LY28-6424-02

Program Product

IBM DOS/VS COBOL Subroutine Library Program Logic

Program Number: 5746-LM4

IBM

PREFACE

This publication describes the object-time subroutine library used by the IBM DOS/VS COBOL compiler. It is intended for use by persons involved in library maintenance and by system programmers involved in altering the library for installations requiring such alteration. This publication supplements the subroutine listings and their comments, but it is not a substitute for them. The publication is divided into the following parts:

- An introduction which describes the contents and the functions of the library and specifies the relationships between the library and the compiler and the library and the operating system.
- A methods of operation section which describes the function of each subroutine in the library, the code used in the object program to interface with each subroutine, and the output (where applicable) of each subroutine. This section is divided into two main parts: the subroutines for object-time program operations; the subroutines for object-time debugging operations; and the subroutines for object-time execution statistics.
- A program organization section which consists of diagrams and flowcharts. The diagrams describe the flow of control, loading and calling dependencies, and virtual storage layouts in instances where several programs are present together in virtual storage. Flowcharts are provided for most of the data management subroutines, all of the subroutines for object-time debugging operations, and for other complex subroutines.
- A data areas section which describes the tables used by the subroutines for object-time debugging operations and control blocks for VSAM subroutines.
- A diagnostic aids section which includes execution-time messages and error messages from the debugging subroutines, virtual storage layouts, information on locating DTF's and data, and special diagnostic aids for debugging subroutines.
- A glossary of special terms.

Third Edition (September 1985)

This is a reprint of LY28-6424-01 incorporating changes released in the following Technical Newsletters:

LN20-9122-00 (dated 1 August 1975)
LN20-9183-00 (dated 3 December 1976)
LN20-9237-00 (dated 5 August 1977)
LN20-9348-00 (dated 15 May 1981)

Effective use of this manual requires an extensive knowledge of the IBM Assembler Language, DOS/VS System Control, DOS/VS COBOL and the IBM DOS/VS COBOL Compiler. Prerequisite and related publications include:

IBM DOS/VS Operating Procedures, Order No. GC33-5378

IBM OS/VS and DOS/VS Assembler Language Guide, Order No. GC33-4010

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

IBM DOS/VS System Utilities, Order No. GC33-5381

IBM DOS/VS Supervisor and I/O Macros, Order No. GC33-5373

IBM DOS/VS Access Method Services, Order No. GC33-5382

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

Prerequisite Program Product documents include:

IBM DOS Full American National Standard COBOL, Order No. GC28-6394

IBM DOS/VS COBOL Compiler and Library Programmer's Guide, Order No. SC28-6478

IBM DOS/VS COBOL Compiler and Library Installation Reference Material, Order No. SC28-6479

IBM DOS/VS COBOL Compiler Program Logic, Order No. LY28-6423

The following publications provide detailed information on the IBM 3886 Optical Character Reader:

IBM 3886 Optical Character Reader General Information Manual, Order No. GA21-9146

IBM 3886 Optical Character Reader Input Document Design and Specifications, Order No. GA21-9148

DOS/VS Planning Guide for the IBM 3886 Optical Character Reader, Model 1, Order No. GC21-5059

The following publications provide information on the IBM DOS/VS Sort/Merge Program Product, Program Number 5746-SM1, and the DOS Sort/Merge Program Product, Program Number 5743-SM1:

IBM DOS/VS Sort/Merge General Information, Order No. GC33-4030

IBM DOS/VS Sort/Merge Installation Reference Material, Order No. SC33-4026

IBM DOS Sort/Merge Programmer's Guide, Order No. SC33-4018

The titles and abstracts of related publications are listed in IBM System/360 and System/370 Bibliography, Order No. GA22-6822.

ACKNOWLEDGMENT

The following extract from Government Printing Office Form Number 1965-0795689 is presented for the information and guidance of the user:

"Any organization interested in reproducing the COBOL report, and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention 'COBOL' in acknowledgement of the source, but need not quote this entire section.

"COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

"No warranty, expressed or implied, is made by any contributor or by the COBOL committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

"Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

"The authors and copyright holders of the copyrighted material used herein

> FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the UNIVAC (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator, Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications."

Summary of Amendments — Number 3

Date of Publication: May 15, 1981

Form of Publication: TNL LN20-9348 to LY28-6424-1

New: Program and Documentation

The library has been expanded with the following new subroutines:

ILBDCVB	Convert to Binary
ILBDSTG	STRING
ILBDUST	UNSTRING
ILBDINS	INSPECT
ILBDDTE	DATE, DAY, and TIME
ILBDCMM	GETCORE/FREECORE
ILBDACS	Compare with Alternate Colatting Sequence
ILBDSIO	SAM I/O
ILBDBUG	Use-for-Debugging Declaratives

The new documentation supplies explanatory text for these new routines, and supplies flowcharts where appropriate. In addition, these existing subroutines have been modified: ILBDGDO, ILBDSAE, ILBDMNS, ILBDVOC, ILBDVIO, ILBDSRT, ILBDMRG, ILBDMVE, ILBDSPA, ILBDDBG, and ILBDMP24.

Summary of Amendments — Number 2

Date of Publication: December 3, 1976

Form of Publication: TNL LN20-9183 to LY28-6424-1

IBM DOS/VS COBOL

Maintenance: Documentation

- Minor technical changes and additions have been made to the text.

Summary of Amendments — Number 1

Date of Publication: March 15, 1974

Form of Publication: Revision, LY28-6424-1

Support of New CBL Statement Option

New: Programming Feature

Release 2 of the IBM DOS/VS COBOL Subroutine Library supports the object-time execution statistics option COUNT/NOCOUNT.

Support of SORT-OPTION IS data-name in SD Statement

New: Programming Feature

Release 2 of the DOS/VS COBOL Subroutine Library supports the SORT–OPTION IS data-name clause. This allows the programmer more flexibility in handling sort files and use of SORT/MERGE program messages.

ACCEPT Verb

New: Programming Feature

The ACCEPT verb library subroutine now translates lowercase input into the uppercase equivalents.

Debug Common Area

Maintenance: Documentation Only

The debug common area has now been documented as a principal data area used by library subroutines.

CONTENTS

ILLUSTRATIONS

FIGURES

DIAGRAMS

CHARTS

The IBM DOS/VS COBOL Library provides subroutines that can be link edited with object modules produced by the program product IBM DOS/VS COBOL Compiler (Program Number 5746-CB1). The library also provides subroutines that can be dynamically fetched during problem program execution.

LIBRARY CONTENTS

The compiler uses a number of subroutines to perform frequently required operations. Because these subroutines are too extensive to be efficiently placed into the object module whenever needed, they are stored in the relocatable library and included in the phase by the linkage editor. Exceptions to this are transient subroutines $$BCOBER, $$BCOBR1, $$BFCMUL, $$BCOBEM, and the SYMDMP subroutines, which are stored in the core image library.

The COBOL Object-time Library contains subroutines to perform the following operations:

- Internal data format conversion.
- Arithmetic operations.
- Input/Output operations.
- Miscellaneous operations to support such statements as SEARCH or DISPLAY and specialized operations such as class tests or compares.
- Internal data format conversions for input and output files coded in the American National Standard Code for Information Interchange, X3.4-1968.
- Generation of a formatted trace of the last procedures executed before an abnormal termination of a job in response to the specification of the flow trace option. The number of procedures to be traced is specified by the user.
- Identification of the statement being executed at the time of an abnormal termination of a job in response to the specification of the statement number option. The information includes the name of the program containing the statement and the number of the statement and of the verb being executed at the time of abnormal termination.
- Generation of additional execution-time information for debugging purposes in response to the specification of the symbolic dump option. This information includes symbolic formatted dumps of named data areas taken dynamically at specified points in the Procedure Division, and a symbolic formatted dump when a program terminates abnormally. A dump taken at abnormal termination consists of three parts: an abnormal termination message identifying the source statement causing the error, selected areas in the Task Global Table, and data items from the Data Division. Note that a dynamic dump, requested when a STOP RUN or GOBACK statement is encountered, produces, in effect, an "end-of-job" dump.
- Generation of object-time execution statistics for debugging, testing, and optimization in response to the COUNT option. The statistics include a listing of the Procedure Division verbs with execution frequency information and an executable verb summary. The statistics are provided at normal and abnormal termination.

ENVIRONMENTAL AND PHYSICAL CHARACTERISTICS

The DOS/VS COBOL Subroutine Library is designed for use under the IBM DOS/VS Operating System with object modules produced by the DOS/VS COBOL Compiler. A DOS Release 29 is the minimum level required.

The DOS/VS COBOL Subroutine Library is part of the DOS/VS core image and relocatable libraries, which must reside on a disk storage device.

If the SYMDMP option is specified, the library subroutine called to supply the symbolic formatted dump requires that the dictionary of symbolic names and other information produced during compilation be present at execution time. This information is written on an additional work file designated as SYS005 during compilation. SYS005 may reside on either a tape or direct access device. The work file may be named according to the user's option at execution time.

OPERATIONAL CONSIDERATIONS

Phases 50, 51, and Phase 60 or 64 of the DOS/VS COBOL Compiler generate the calls to the subroutines contained in the COBOL Object-time Library. (Note that Phase 60 or 64 generates these calls in the initialization routines in the object module.) Parameters are passed to the subroutines in one of the following ways:

- In general or floating-point registers.
- As in-line constants (DCs) following the call.
- In the WORKING CELL area of the Task Global Table (TGT) in the object module.

The subroutines can return parameters in registers or in the WORKING CELL area.

