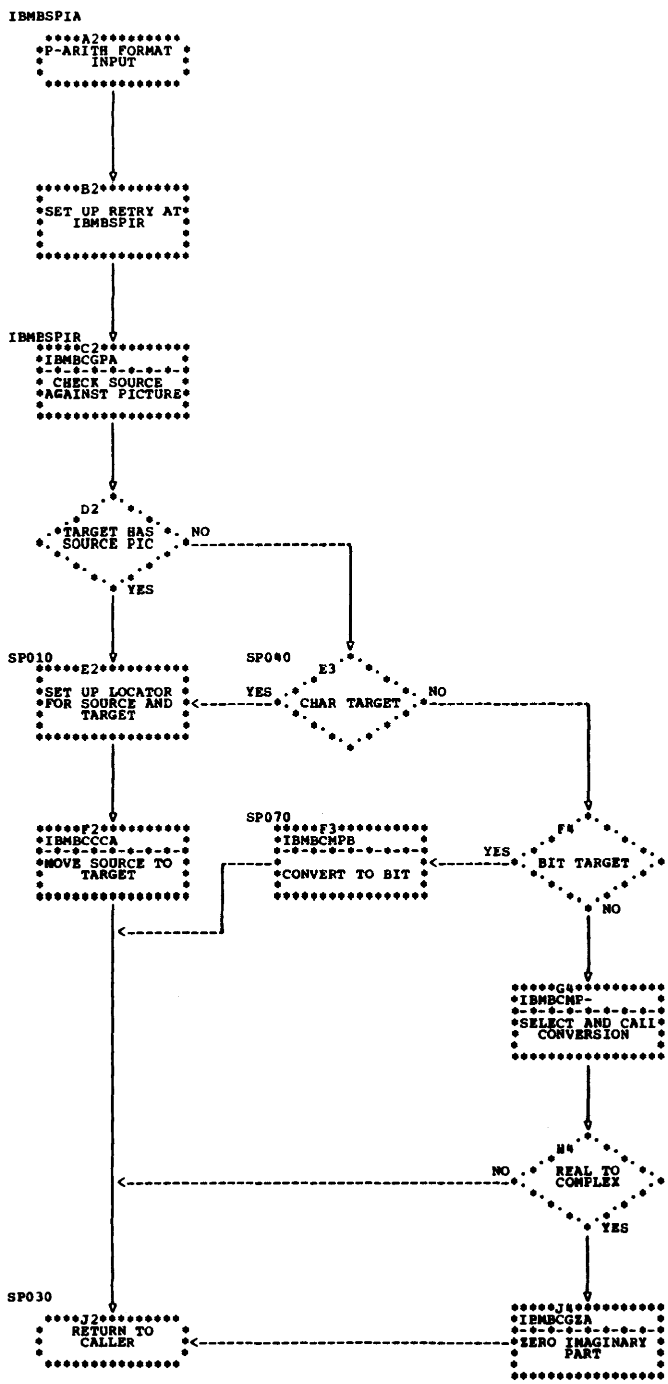


Chart BSPI. Input conversion director (P format)