Note: References to the WORKING CELL area are in the form of a displacement from register 13 which points at execution time to the beginning of the Task Global Table. In the calling sequences in Section 2: "Method of Operation," the references are in the form:

```
WORKA(length,13)
```

METHODS OF OPERATION

The subroutines described below perform frequently required program operations at object time. These operations include internal data format conversions, arithmetic operations, input/output operations, miscellaneous operations to support such statements as SEARCH or DISPLAY and specialized operations such as class tests or compares, and certain operations connected with the ASCII support feature of the compiler.

Flowcharts are provided in "Section 2: Program Organization" for some of the subroutines. Each chart identifier appears in square brackets after the name of its subroutine.

ARITHMETIC CONVERSION SUBROUTINES

The subroutines described below perform the arithmetic conversions between the eight numeric data formats permitted in COBOL. The conversions from internal decimal to external decimal, from external decimal to internal decimal, and from internal decimal to report are done in-line and do not require use of the library.

The following conventions are used for the conversion subroutine parameters:

BINARY: Single words are in register 0; double words are in registers 0 and 1.

INTERNAL DECIMAL: The number is passed in the first 10 bytes of the WORKING CELL area in the Task Global Table (TGT). It is right justified with high-order zeros.

EXTERNAL DECIMAL: The number is passed in the first 18 bytes of the WORKING CELL area in the TGT. It is right justified with high-order zeros.

INTERNAL FLOATING-POINT: The number is long form in floating-point register 0.

EXTERNAL FLOATING-POINT: The number is variable in length. For input to the subroutine, it is pointed to by general register 3. For output from the subroutine, it is in the WORKING CELL area in the TGT.

STERLING NONREPORT: Sterling nonreport items are either internal decimal for computational purposes (right justified in a 16-byte field) or external decimal for display purposes (variable length, from 4 to 20 bytes).

STERLING REPORT: Sterling report items are internal decimal for computational purposes. They are right justified in a 16-byte field.

Note: The external floating-point (EF) number parameter code bits are:

Bit	Meaning, if on
1-5	Not used
6	Mantissa PICTURE sign is negative
7	Exponent PICTURE sign is negative
8	EF number has a real decimal point

Binary to Internal Decimal (ILBDBID0)

Operation: Converts a double precision binary number into a 10-byte internal decimal number. The binary number must be in register pair 0, 1 or 2, 3 or 4, 5.

Linkage:

```
L     15,=V(entry point)
PALR  14,15
```

Note: Substitute for entry point as follows:

ILBDBID0 if binary number is in register pair 0, 1
ILBDBID1 if binary number is in register pair 2, 3
ILBDBID2 if binary number is in register pair 4, 5

Output: A 10-byte internal decimal number starting at WORKA(13), where 13 is the register pointing to the TGT.

Binary to External Decimal (ILBDBIE0)

Operation: Converts a double precision binary number into an 18-byte external decimal number. The binary number must be in register pair 0, 1 or 2, 3 or 4, 5.

Linkage:

```
L     15,=V(entry point)
PALR  14,15
```

Note: Substitute for entry point as follows:

ILBDBIE0 if binary number is in register pair 0, 1
ILBDBIE1 if binary number is in register pair 2, 3
ILBDBIE2 if binary number is in register pair 4, 5

Output: An 18-byte external decimal number starting at WORKA(13), where 13 is the register pointing to the TGT.

Binary to Internal Floating-Point (ILBDBII0)

Operation: Converts a double precision binary number into a double precision floating-point number.

Linkage:

```
LM    0,1,BI-number
L     15,=V(ILBDBII0)
BALR  14,15
DC    XL2'Decimals in BI number'
```

Output: A double precision floating-point number in floating-point register 0.

Internal and External Decimal to Internal Floating-Point (ILBDDCI0)

Operation: Converts a 16-byte internal decimal number or an 18-byte external decimal number into a double precision internal floating-point number. Register 13 points to the TGT.

Linkage:

For internal decimal:

```
ZAP   WORKA(16,13),ID-field
L     15,=V(ILBDDCI1)
BALR  14,15
DC    XL2'Decimals in ID number'
```

For external decimal:

```
MVC   WORKA(18,13),ED-field
L     15,=V(ILBDDCI0)
BALR  14,15
DC    XL2'Decimals in ED number'
```

Output: A double precision internal floating-point number in floating-point register 0.

Internal Floating-Point to Binary (ILBDIFB0)

Operation: Converts a double precision internal floating-point number into either a binary number, or into a binary number and an exponent to the base 10, depending on where the subroutine is called from. The internal floating-point number is put into floating-point register 0. If the internal floating-point number is too big, the binary number is set to the maximum. If the internal floating-point number is too small, the binary number is set to the minimum. No error indication is given.

Linkage:

```
LD    0,FP-number
   or
SDR   0,0
LE    0,FP-number
```

Followed in either case by:

```
L     15,=V(ILBDIFB1)
CNOP  6,8
BALR  14,15
DC    XL8'double precision floating-
        point number' (of the form
        10**X where X is the number of
        decimals in the result field)
```

Output: A binary number in register pair 0,1.

Note: If this subroutine is called by another subroutine, the linkage and output are as follows:

If called by ILBDIFD0:

Linkage:

```
LD    0,Internal floating-point number
LD    2,Decimals in result
L     15,=V(ILBDIFB0)
BALR  14,15
```

Output: A binary number in register pair 0,1.

If called by ILBDTEF3:

Linkage:

```
LD    0,Internal floating-point number
LD    6,Digits in external floating-
        point mantissa
L     15,V(ILBDIFB2)
BALR  14,15
```

Output: A binary number in register pair 0,1, and a power-of-10 exponent in register 2.

Internal Floating-Point to Internal Decimal (ILBDIFD0)

Operation: Converts a double precision internal floating-point number into a 10-byte internal decimal number. If the internal floating-point number exceeds the maximum permissible length, register 15 is set to 0 and a normal exit is taken.

Linkage:

```
LD    0,FP-number
   or
SDR   0,0
LE    0,FP-number
```

Followed in either case by:

```
L     15,=V(ILBDIFD0)
CNOP  6,8
BALR  14,15
DC    XL8'FP-number'
        (of the form 10**X where
        X is the number of
        decimals in the result
        field)
```

Output: A 10-byte internal decimal number starting at WORKA(13) where register 13 points to the TGT.

Internal and External Decimal to Binary (ILBDIDB0)

Operation: Converts a 10-byte internal decimal number or an 18-byte external decimal number into a double precision binary number. The decimal field starts at WORKA(13) where register 13 points to the TGT.

Linkage:

```
ZAP   WORKA(10,13),ID-field
L     15,V(entry point)
BALR  14,15
```

Note: Substitute for entry point as follows:

ILBDIDB0, if input is an internal decimal number

ILBDIDB1, if input is an external decimal number

Output: A double precision binary number in register pair 0,1.

Decimal to Binary (ILBDCVB0), Binary to Decimal (ILBDCVB1) [AA]

Operation: The subroutine converts a signed, unsigned, or separate signed external decimal number or a signed or unsigned internal decimal number to binary and converts binary numbers back to external or internal decimal numbers.

When the subroutine receives control at entry point ILBDCVB0, it initializes the PASS1 switch. Two passes must be made by the subroutine if it is necessary to handle two fields when the subroutine is called by the generated code for an UNSTRING verb. If register 2 contains zero, it is assumed, however, that there are not two fields to be processed. The subroutine then checks register 5 for a field address. If register 5 also contains zero, the call to the subroutine has been generated by the code for an UNSTRING verb; in this case, the POINTER and TALLYING fields of an UNSTRING statement are to be initialized to one and zero, respectively. The subroutine passes control to PLACEBIN to perform the initialization.

If either or both registers 2 and 5 contain field addresses, ILBDCVB0 obtains the type flags, field size, and field address and branches to the CNVRTBIN routine. The type flags are used to index a table, called NDXTBL, and obtain the displacement of the code for handling the specific field type. The necessary information for the field, such as the type of sign and where it is located in the field, is set up. Then control is passed to a common set of instructions which move the field to the proper work area for conversion; the field is packed if necessary.

Following this processing, the number is treated as a double-precision number even if it was a single-precision number in the beginning. The number occupies two doubleword work areas: the low-order nine digits (in packed format) are in one work area and the high-order nine digits are in the other. These digits are converted to the binary in two registers. The value of the high-order register is multiplied by 10^9 to reflect its actual value, and the sign is adjusted to negative if necessary.

Control is then passed to PLACEBIN. If the binary values are to be returned in registers, PLACEBIN merely returns control to the caller. However, if the call to ILBDCVB0 was generated by the code for an UNSTRING verb, the binary values are placed in a work area for later use by ILBDUST0, and control is

returned to the caller. The address of the work area is contained in the SCUSTWRK field in ILBDMNS0. If the SCUSTWRK field contains zero, no storage has been obtained yet for the work area. In this case, ILBDCVB0 issues a GETVIS macro instruction to obtain storage for the work area and enters the address of the area obtained in the SCUSTWRK field. The binary values are then placed in the correct location in the work area and control is returned to the caller.

When the subroutine receives control at entry point ILBDCVB1, it determines whether the number to be converted is already in registers 1 and/or 2. If it is not, it obtains the number from the appropriate location in the USTWRK work area. Then, ILBDCVB1 sets the type flags and branches to the CNVRTDEC routine.

CNVRTDEC converts the values in registers 1 and 2 to packed decimal format. The high-order nine digits from register 1 occupy one doubleword work area and the low-order nine digits from register 2 occupy another. If the receiving field for the converted value is internal decimal, the two doubleword areas are moved to form one 18-digit number. If the receiving field is external decimal, the doubleword work areas are unpacked to form one 18-digit external decimal number and zone bits are adjusted. Next, the type flags for the field are used to index the NDXTBL table and obtain the displacement of the code for handling the sign of the number. When sign processing is completed, the converted number in the work area is moved into a field, the address of which was passed to ILBDCVB1 in register 5. Finally control is returned to the caller.

ILBDCVB0

Linkage generated for an UNSTRING verb:

```
LA      0,='Type flags' (see Note 1)
LA      1,='Size of field'
LA      2,=Address of field
```

The foregoing three instructions are generated only if POINTER was specified, or if POINTER was not specified:

```
SR      2,2
LA      3,='Type flags' (see Note 1)
LA      4,='Size of field'
LA      5,=Address of field
```

The foregoing four instructions are generated only if TALLYING was specified, or if TALLYING was not specified:

```
SR      5,5
L       15,V(ILBDCVB0)
BALR    14,15
```

Linkage generated by ILBDUST0 or ILBDSTG0:

```
SR      2,2
LA      3,='Type flags' (see Note 2)
LA      4,='Size of field'
LA      5,=Address of field
L       15,V(ILBDCVB0)
BALR    14,15
```

ILBDCVB1

Linkage generated for an UNSTRING verb:

If value is POINTER field

```
LA      2,1
```

or if value is TALLYING field

```
LA      2,0
LA      3,='Type flags' (see Note 1)
LA      4,='Size of field'
LA      5,=Address of field
L       15,V(ILBDCVB1)
BALR    14,15
```

Linkage generated by ILBDUST0 or ILBDSTG0:

```
LM      1,2,double-precision binary number
   or
L       2,single-precision binary number
LA      3,='Type flags' (see Note 2)
LA      4,='Size of field'
LA      5,=Address of field
L       15,V(ILBDCVB1)
BALR    14,15
```

where 'Type flags' bits have the following meaning:

Bits	Meaning
0-1	Unused
2	Indicates whether binary values are passed to, or to be passed from ILBOCVB in registers. See Notes 1 and 2.
3	Set to 1 if number being passed in register is a double-precision number. If bit 2 is not set to 1, this bit is meaningless.

4-7

Code	Field Type
0110	External decimal, unsigned
0111	External decimal, sign is trailing overpunch
1000	External decimal, sign is leading overpunch
1001	External decimal, sign is trailing separate character
1010	Eternal decimal, sign is leading separate character
1011	Binary
1100	Internal decimal, unsigned
1101	Internal decimal, signed

Note 1: Bit 3 of the "Type flags" is never set for the POINTER and TALLYING fields passed by the generated code for the UNSTRING verb. These fields are treated specially: when converting these fields to binary, the converted values are placed in a work area, called USTWRK, which is later used when ILBDUST0 is called by the generated code for the same UNSTRING verb; when converting these fields to decimal, the binary values are obtained from USTWRK.

Note 2: Bit 3 of the 'Type flags' must be set when ILBDCVB is entered under any other conditions than those stated in Note 1.

Output: The output from ILBDCVB0 is a binary number either in registers 2,3, or in the USTWRK work area. The output from ILBDCVB1 is an internal or external decimal number at the location specified in the calling sequence.

Calling Information: Called by the compiled code for an UNSTRING verb or by the subroutines ILBDUST0, ILBDINS0, and ILBDSTG0. Calls no other subroutines.

All Numeric Forms to External Floating-Point (ILBDTEF0)

Operation: Converts a single precision binary, a double precision binary, an internal decimal, or an internal floating-point number into an external floating-point number.

Linkage:

For single precision binary:

```
L      0,BI-number
L      15,=V(ILBDTEF0)
BALR   14,15
DC     XL1'Decimals in EF mantissa'
DC     XL1'Total length of EF number'
DC     XL1'EF parameter code'
         (See note at beginning of this
         section)
DC     XL1'Decimals in BI-number'
```

For double precision binary:

```
LM     0,1,BI-number
L      15,=V(ILBDTEF1)
BALR   14,15
DC     XL1'Decimals in EF mantissa'
DC     XL1'Total length of EF number'
DC     XL1'EF parameter code'
         (See note at beginning of this
         section)
DC     XL1'Decimals in BI-number'
```

For internal decimal:

```
ZAP    WORKA(16,13),ID-field
L      15,=V(ILBDTEF2)
DC     XL1'Decimals in EF mantissa'
DC     XL1'Total length of EF number'
DC     XL1'EF parameter code'
         (See note at beginning of this
         section)
DC     XL1'Decimals in ID number'
```

For internal floating-point: either

```
SDR    0,0
LE     0,FP-number
       or
LD     0,FP-number
```

Followed in either case by:

```
L      15,=V(ILBDTEF3)
CNOP   2,8
BALR   14,15
DC     XL1'Decimals in EF mantissa'
DC     XL1'Total length of EF number'
DC     XL1'EF parameter code'
         (See note at beginning of this
         section)
DC     XL1'Slack byte'
DC     XL8'FP-number'
         (of the form 10**X, where X is
         the number of digits in the EF
         mantissa)
```

Output: The external floating-point result is in WORKA+24(L,13) where register 13 points to the TGT.

Calling Information: Called by compiled code or by the object-time SYMDMP subroutine (ILBDMP23).

External Floating-Point to Internal Floating-Point (ILBDEFL0)

Operation: Converts an external floating-point number into an internal floating-point number.

Linkage:

```
L      3,=A(EF-number)
L      15,=V(ILBDEFL0)
BALR   14,15
DC     XL1'Decimals in EF mantissa'
DC     XL1'Total length of EF-number'
DC     XL1'EF parameter code'
         (See note at beginning of this
         section)
DC     XL1'Slack byte'
```

Output: An internal floating-point number in floating point register 0.

ARITHMETIC VERB SUBROUTINES

The five subroutines described below perform involved calculations, such as exponentiation, or calculations involving larger numbers. Arithmetic operations not in these categories are performed in-line and do not require use of the library.

Decimal Multiplication (ILBDMXU0)

Operation: Multiplies two 30-digit decimal numbers to produce a 60-digit decimal number. Input signs are expected to be C, F, or D.

Linkage:

```
ZAP    WORKA(16,13),MPLIER
ZAP    WORKA+16(16,13),MPCAND
L      15,=V(ILBDXMU0)
BALR   14,15
```

Output: The product, a 60-digit decimal number is placed in the 32-byte field following the multiplicand in the working cell area in the TGT.

Decimal Division (ILBDXDI0)

Operation: Divides a 60-digit decimal number by a 30-digit decimal number to yield a 60-digit decimal quotient. The dividend and divisor are both signed decimal numbers, right aligned in their fields.

Linkage:

```
MVC    WORKA(32,13),Dividend
         (if dividend is 32 bytes)
       or
XC     WORKA(16,13),WORKA(13)
         (if dividend is 16 bytes or
         less)
ZAP    WORKA+16(16,13),Dividend
```

Followed in either case by:

```
ZAP    WORKA+48(16,13),Divisor
L      15,=V(ILBDXDI0)
BALR   14,15
```

Output: The quotient, a 60-digit decimal number, is in the 32-byte field following the divisor in the working cell area in the TGT. The sign is determined by the rules of algebra from the dividend and the divisor signs. No remainder is returned.

Decimal Fixed-Point Exponentiation (ILBDXPR0)

Operation: Exponentiates any 30-digit packed decimal base to a binary exponent. This subroutine calls packed decimal multiplication and division routines.

Linkage:

```
ZAP   WORKA(16,13),BASE(L)
L     0,EXPONENT
L     15,=V(ILBDXPR0)
BALR  14,15
DC    XL1'Decimal places in base'
DC    XL2'Decimal places required in
        result'
```

Output: A 16-byte packed decimal number at the beginning of the working cell area in the TGT.

Floating-Point Exponentiation to an Integer Exponent (ILBDGPW0)

Operation: Exponentiates a double precision floating-point number to a binary exponent.

Linkage:

```
LD    0,BASE
      or
SDR   0,0
LE    0,BASE
```

Followed in either case by:

```
      0,EXPONENT
        (EXPONENT was converted to
        binary, if necessary)
L     15,=V(ILBDGPW0)
BALR  14,15
```

Output: The result is in floating-point register 0.

Floating-Point Exponentiation to a Noninteger Exponent (ILBDFPW0)

Operation: Exponentiates a long-form floating-point base to a floating-point exponent.

Linkage:

```
LD    0,BASE
      or
SDR   0,0
LE    0,BASE
```

Followed in either case by:

```
MVC   WORKA+8(8,13),EXPONENT
        (EXPONENT was converted into
        long-form floating-point,
        if necessary)
L     15,=V(ILBDFPW0)
BALR  14,15
```

Output: The result is in floating-point register 0. To avoid imaginary numbers (involving the square root of -1), the base is always treated as a positive number, and the result will always be positive. Any condition which would cause exponent overflow results in an answer equal to the largest floating-point number. Any condition which would cause exponent underflow results in an answer equal to 0.

DATA MANIPULATION SUBROUTINES

The subroutines described below manipulate data both in virtual storage and on files. They also perform some editing and initializing functions.

SORT (ILBDSRT0) And MERGE (ILBDSRT0 And ILBDMRG0) [CA]

Sort Operation: ILBDSRT0 acts as an interface between the COBOL generated object program and the Program Product Sort/Merge program. It links to the Sort/Merge program, using parameters from the COBOL object program. If INPUT PROCEDURE or OUTPUT PROCEDURE has been specified, ILBDSRT0 branches at exits from the Sort/Merge program to the sequence of instructions specified in the COBOL object program.