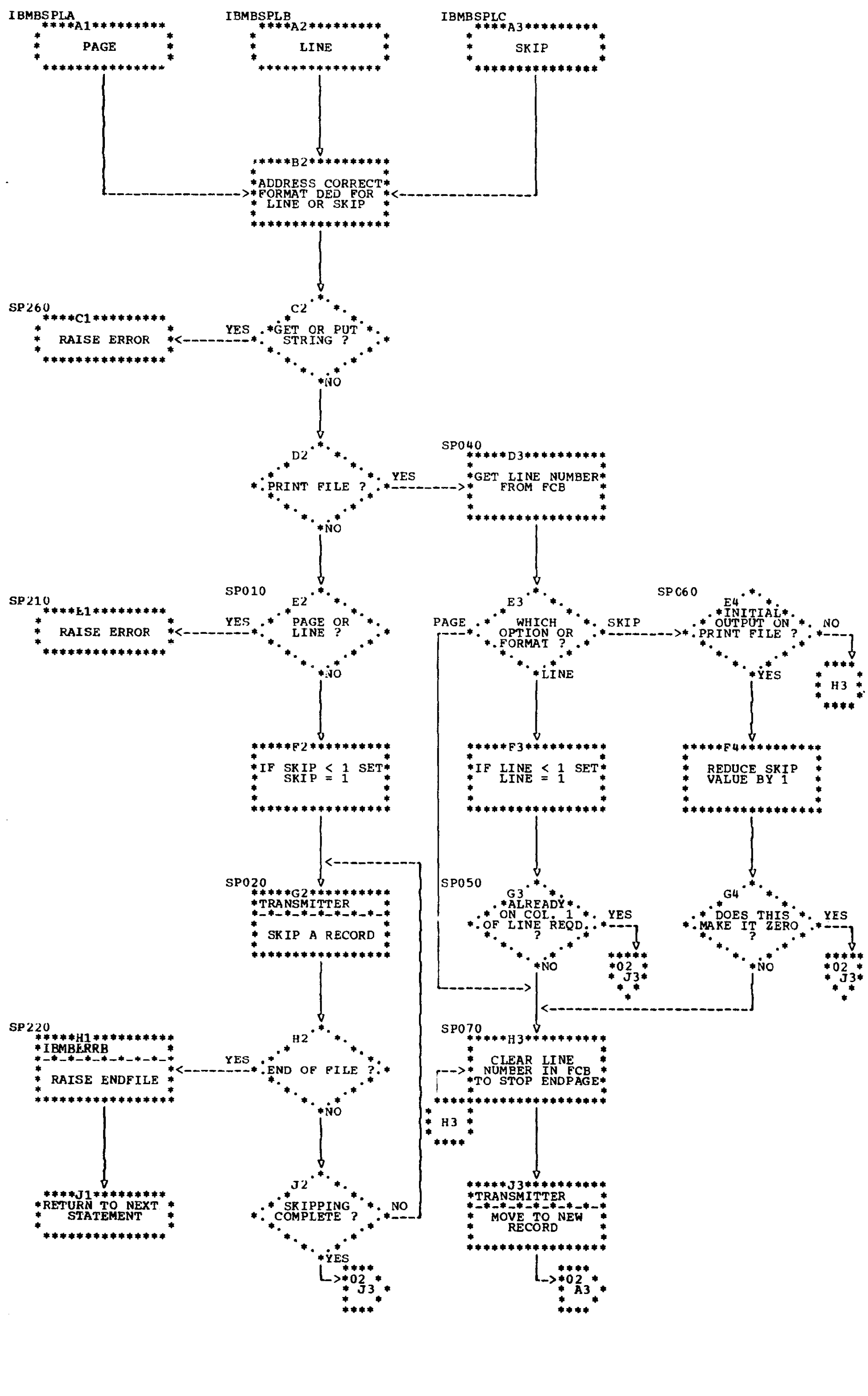


Chart DSPL. PAGE, LINE, and SKIP (part 1 of 2)