If, instead of the INPUT PROCEDURE, the USING option of the COBOL SORT statement has been specified, at the exit from the SORT/MERGE program the subroutine branches to the compiler-generated code to open the USING file(s). If more than one file is specified in the USING statement, they are all opened at once. The subroutine then reads every record from the first file until end-of-file, closes it and then reads all the records from the next file until end-of-file, closes it, and so on.

If, instead of the OUTPUT PROCEDURE, the GIVING option of the COBOL SORT statement has been specified, at the exit from the SORT/MERGE program the subroutine branches to the compiler-generated code to open the GIVING file. The subroutine then writes every record onto the GIVING file and closes it when the operations with it are complete. Finally, the subroutine returns control to the COBOL object program when the sort operation is complete.

Sort Flow of Control: Diagram 1 (see "Program Organization" Section) describes the logical flow among the three programs which are active during execution of a COBOL SORT statement. The statement has specified both INPUT PROCEDURE and OUTPUT

PROCEDURE; but checkpoint records are not to be taken.

The COBOL object program sets up the parameter list, and branches to ILBDSRT0. This parameter list consists of 10 address constants pointing to the card images describing the parameters for the Sort/Merge program (see Figure 1, items 1 through 6). The parameter list also contains the addresses of the three branch tables and SORT-RET cell in the TGT (see Figure 1, items 7 through 10). After initialization the subroutine links to the Sort/Merge program. When phase 1 of the Sort/Merge program reaches exit E15, it returns to the subroutine. The first time this path is followed, ILBDSRT0 branches to the routine in the COBOL object program which initializes the PERFORM statement of the input procedure specified. Control is then passed to that procedure.

When the RELEASE statement is encountered in the input procedure, control returns to the subroutine. The subroutine then establishes the linkage back to the statement after the E15 exit instruction of the Sort/Merge program. The Sort/Merge program then loops through its phase 1 operation until it is ready to receive another record. It then passes control to ILBDSRT0, which in turn passes control to the statement in the input procedure immediately following the RELEASE statement.

This same flow of control through the Sort/Merge program, ILBDSRT0, and the COBOL input procedure continues until the input procedure has released the last record. Then control passes to the end of the procedure, and by means of the subroutine, to phase 2 of the Sort/Merge program.

The interaction among the three programs during the output procedure is essentially the same as during the input procedure. Phase 3 of the Sort/Merge program returns to the subroutine at exit E35 whenever it is prepared to return a sorted record. Linkage between the subroutine and the output procedure is similar to that between the subroutine and the input procedure. After the last record has been returned by the Sort/Merge program, control returns through ILBDSRT0 to the COBCL object program at the instruction immediately following the one which originally branched to the subroutine.

Merge Operation: ILBDSRT0 and ILBDMRG0 act as the interface between the COBOL object program and the Sort/Merge Program (Program Number 5746-SM1).

ILBDSRT0 performs the following functions:

- Calls ILBDMRG0 for initialization
- Links to the Sort/Merge program using parameters from the COBOL object program.
- At exit E32 from the Sort/Merge program branches to ILBDMRG0.
- When a merged sequence is determined, branches at exit E35 from the Sort/Merge program to the COBOL object program OUTPUT PROCEDURE or to the code for the GIVING option which is the same as for the SORT statement.

ILBDMRG0 performs the following functions:

- At initialization saves the following information
  - -- SD buffer address
  - -- Address of the open USING files routine in the COBOL object program
  - -- Number of input files
  - -- Recording mode of SD
  - -- Address of error exit for VSAM files
- At exit E32 from the Sort/Merge program
  - -- Branches to the compiler-generated code to open all the USING files
  - -- Reads each record from the input files requested by the Sort/Merge program
  - -- Passes the record to the Sort/Merge program for merging with a record from other files
  - -- Performs a CLOSE operation on a file on which an end-of-file occurred

Merge Flow of Control: The flow of control for MERGE processing is shown in Diagram 2 in "Section 2: Program Organization." This diagram describes the logical flow among the four programs which are active resident in storage during execution of the COBOL MERGE statement.

The COBOL object program sets up the parameter list and branches to ILBDSRT0. This parameter list consists of ten address constants pointing to the card images describing the parameters for the Sort/Merge program (see Figure 2). Items 1 through 6 are set up by the COBOL compiler. ILBDSRT0 sets up the rest of the list, then links to ILBDMRG0 for initialization. On return from ILBDMRG0, ILBDSRT0 links to

theSort/Merge program. When phase 3 of the Sort/Merge program reaches exit E32, it returns to ILBDSRT0 which then branches to ILBDMRG0.

The first time this path is followed, ILBDMRG0 branches to the COBOL object program which opens all the input files and passes control back to ILBDMRG0 with pointers to opened files, DTFs, and BLs in WORKING CELLS of the TGT. ILBDMRG0 then reads a record from the input file requested by the Sort/Merge program, and establishes the linkage back to the statement after the E32 exit instruction of the Sort/Merge program.

The Sort/Merge program loops through exit E32 until a merged sequence is established. It then returns to ILBDSRT0 at exit E35. Linkage between ILBDSRT0 and the OUTPUT PROCEDURE is the same as that for the SORT statement. (If GIVING is specified, ILBDSRT0 writes a record onto an output file.)

This flow of control through the Sort/Merge program, ILBDSRT0 and ILBDMRG0 at exit E32, and ILBDSRT0 at exit E35 continues until ILBDMRG0 has released the last record to the Sort/Merge program and after the last merged record has been returned by the Sort/Merge program. Control is passed from ILBDSRT0 to the COBOL object program at the instruction immediately following the one that originally branches to the subroutine.

Parameters Passed to the Sort/Merge Program: ILBDSRT0 passes in register 1 a pointer to a ten-word parameter list (see Figure 1 for the SORT statement). The ten parameters contain addresses of the control areas that exist in virtual storage during execution of the SORT statement. The first six control areas are generated as EBCDIC literals by phase 51 of the COBOL compiler. They correspond to the control cards that are needed by the Sort/Merge program to define the specific sort operation. The next three control areas are tables of branch addresses that are located in ILBDSRT0. The final control area is a location SORT-RET in the TGT into which a return code is placed by the Sort/Merge program.

(1)	↑ SORT FIELDS=(p_1,l_1,f_1,s_1{,...p_{12},l_{12},f_{12},s_{12}}),WORK=x,[SIZE=value]
(2)	↑ RECORD TYPE={F / V},LENGTH={1 / (l_1,...,[l_5])}
(3)	↑ INPFIL EXIT
(4)	↑ OUTFIL EXIT
(5)	↑ OPTION [LABEL=(,,{S / U})],[STORAGE=value] or ↑ SORT/OPTION data-name
(6)	(a) ↑ MODS PH1=(,,E15),PH3=(,,E35) (b) MODS PH1=(,,E11,E15),PH2=(,,E21),PH3=(,,E31,E35)
(7)	↑ PH1 B E11 - CHECKPOINT ROUTINE B E15 - INPUT PROCEDURE OR READ USING FILE(S)
(8)	↑ PH2 B E21 - CHECKPOINT ROUTINE
(9)	↑ PH3 B E31 - CHECKPOINT ROUTINE B E32 - EXIT E32 IS NOT USED B E35 - OUTPUT PROCEDURE OR WRITE THE GIVING FILE
(10)	↑ Return Code (2-byte area in the TGT)
	Note: ↑ indicates "address of". For an explanation of each parameter, see the appropriate circled number under "Explanation of Parameter Lists" in this section.

Figure 1. Parameter List Passed by ILBDSRT0 for SORT

Explanation of Parameter Lists:

(1) SORT or MERGE Control Statement:

The FIELDS parameter describes the SORT or MERGE keys of which a maximum of 12 may be specified. For each key there are four parameters.

p_n=the position within the record, of the high order byte of the control data field

l_n=the length in bytes of the control data field

f_n=the format of the data in each of the control data fields, for example, ZD (zoned decimal), CSL (leading separate character for ASCII)

s_n=the sequence into which the control data field will be sorted, that is, A (ascending), D (descending)