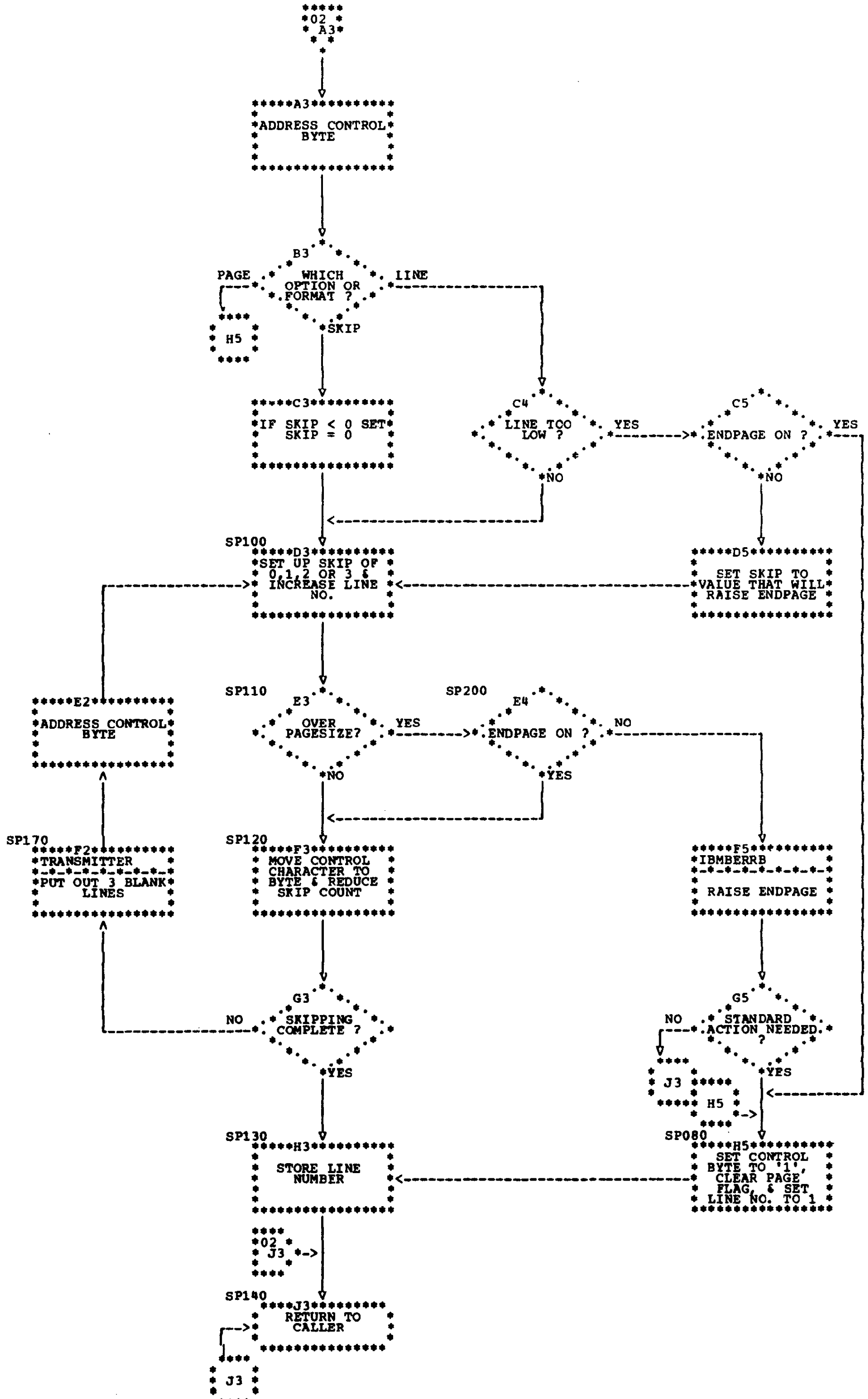


Chart DSPL. PAGE, LINE, and SKIP (part 2 of 2)

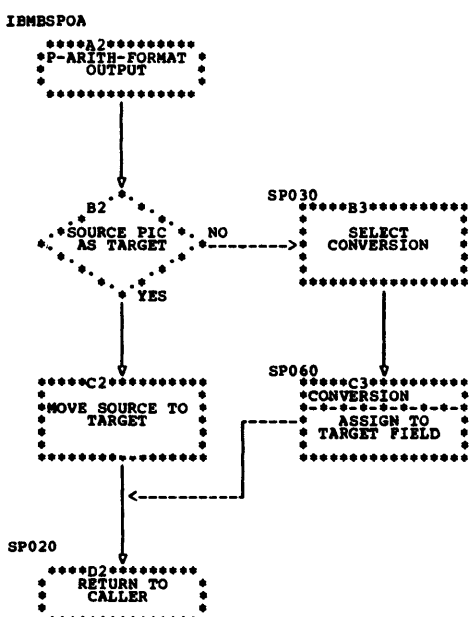


Chart BSPO. Output conversion director (P format)