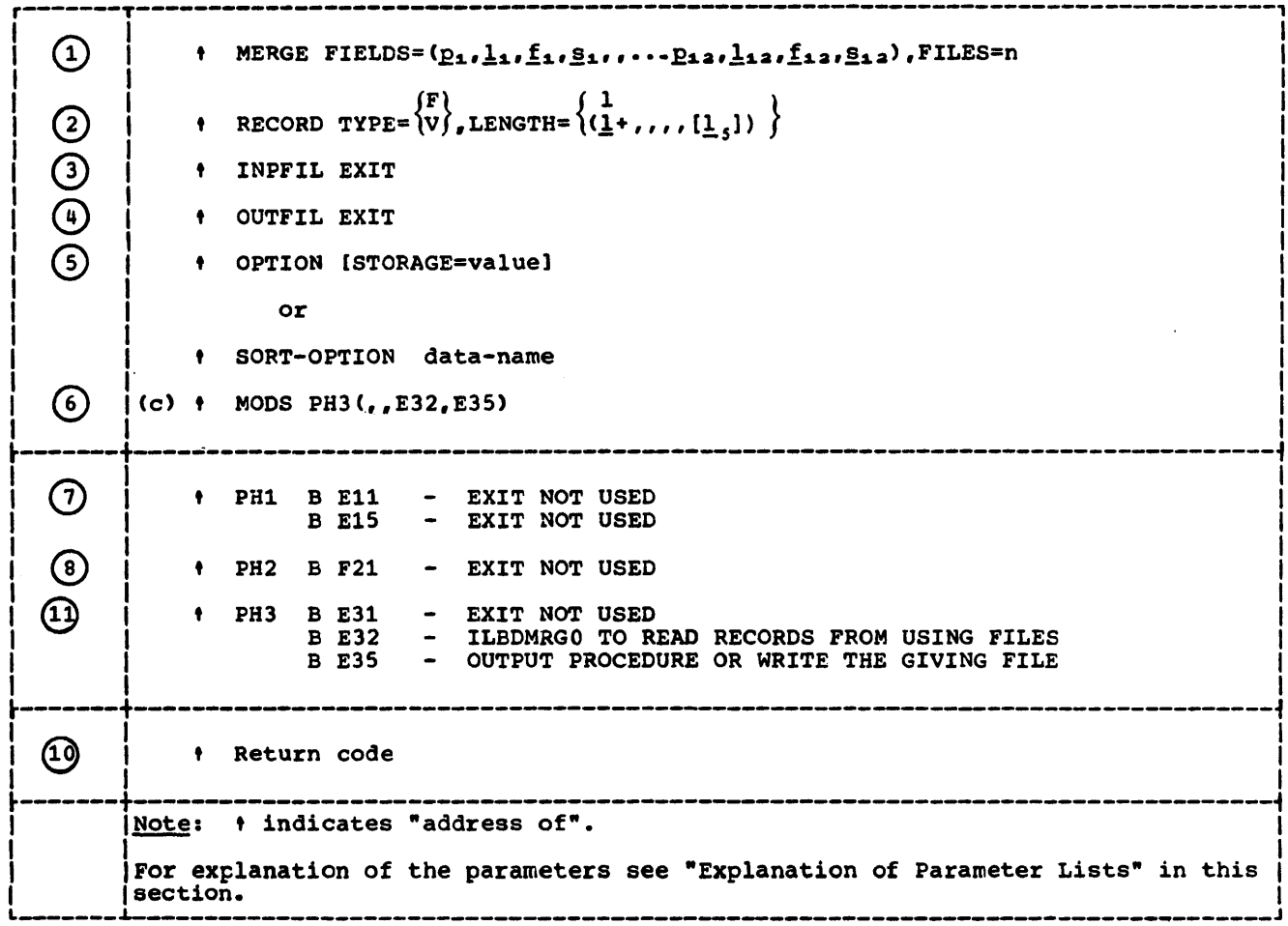


Figure 2. Parameter List by ILBDSRT0 for MERGE

The WORK parameter specifies the number of devices available for tape intermediate storage or the number of extents available for disk intermediate storage. For tape devices, the range of acceptable values is 3 through 9. For direct-access devices, the range of acceptable values is 1 through 8. If no value is specified, default values of 1 and 3 will be assigned for disk and tape, respectively.

The FILES parameter specifies the number of input files that are to be merged. If SORT-FILE-SIZE is specified, the SIZE= parameter is added for SORT only.

② RECORD Control Statement:

The TYPE parameter is used to differentiate between F (fixed length) and V (variable length) records.

The LENGTH parameter for fixed-length records specifies the number of bytes (l) of one logical record in the input file. For variable length records the LENGTH parameter specifies the maximum (l_1) number of bytes in a single input record. If SORT-MODE-SIZE exists, l is also specified.

③ INPFIL Control Statement:

The EXIT parameter indicates to the Sort/Merge program that all input records are supplied

at exit E15 and that a routine at that exit reads the input file and passes the records one at a time to the Sort/Merge program.

(4) OUTFIL Control Statement:

The EXIT parameter specifies to the Sort/Merge program that a routine at exit E35 will process each record after it has been sorted and that a routine at that exit writes the output file. This routine is the output procedure that has been specified or created.

(5) OPTIONS Control Statement:

The LABEL parameter indicates the type of label on the work files. It may be U (unlabeled) or S (standard). All work files must have the same type of label. (Not used for MERGE.) If SORT-CORE-SIZE exists, the STORAGE= parameter is added.

If SORT-OPTION is specified in the SD statement, the address of the data-name is passed to subroutine ILBDSRTO. The value contained in the data-name field may have the following format:

```
 OPTION

      [ PRINT          ]
      [ PRINT=NONE     ]
      [ PRINT=ALL      ]
      [ PRINT=CRITICAL ]

      [,LABEL=(,,work)]

      [ ,STORAGE= { n       } ]
      [           { nK      } ]
      [           { (n,VIRT)} ]
      [           { (nK,VIRT)} ]

      [,ALTWK]  [,ERASE]

      [,ROUTE=LST]
      [,ROUTE=LOG]

      [,SORTWK=work                      ]
      [,SORTWK=(work1,...workm)          ]
```

Note: At least one blank must follow the last operand. Figure 3 summarizes the SORT-OPTION operands and their defaults. The word OPTION must start in column 1.

PRINT Option

```
[ PRINT          ]
[ PRINT=NONE     ]
[ PRINT=ALL      ]
[ PRINT=CRITICAL ]
```

PRINT and PRINT=ALL specify that all messages are to be printed by the Sort/Merge program. This includes error and end-of-job messages, control card information, various size calculations, and other informative messages.

PRINT=NONE specifies that no messages are to be printed. It is useful if you have no alternate message device and do not want messages listed with other printed output. A message device need not be assigned.

PRINT=CRITICAL specifies that only messages resulting from conditions that can cause program termination are to be printed. For more details on these conditions and messages, refer to IBM DOS/VS Sort/Merge Programmer's Guide, Order No. SC33-4028.

Note: PRINT=ALL is assumed until the SORT-OPTION statement is read. Therefore, if PRINT=NONE or PRINT=CRITICAL are to be used, these options must precede all others in the SORT-OPTION statement.

LABEL Option

```
[LABEL=(,,work)]
```

This operand specifies the type of labels associated with the work files. The parameter represented by work is either S (standard labels) or U (unlabeled). This operand is required if the SORT-OPTION statement is specified and unlabeled work files are used. If it is omitted, standard labels are assumed for all files.

Statement	Operands	Comments
OPTION	PRINT={ALL/NONE/CRITICAL} or PRINT	Default=ALL
	STORAGE=n/(n,VIRT)/(nK,....)	Default: see discussion.
	LABEL=(,,work)	Default=standard labels.
	ALTWK	
	ERASE	
	ROUTE={LST/LOG}	Default: Ph0 message on printer and console and Ph1-3 on console.
	SORTWK={work/($work_1$,....$work_m$)}	Default=(1,2,...m)

Figure 3. Summary of SORT-OPTION Operands

Note: When standard labels are used, the Sort/Merge program uses the DOS/VS system facilities to process these labels. Unlabeled tape files are processed by the Sort/Merge program. No user programming is required.

STORAGE Option

$$\left[,\text{STORAGE=} \left\{ \begin{matrix} n \\ nK \\ (nK,VIRT) \\ (nK,VIRT) \end{matrix} \right\} \right]$$

This option is required to specify to the Sort/Merge program how much storage to use and whether it can fix pages.

n specifies a decimal number of bytes of storage to be made available to the Sort/Merge program (together with its user routines). nK specifies the decimal number of K (1024 bytes) of storage available.

The default is the value of the SIZE parameter on the EXEC job control statement. If both SIZE and STORAGE are specified, the lower value is taken. If neither is specified, the default is the partition size of the required size calculated by the Sort/Merge program (but at least 64K), which ever is smaller. The Sort/Merge program terminates if n is less than 16K bytes. If n is greater than the partition size, it is ignored.

If the Sort/Merge program is invoked from another program, the defaults are calculated in a similar way, but the value of the SIZE parameter and the partition size are adjusted downwards by the difference between the address of the Sort/Merge load point and the starting address of the partition.

If VIRT is specified, the Sort/Merge program does not try to fix pages when running in virtual mode. You may need to specify VIRT to prevent interference with other jobs running simultaneously, or to allow a user-written routine to fix pages. VIRT should be avoided whenever possible, since it has an unfavorable effect on Sort/Merge performance. It is ignored when the Sort/Merge program is running in real mode. The value in SORT-CORE-SIZE is ignored if the SORT-OPTION clause is specified.

ALTWK Option

ALTWK specifies an alternate work drive (tape only) in a sorting job. This doubles the maximum input file size allowed. The address of the alternate device must be different from the address of all other devices used in the job. Figure 3 shows the file name and default symbolic unit name.

ERASE Option

ERASE specifies that work data sets used during a sort are to be erased at the end of the job. It is ignored if 2400-series tapes are used for work areas. If the sort terminates abnormally,

- ERASE is performed unless the checkpoint facility has been specified;
- if ERASE is performed, and if a workfile has been pooled with output, the output file is also erased.

Note that the Sort/Merge program does not close work data sets, even when terminating normally.

ROUTE Option

```
[,ROUTE=LST]
[,ROUTE=LOG]
```

LST specifies that messages are to be routed to the SYSLST file by the Sort/Merge program. Messages requiring operator intervention are also printed on SYSLOG if allocated to a DOS/VS supported console device.

LOG specifies that messages are to be routed to the controls.

Note: The default is assumed until the ROUTE option has been read.

SORTWK Option

```
[,SORTWK=work                          ]
[,SORTWK=(work1,...workm)              ]
```

This operand specifies the logical unit numbers associated with the work files. The parameters within parentheses must be replaced by symbolic unit numbers of a maximum of three significant digits from 1 to 221, or a comma. When a comma is coded, or if the operand is omitted, the Sort/Merge program uses the default assignment. Figure 4 summarizes the file names and default symbolic unit names.

Use of Device	Filename	Symbolic Unit Name
work	SORTWK1	SYS001
	.	.
	.	.
	.	.
	SORTWK9	SYS(M)
ALTWK	SORTALT	SYS(M+1)
M=the number of work files, as specified in the SELECT statement for the SD file.		