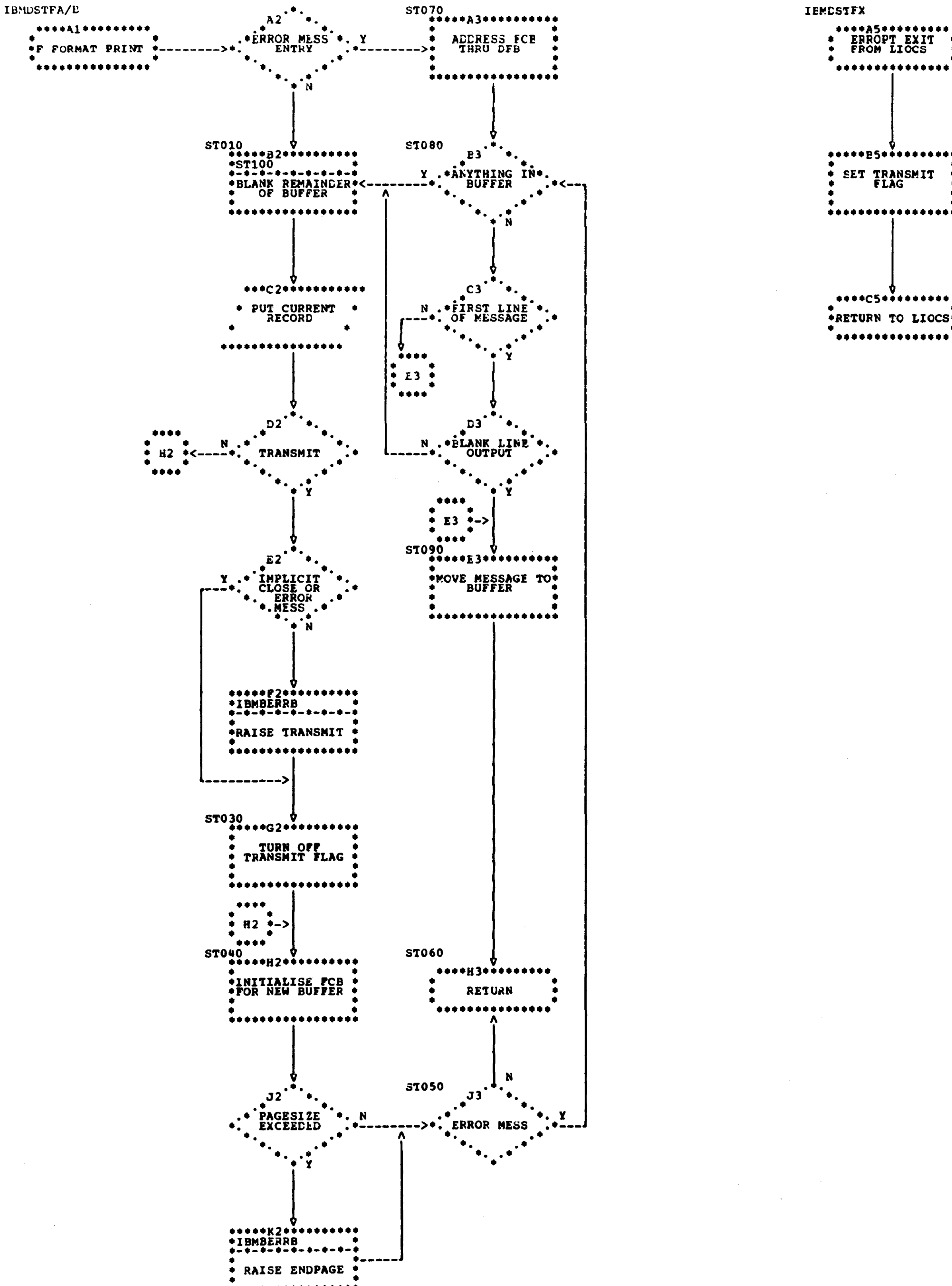


Chart DSTF. Stream input transmitter

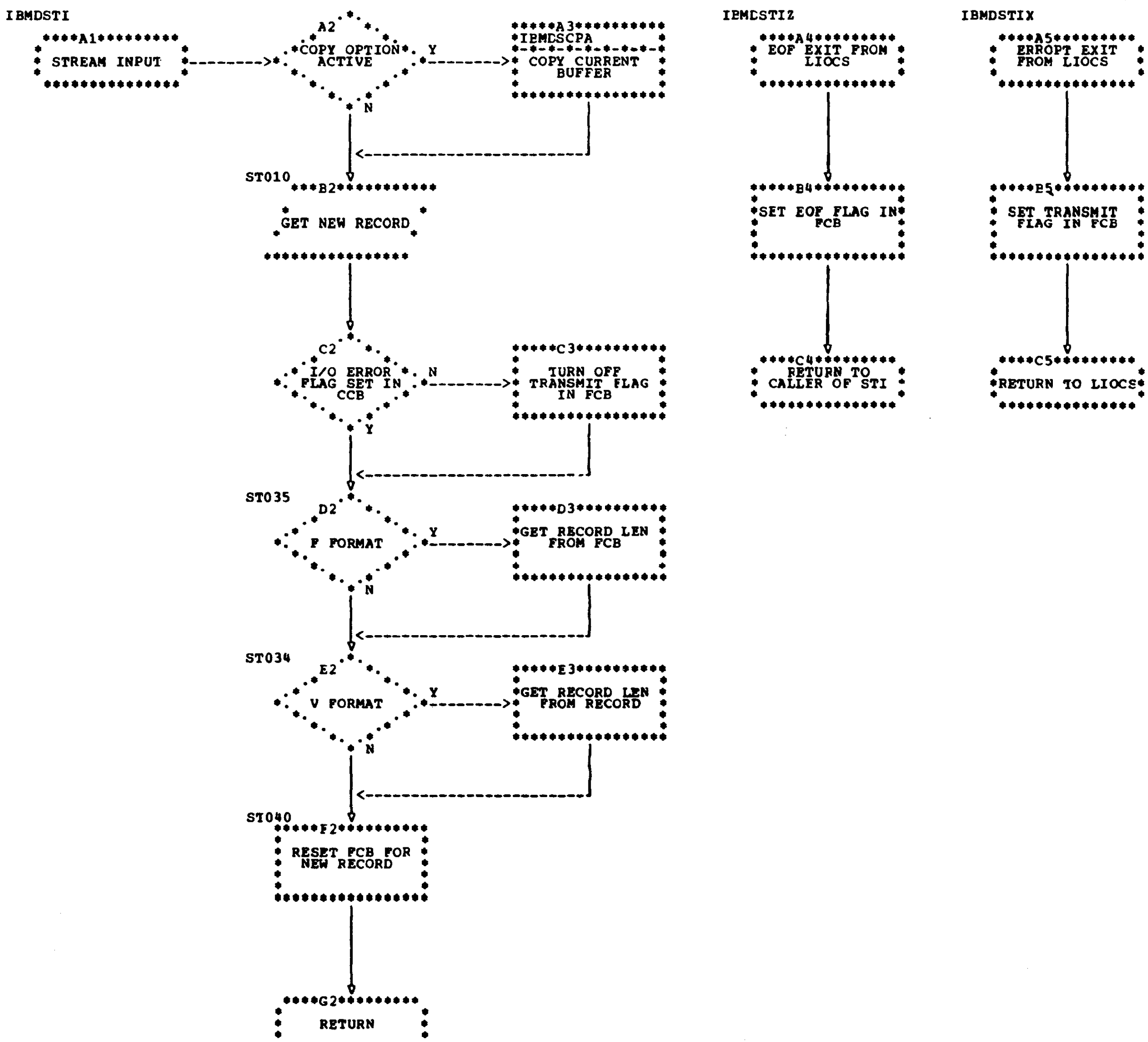