Figure 4. Sort/Merge File Name and Default Symbolic Unit Names

(6) MODS Control Statement:

This statement specifies the exits used to branch out of the Sort/Merge program to the subroutine. The PHn entry specifies the phase in which the exit occurs. The Enn entry specifies the number of the exit used in branching to the subroutine. When RERUN has been specified in the COBOL source program (that is, when checkpoint records have been requested), the format generated is indicated by (b) in Figure 1. Otherwise, the format indicated by (a) in Figure 1 is used.

(a) This statement indicates to the Sort/Merge program that exit E15 is to be used in phase 1; no modification is to be made to phase 2; and exit E35 is to be used in phase 3.

(b) This statement indicates to the Sort/Merge program that exits E11 and E15 are to be taken in phase 1; exit E21 is to be used in phase 2; and exits E31 and E35 are to be used in phase 3.

(c) This statement indicates to the Sort/Merge Program that no modifications are to be made to Phases 1 and 2, and that Exits E32 and E35 are to be used in Phase 3.

(7) PH1 (Phase 1 Branch Table):

This branch table consists of branch instructions that the Sort/Merge program uses when a phase 1 exit is requested on the MODS control statement. It is not used for MERGE. The three branch tables are assembled in the SORT subroutine according to the form shown in Figure 1. To pass control to a routine at a

particular exit, the Sort/Merge program uses the branch table and a fixed displacement associated with each program exit and then branches to the routine indicated.

For example, when the Sort/Merge program goes to the E15 exit, it loads register 15 with the address of the beginning of the table and issues a BAL 14,4(15) instruction. The Sort/Merge program passes in register 1 the address of a parameter list containing pointers to any records, checkpoint lists, etc., applicable to that exit.

(8) PH2 (Phase 2 Branch Table):

This branch table is used to pass control to the Checkpoint subroutine (ILBDCKP0) during Phase 2 of the Sort/Merge program, if the E21 exit is specified (see Figure 1, item 6b). It is not used for MERGE.

(9) PH3 (Phase 3 Branch Table):

This branch table is used to return a sorted record to ILBDSRT0. It is also used to pass control to the Checkpoint subroutine, if the E31 exit is specified.

(10) Return Code:

The Sort/Merge program stores a return code of 0 or 16 indicating the success or failure, respectively, of the sort procedure. The user can test the return code by referring to the reserved word SORT-RETURN.

(11) PH3 (Phase 3 Branch Table)

This branch table is used to branch to ILBDMRG0 to get a record from one of the input files to be merged or to return a merged record to ILBDSRT0 from the Sort/Merge Program.

Linkage to ILBDSRT0 for SORT or MERGE:

```
MVC   PARAM CELLS,literal-1 for SORT FIELDS
MVC   PARAM CELLS,literal-2 for RECORD TYPE
LA    0,PARAM CELLS for SORT FIELDS
LR    1,0
LA    1,PARAM CELLS for RECORD TYPE
LA    2,INPFIL
LA    3,OUTFIL
LA    4,OPTIONS    (SORT without SORT-OPTION
                    specified in SD)

    or

LA    4,SORT-OPTION data-name  (SORT with
                                SORT-OPTION
                                data-name
                                specified in
                                SD)
LNR   4,4

     or

SR    4,4                      (MERGE)
STM   0,4,WORKCELLS
LA    0,MODS
LA    1,INPUT PROCEDURE
LA    2,Input buffer
LA    3,OUTPUT PROCEDURE
STM   0,3,WORKCELLS+16
LH    4,'number of USING files'
                                SORT only
STH   4,WORKCELLS+38
L     15,=V(ILBDSRT0)
BALR  14,15
```

The code generated to open USING files for MERGE and SORT and pass back DTF and BL addresses to ILBDMRG0 and ILBDSRT0 is as follows:

```
GN1  EQU  *
     ST   14,XSA     return address to
                     ILBDSRT0 or ILBDMRG0
```

The OPEN code for multiple file-names is as follows:

```
     L    1,DTF#2    Open coding for
     L    2,BL#2     multiple file-names
     STM  1,2,WORKING CELLS#1
     L    1,DTF#3
     L    2,BL#3
     STM  1,2,WORKING CELLS#9
     MVI  WORKING CELLS#9,X'80'   generated
                                  only if
                                  VSAM file
     .
     .
     .
     .
     L    14,XSA
     BC   15,14
GN2  EQU  *
```

Before control is passed to ILBDSRT0, addressability is set up for the parameters that are needed in the Sort/Merge program and for the input and output procedures that have been specified.

Output: If the GIVING option is employed, the output is the sorted or merged file; if not, the output is sorted records passed singly.

Linkage to ILBDMRG0 from ILBDSRT0:

At initialization the code is:

```
L     1,A(SDBUFFER)
L     2,A(USING GN)
LH    3,'NUM OF USING FILES'
LA    4,1              If SD recording
                       mode is V or S
      or
SR    4,4              If SD recording
                       mode is F
L     15,=V(ILBDMRG0)
L     5,A(ERROREXIT)
BAL   14,4(15)
```

At exit E32 from the Sort/Merge program the code is:

```
L     15,=V(ILBDMRG0)
BR    15
```

Output: The record is passed to the Sort/Merge program for merging.

Dummy SORT (ILBDDUM0)

Operation: This subroutine is a 2-byte dummy subroutine which is loaded after the object module. If the SYMDMP option is not specified, the load point of ILBDDUM0 is used as the load point for the DOS/VS Operating System SORT Program. If the SYMDMP option is specified, the load point of the SORT program is determined by adding the length of the allocated SYMDMP modules and tables to the load point of ILBDDUM0.

Linkage: None.

Output: None.

Move (ILBDVMO0)

Operation: Used when one or both operands are variable in length or exceed 4096 bytes in length and the MVCL instruction cannot be used because the operands overlap. The variable-length operand may exceed 256 bytes. The subroutine has two entry points, depending upon whether the move is left justified or right justified.

Linkage:

```
L     15,=V(ILBDVMO0) (left-justified)
   or
L     15,=V(ILBDVMO1) (right-justified)
BALR  1,15
DC    XL10'Operand-A Information'
        (See Note)
DC    XL10'Operand-B Information'
        (See Note)
```

Note: Substitute one of the following:

For a variable-length operand:

```
DC    XL1'Type code'
DC    AL3(displacement of the variable-
        length cell in the TGT from the
        base register code)
DC    AL1(base register code)
DC    AL3(displacement of base locator
        from the above base register)
DC    XL2'Displacement of item from
        BL address'
```

For a fixed-length operand:

```
DC    XL1'Type code'
DC    XL3'Length of operand'
DC    AL1(base code)
DC    AL3(displacement of item from
        above base)
DC    XL2'Displacement of item from
        BL address'
```

The Type codes are:

Bit	Meaning, if on
0	Figurative constant
1	Not used
2	Variable length
3	Direct pointer to the Program Global Table (for a literal)
4-7	Not used

Output: None.

Moving Characters (ILBDMOV0) [CB]

Operation: This subroutine executes an MVC instruction of any length.

Linkage:

```
L     3,LENGTH
L     5,A(Receiving field)
L     2,A(Sending field)
L     15,=V(ILBDMOV0)
BALR  14,15
```

Output: The MVC is executed if the length is positive.

Transform (ILBDVTR0)

Operation: This subroutine translates a field (operand) of variable length or of a length greater than 256 bytes. It uses the translate table, ILBDTRN0, which it moves to a work area and then modifies according to the needed transformation.

Linkage:

```
L     2,=A(ILBDTRN0)
L     15,=V(ILBDVTR0)
BALR  1,15
DC    XL1'Type code' (see Note)
DC    XL3'Length of item'
DC    AL1(base code)
DC    AL3(displacement of pointer in
        TGT or displacement of literal
        text)
DC    XL2'Displacement from BL'
```

Note: The Type code bits are:

Bit	Meaning, if on
0-1	Not used
2	Variable-length item
3	Direct pointer (for example, a pointer for a literal or TALLY)
4-7	Not used

Output: The data field is transformed as requested.

MOVE Figurative Constant (ILBDANF0)

Operation: This subroutine moves a figurative constant of more than one character into a nonnumeric receiving field. The result may be right or left justified.

Linkage:

```
L     0,Length of receiving field
LA    1,Receiving field
LA    2,Figurative constant
L     15,=V(ILBDANF0)
BALR  14,15
DC    X'00'(Flag byte:  Bit 0 = 1 if
        the receiving field is
        right adjusted)
DC    X'00'(Length of figurative
        constant)
```

Output: The receiving field is filled with the figurative constant.

MOVE to Right-Justified Field for System/370 (ILBDSMV0)

Operation: This subroutine moves characters into a right-justified receiving field when the user has specified IBM-370 in the OBJECT-COMPUTER paragraph and the receiving field is either greater than 512 bytes in length or variable in length.

Linkage:

```
LA    0,Receiving field
LH    1,Length of receiving field
LA    2,Sending field
LH    3,Length of sending field
L     15,=V(ILBDSMV0)
BALR  14,15
```

Output: The characters are transferred to a right-justified receiving field.

Calling Information: Called by compiled code.

Alphanumeric Edit (ILBDANE0)

Operation: This subroutine moves a data-name, literal, or figurative constant into a right- or left-adjusted alphanumeric edited field. Each group of X's in the PICTURE is treated as an individual field.

Linkage:

```
L     0,Length of sending field
LA    1,Sending field
LA    2,Receiving field
LA    3,Edit mask (see Note 1)
L     15,=V(ILBDANE0)
BALR  14,15
DC    X'00'(Flag byte; see
        Note 2)
DC    X'00'(Mask length)
DC    X'0000'(Receiving length)
```

Note 1: Edit mask is an encoded form of the COBOL alphanumeric edit picture.