Chart DSTI. Stream F-format print transmitter

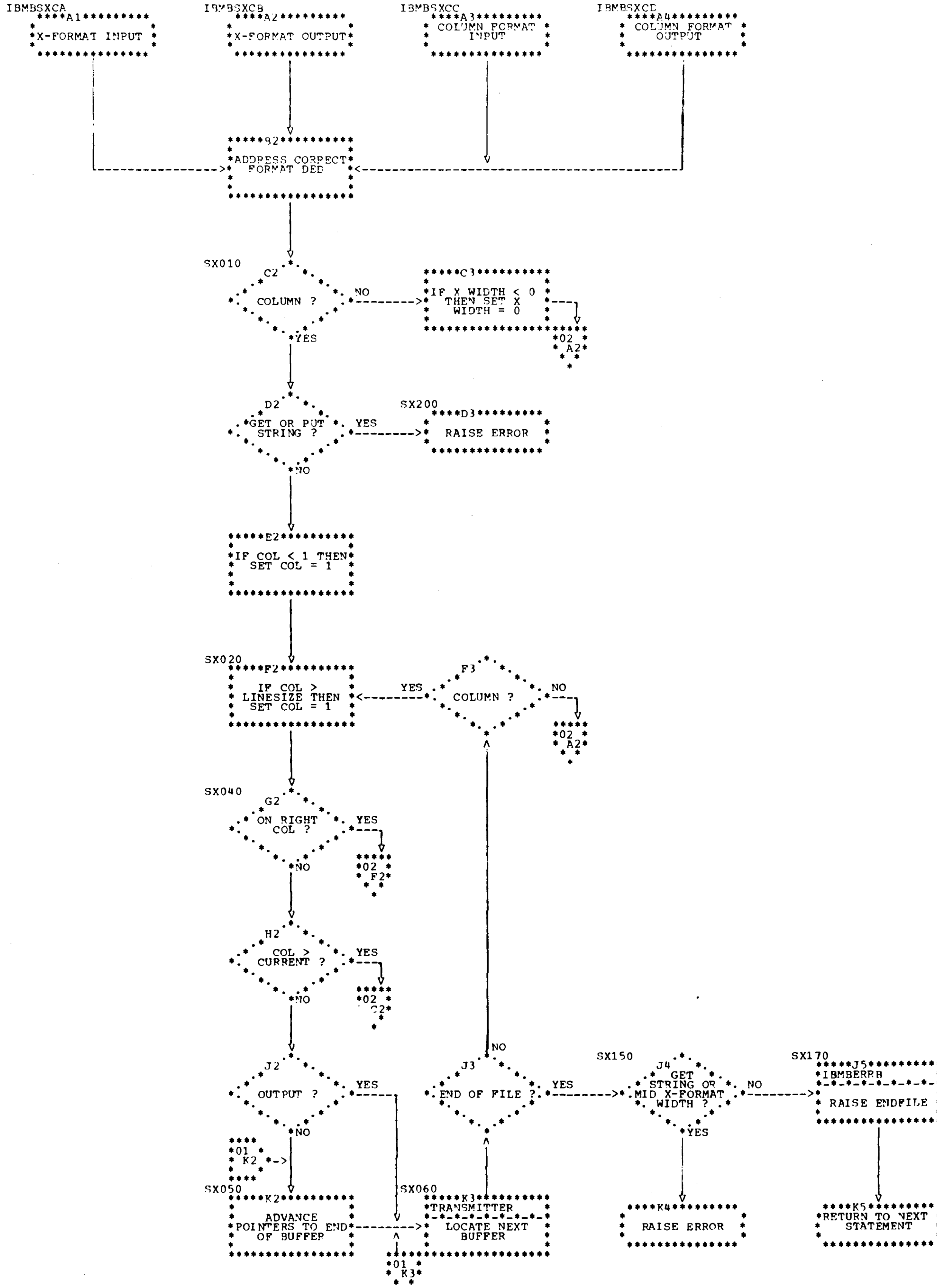


Chart DSXC. X and COLUMN format items (part 1 of 2)

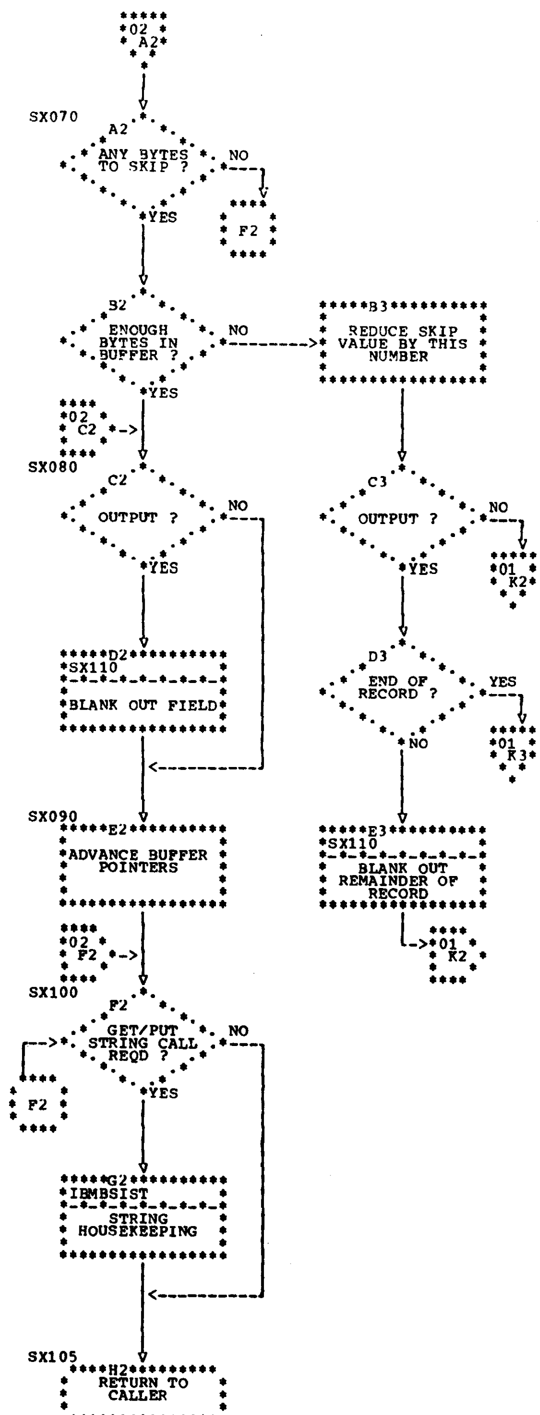


Chart DSXC. X and COLUMN format items (part 2 of 2)