Note 2:

Bit	Meaning, if on
0	Right-adjusted receiving field
1	Sending field is a figurative constant
2-7	Not used

Output: The completed alphanumeric edited move.

STRING (ILBDSTG0) [CBA]

Operation: The subroutine moves one or more data items contiguously into a specified receiving field following the rules for the STRING verb. The subroutine moves all or, when DELIMITED BY is specified, part of each data item in the sending fields. If the number of characters to be moved exceeds the length of the receiving field, or if the value of POINTER is less than one or greater than the size of the receiving field, an overflow condition exists. If an OVERFLOW routine was specified, it receives control in response to an overflow condition.

Linkage

```
LA      0,FIELDA
LA      1,FIELDB
LA      2,FIELDC
LA      3,receiving field
LA      4,length of POINTER
LA      5,POINTER
LA      15,=V(ILBDSTG0)
BALR    14,15

     .
     .
     .
FIELDA  DC CL2'Length of receiving field'
FIELDB  DC AL4(parameter set 1)
           AL4(parameter set 2)
              .
              .
              .
           XL1'80'
           AL3(last parameter set)
FIELDC  DC XL1'Number of sending fields in
             parameter set 1'
              .
              .
              .

           XL1 'Number of sending fields
           in last parameter set'
```

where
the parameter set format for sending fields and delimiters is as follows:

```
DC  XL1'Switch byte' (see Note 1)
DC  XL3'Size of field'
DC  XL1'Base of base locator'
DC  XL1 - Unused
DC  XL1'Locator of base locator'
DC  XL2'Displacement from base locator'
```

There is one parameter set for each sending field, followed by one set for the delimiter. However, if the delimiter is SIZE, X'FFFF' is generated instead of the parameter set.

Note: Switch byte has the following meanings:

Value	Meaning
X'10'	Field is a literal or figurative constant, pointer is direct
X'20'	Pointer is direct
X'40'	Size field contained the displacement of the VLC in the TGT for a sending field or delimiter

Output: The concatenated data items in the specified receiving field.

Calling Information: Called by compiled code. Calls subroutine ILBDCVB0 and ILBDACS0.

UNSTRING (ILBDUST0) [CBB]

Operation: The subroutine separates contiguous data and moves it from one sending field into one or more receiving fields according to the rules for the UNSTRING verb.

When the subroutine is entered for the first time for an UNSTRING statement, storage is obtained and initialization processing is performed. The subroutine checks whether the POINTER value, if specified, is less than one or greater than the size of the sending field. If so, the return address associated with the OVERFLOW option is taken.

Following the above processing, or upon subsequent entries to UNSTRING, or after processing each of the four possible types of fields, the subroutine checks for the end of the input parameter fields. If the end has been reached, end processing is performed and control is returned to the calling program. If the end of the sending field has not been reached when the final call to UNSTRING has been executed, an overflow condition exists and the return address associated with the OVERFLOW option is taken. If the end of the input parameter fields has not been reached, the subroutine determines what type of field has been specified as input: DELIMITED BY, RECEIVING, DELIMITER IN, or COUNT IN.

For a DELIMITED BY field, a Delimiter table is created and an entry is made for each delimiter. An entry contains the flag byte from the input parameters, and the address and length of the delimiter. The

count of delimiters processed is saved (if this is the first call) as the number of entries in the table and the input parameter field is updated by the length of the parameter field for DELIMITED BY. The subroutine then checks for the end of the input DELIMITED BY parameter fields and continues as described above. When there are no more DELIMITED BY fields, the subroutine checks for other types of fields.

For a RECEIVING field, the subroutine gets the address of the next byte of the sending field to be processed. If the end of the sending field has been reached, processing is terminated, and although no overflow condition exists, the return address associated with the OVERFLOW option is taken.

If the end of the sending field has not been reached, the subroutine checks for delimiters. If no delimiters were specified, the size of the field to be moved from the sending field to the receiving field is equal to the size of the receiving field.

If delimiters were specified, the subroutine scans the sending field for a match with any one of the delimiters stored in the delimiter table. If a match is found, the size of the sending field is equal to the length of the field extending from the location of the next byte of the sending field to be processed to the location of the first character of the delimiter or the end of the sending field, whichever comes first. If the match is not equal and if PROGRAM COLLATING SEQUENCE is in effect, the operands are translated according to the collating sequence specified. Information about the delimiter is saved for later use in processing the DELIMITER IN field.

When the length of the sending field has been determined, processing to prepare the data to be moved is performed. An internal routine performs the move according to the receiving field type. After moving the data, POINTER and TALLY, and the address of the next byte of the sending field to be processed are updated.

If a delimiter was found, the subroutine now checks to determine whether ALL was specified. If ALL was specified, the subroutine determines the number of complete delimiters that follow in the sending field and, accordingly, updates the size and end address of the delimiter field, which was saved for DELIMITER IN processing. In all cases, whether there is a delimiter or not, the subroutine updates the input parameter field by the length of the parameter field for RECEIVING and branches to check for other types of fields or for the end of the input parameter fields. The size of the isolated delimiter field is added to the POINTER value either at the beginning of processing for the next RECEIVING field or at the end of the subroutine if this is the last call.

For a DELIMITER IN field, the information stored while processing a RECEIVING field is used to determine how much of the delimiter field is to be moved to the DELIMITER IN field. The subroutine then handles the move in the same manner as for a RECEIVING field. After moving the delimiter field, the subroutine updates the input parameter field by the length of the parameter field for DELIMITER IN and branches to check for other types of fields or for the end of the input parameter fields.

For a COUNT IN field, the subroutine generates a call to the subroutine entry point ILBDCVB1, which performs the actual processing for this field. Upon return from ILBDCVB1, the subroutine updates the input parameter field by the length of the parameter field for COUNT IN and branches to check for other types of fields or for the end of the input parameter fields.

Linkage

If POINTER and/or TALLYING is specified, a call to ILBDCVB0 precedes these instructions.

```
     L     15,GN1
     BALR  1,15
        .
        .  (generated code for statements
        .    following the UNSTRING verb)
```

- First call to ILBDUST0 for an UNSTRING verb

```
GN1  ST    1,PARAM CELLS
        .
        .  (possible subscripting)
        .
     L     2,OVERFLOW return address
     MVI   WORKING CELLS, indicator whether
             OVERFLOW specified
     L     3.sending field address
     L     4,length of sending field
     L     15,GNx1
     BALR  1,15
     Input parameter fields (see note)
GNx1 L     15,=V(ILBDUST0)
     BALR  14,15
```

In addition to the above instructions, the following information is placed in the WORKING CELLS field of the TGT if needed:

Byte	Contents
0	=X'FF' if overflow specified =X'00' if overflow not specified
1	This COBOL program's character for COMMA
2	This COBOL program's character for DECIMAL POINT
3	This COBOL program's character for CURRENCY SIGN
4-7	Address of ILBDCVB0

In the TGT working cells, the TUNSF bit (X'20') in the TGWRKCL2 field (TGT + X'74') is set to indicate that this is the first call to ILBDUST0 for this UNSTRING verb. If this is the last call, the TUNSL bit (X'10') is set. (It is possible for both first and last indicators to be set on in the case where this is the only call to ILBDUST0.) In the case where this is a first call and a subsequent call is expected. TGWRKCL2 is set to X'00' on return to compiled code from any but the last call. In ILBDMNS0, the SCUSTWRK field (ILBDMNS0 + 8) contains either the address of a work area obtained through a GETVIS macro instruction or zeros to indicate that ILBDUST0 must obtain the work area. If the work area has been obtained, the initial POINTER and TALLYING values have been placed in bytes 4-19 of this work area.

- Subsequent calls for an UNSTRING verb:

```
        L     15,GNx2
        BALR  1,15
        Input parameter fields (see note)
GNx2    L     15,V=(ILBDUST0)
        BALR  14,15
```

- Last call for an UNSTRING verb:

```
        L     15,GNx3
        BALR  1,15
        Input parameter fields (see note)
GNx3    L     register, address of SUBCOM
        MVI   TGWRKCL2,TUNSL
        L     15,=V(ILBDUST0)
        BALR  14,15
```

OVERFLOW return address -- If POINTER and/or TALLYING is specified, a call to ILBDCVB1 is generated.

```
        L     15,PARAM cells
        BCR   15,15
```

Note: The format of the input parameter fields is as follows:

```
DC      XL1'Type Flags'
```

Bits	Contents
0-1	ID bits

Code	Meaning
00	DELIMITER field
01	RECEIVING field
10	DELIMITER-IN field
11	COUNT-IN field

2 — If 1, All specified for DELIMITER field (also may be set for COUNT-IN field with different meaning for use by ILBDCVB0 subroutine)

3 — If 1, base locator is direct; only valid for a DELIMITER field; 0 for other field types

4-7 — As indicated in the following chart:

Code	Meaning
0000	Variable group
0001	Alphanumeric
0010	Alphanumeric, right-justified
0110	External decimal, unsigned
0111	External decimal, trailing overpunch
1000	External decimal, leading overpunch
1001	External decimal, separate trailing
1010	External decimal, separate leading
1011	Binary
1100	Internal decimal, unsigned
1101	Internal decimal, signed

```
DC      XL3'Length to be considered from
        sending field AL3(VLC) if this is
        a variable group field'
DC      AL4(base locator)
DC      XL2'DISPLACEMENT' -- these two
        fields are used to compute address
```

The preceding ten bytes are present for all types of fields; the following fields are present only for type specified.

- DELIMITER Field

```
DC      XL1'NN' sequence number starting at
          zero
```

- RECEIVING or DELIMITER-IN field if numeric.

```
DC      XL1'Number of digits to right of
          decimal'
DC      XL1'Scaling factor'
```

Output: The characters are transferred to the receiving field.