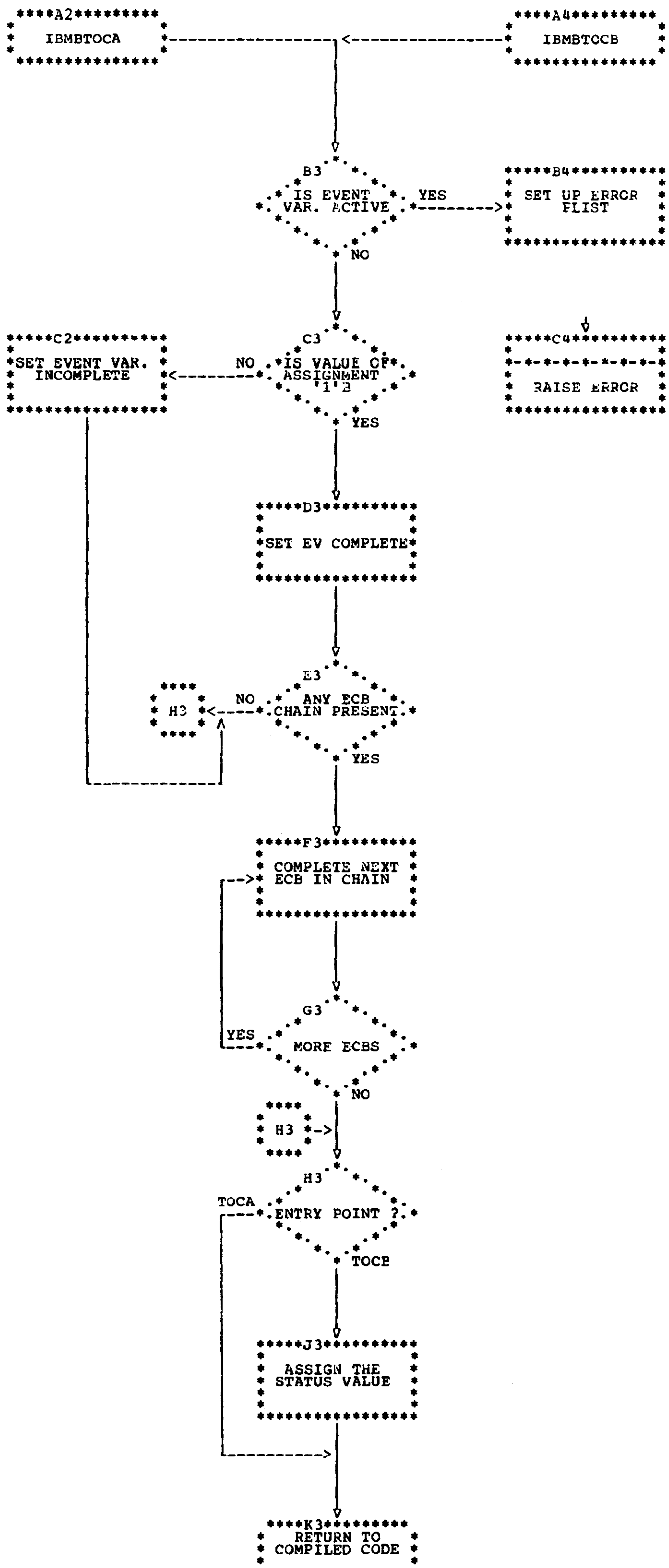
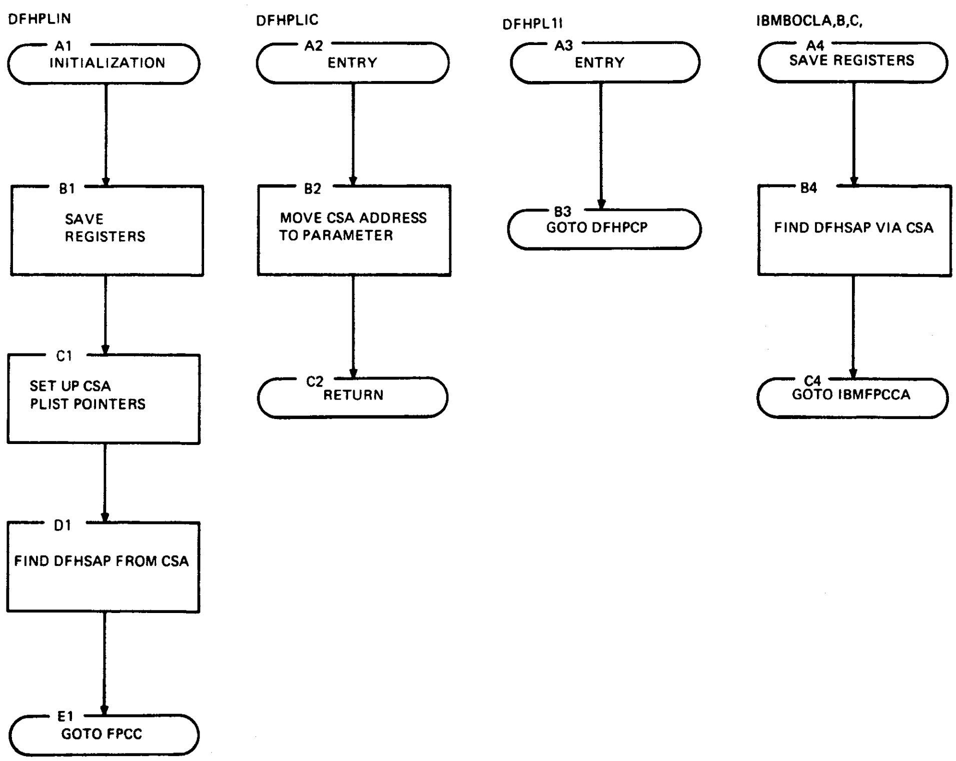


Chart BTOC. COMPLETION pseudovariable



|Chart DFHPL1I. CICS initialization bootstrap

Index

IBM **Technical Newsletter**

This Newsletter No. LN33-6179
Date October, 1976

Base Publication No. LY33-6011-1
System S360/S370-29

Previous Newsletters LN33-6118

DOS
PL/I Resident Library:
Program Logic

This Technical Newsletter, a part of Version 1, Release 5, Modification 0 of the IBM DOS PL/I Resident 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	185,186
3,4	191–194
4.1 (added)	203,204
5–8	211–214
15,16	247,248
16.1 (added)	265,266
37,38	321,322
51,52	351–358
52.1 (added)	Reader's comment form

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 resident 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.

IBM **Technical Newsletter**

This Newsletter No.	LN33-6118
Date	June, 1974
Base Publication No.	LY33-6011-1
File No.	S360/S370-29
Previous Newsletters	None

DOS
PL/I Resident Library:
Program Logic

This Technical Newsletter, an update to Version 1, Release 4, Modification 1 of the DOS PL/I Optimizing Compiler 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
3–6
23, 24
37–40
211, 212
355–358

A change to the text is indicated by a vertical line to the left of the change.

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 Resident Library:
Program Logic

Order No. LY33-6011-1

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

Cut or Fold Along Line

What is your occupation? _

Number of latest Technical Newsletter (if any) concerning this publication: _ _ _ _ _ _ _ _ _ _ _ _

Please indicate in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

LY33-6011-1

Your comments, please . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Cut or Fold Along Line

Fold Fold

First Class
Permit 40
Armonk
New York

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

Postage will be paid by:

International Business Machines Corporation
Department 813(HP)
1133 Westchester Avenue
White Plains, New York 10604

Fold Fold

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)

DOS PL/I Resident Library: Program Logic File No. S360/S370-29 Printed in U.S.A. LY33-6011-1