Calling Information: Called by subroutine ILBDUST0. Calls no other subroutines.

INSPECT-(ILBDINS0)[CBC]

Operation: When this re-entrant routine receives control to implement the INSPECT statement, the compiler has already explicitly defined any implied operands. The four major sections of ILBDINS0 then perform as follows: XSETUP obtains a work area and performs initialization housekeeping. XDELIM sets up the delimiter limits for each clause, and builds a translate table. XSCAN scans the identifier, performing replacement and tallying as necessary. XTERM loops back to XDELIM if a Format 3 INSPECT has only completed the TALLYING portion; otherwise, it performs termination housekeeping.

Linkage

```
LA      1,parameter list
LA      13,TGT
L       15,V(ILBOINS0)
BALR    14,15
```

The parameter list is:

Word	Byte	Use
1	0	XX0 switches for ID-1 (internal format information)
	1-3	length of ID-1
2	0-3	address of ID-1

The following seven words are repeated for each TALLYING or REPLACING operation to be performed (the final such group is denoted by the high-order bit being on):

Word	Byte	Use
1	0	YY switches (same as corresponding verb)
	1	XX1 switches for OP-1 (internal format information)
	2	XX2 switches for OP-2 (internal format information)
	3	XX3 switches for OP-3 (internal format information)
2	0	set to zero
	1-3	length of OP-1 (TALLYING or REPLACING operand)
3	0-3	address of OP-1
4	0	set to zero
	1-3	length of OP-2 (comparand; zero if CHARACTERS)
5	0-3	address of OP-2
6	0	set to zero
	1-3	length of OP-3
7	1-3	address of OP-3 (INITIAL operand; zero if omitted)

Output: Updated TALLYING and REPLACING identifiers.

Calling Information: Called by the compiled code. Calls ILBDCMM0 (for GETCORE/FREECORE operations), ILBDCVB0 (for binary conversions), and ILBDACS0 (for alternate collating sequence comparisons).

SEARCH (ILBDSCH0)

Operation: This subroutine searches a table using a binary search technique and returns the address of a desired table entry to the calling routine. From one to twelve keys may be specified, all of which must be satisfied for a successful search. The table must have been presorted on all keys, and all entries must be of the same length. If the search is unsuccessful, control is returned to the AT END address specified by the caller. The subroutine is called by code generated from processing a SEARCH ALL statement.

Linkage:

```
LA    0,Search argument
LA    1,Table descriptor (See
        Note 1)
CNOP  2,4
L     15,=V(ILBDSCH0)
BALR  14,15
DC    x'nn'(See Note 2)
DC    X'nn'(Length of first key)
DC    X'nnnn'(Offset of first key
        from the beginning of table
        entry)
.     (Same 4 bytes
.     of information
.     for each key)
```

Note 1: The table descriptor is a 16-byte area starting at TEMP STORAGE-4 in the TGT and is in the following format:

Byte	Meaning
0-3	Table address
4-7	Maximum number of occurrences
8-11	AT END address
12	Number of keys
13	Not used
14-15	Length of a table entry

The search argument is in a location starting at TEMP STORAGE-2 in the TGT.

Note 2: The type of key is as follows:

Bit	Meaning
0	1=ascending; 0=descending
1	Binary
2	Packed decimal
3	Zoned decimal
4	Alphanumeric
5-7	Not used (all bits 0)

Output: If the desired entry is found, its address is returned in register 0, and control is returned to the instruction appearing after the in-line key descriptions. If the entry is not found, control is returned to the AT END address.

The instructions following the key entries cause the index-name associated with the level of the table being searched to be set to the displacement of the found entry.

Segmentation (ILBDSEM0) [CC]

Operation: This subroutine performs the loading and initializing for the segmentation feature of the compiler when LANGLVL(1) is used. If the GO TO statement has a VN as its operand, this subroutine will do one of the following:

1. Load and initialize, if the segment of destination is independent and not in virtual storage.
2. Load only, if the segment of destination is overlayable and not in virtual storage.
3. Initialize only, if the segment of destination is in virtual storage, independent, and not the same as the origin of branch.
4. Branch to the desired entry point, if the segment of destination is in the root segment.
5. Branch to subroutine ILBDDBG0 if the SYMDMP option is in effect.

If the GO TO has a PN as operand, the subroutine will load a segment if it is not in virtual storage.

ILBDSEM1 is an alternate entry point to the subroutine. If the subroutine is entered at ILBDSEM1, the Procedure Block for the PN is loaded into register 11, and the priority and PN address are calculated and loaded into register 0 to simulate the linkage to ILBDSEM0; then operation is the same as for entry point ILBDSEM0.

Linkage:

For programs for which the optimization option (OPT) has not been specified:

If GO TO with VN as operand:

```
L     15,=V(ILBDSEM0)
L     0,VN#
BALR  14,15
DC    X'PTY'
DC    X'00'
```

IF GO TO with PN as operand:

```
L     15,=V(ILBDSEM0)
L     0,PN#
LCR   0,0
BALR  14,15
```

For programs for which the optimization option (OPT) has been specified:

If GO TO with VN as operand:

```
L      15,=V(ILBDSEM0)
L      0,VN#
BALR   14,15
DC     X'PTY'
DC     X'00'
```

If GO TO with PN as operand:

```
L      15,=V(ILBDSEM1)
BALR   14,15
DC     X'Priority'
DC     X'Block number'
DC     XL2'Displacement of PN from
         block'
```

If GO TO DEPENDING ON:

Control passes to entry point ILBDSEM1 from subroutine ILBDGDO0 with register 14 pointing to a 4-byte parameter list as described above.

Output: There is no output from this subroutine.

GO TO DEPENDING ON (ILBDGDO0, ILBDGDO1, ILBDGDO2) [CCA]

Operation: These routines handle conditional independent segment refresh. ILBOGDO0 uses the value of a particular data name as an index into a list of constants for each PN specified and then transfers control to the proper PN. If the value of the data name is greater than the number of PNs specified, control returns to the next instruction after the calling sequence. The subroutine uses the set of constants to determine the address of the PN, loads the procedure block for that PN into register 11, and then branches to the PN. Entry points ILBOGDO1 and ILBOGDO2 are called to refresh an independent segment when the destination has (or may have) a different priority from the origin. ILBOGDO1 will initialize the PN cells of the target segment if it differs from the origin and is higher than 49; return is to the caller. ILBOGDO2 is invoked for a PERFORM n TIMES statement; it performs a similar initialization function, but does not return to the caller--rather, it goes directly to the destination segment.

Linkage: This subroutine is called only when the optimization option (OPT) is requested and a GO TO DEPENDING ON statement is used.

```
                LH     3,Number of PN's in lis
                L      1,Contents of data name
                L      2,=V(ILBDSEM1)
                         or, if the program
                         is not segmented,
                SR     2,2
                L      15,=V(ILBDGDO0)
                BALR   14,15
1 set of        DC     X'Priority'
constants       DC     X'Block number'
for each        DC     XL2'Displacement of PN
PN                       from block'
specified:
```

ILBDGDO1

```
  IC    RO,priority     Priority of origin
                        segment or, if
  SR    0,0             origin already
                        known to have
                        different priority
                        from target.

  IC    R1,priority     Target segment
                        priority
  L     15,=V(ILBDGD1)
  BALR  14,15
```

ILBDGDO2

```
  IC    R0,priority     Priority of origin
                        segment or
  SR    0,0             if origin already
                        known to have
                        different priority
                        from target.
  L     1,org.          Byte 0= priority o
                        target segment
                        Bytes 1-3= Address
                        within target
                        segment.
  L     15,=V(ILBDGDO2)
  BR    15
```

Output: There is no output from this subroutine.

DATE, DAY, and TIME (ILBDDTE0, ILBDDTE1, ILBDDTE2) [CCB]

Operation: The subroutine performs thre functions in response to the use of the DATE, DAY, and TIME special registers. ILBDDTE0 calculates the time in the forn

hour minute second hundredth-of-a-secon

ILBDDTE1 calculates the date in the form:

year month day

ILBDDTE2 calculates the day in the form:

year day

Linkages:

ILBDDTE0 (for TIME)

```
LA   2,receiving field
     (temp storage 2)
L    15,=V(ILBDDTE0)
BALR 14,15
```

ILBDDTE1 (for DATE)

```
LA   2,receiving field
     (temp storage 2)
L    15,=V(ILBDDTE1)
BALR 14,15
```

ILBDDTE2 (for DAY)

```
LA   2,receiving field
     (temp storage 2)
L    15,=V(ILBDDTE2)
BALR 14,15
```

Output: The date, time, or day is placed in temporary storage; the compiler then generates code to move the date, time, or day to the receiving field.

Calling Information: Called by compiled code. Calls no other subroutines.

SUBROUTINES FOR LIBRARY MANAGEMENT

The subroutines that control storage additions or deletions are described here.

GETCORE/FREECORE Subroutine (ILBDCMM0, ILBDCMM1) [CCC]

Operation:

This subroutine will get storage and free storage for COBOL library subroutines requiring storage additions or deletions. A GETVIS is always issued for a 4K or larger block. The larger GETVIS requests are made when the user's request plus chaining needs (8 bytes for each block) exceed 4K. All GETVIS block requests are rounded to the next 4K boundary. User requests are then chained into the 4K blocks retrieved and the address of the storage is returned to the user. User requests are rounded to the next 128-byte boundary.

ILBDCMM0 Linkage:

For GETCORE

```
L     0,length
L     15,=V(ILBDCMM0)
BALR  14,15
```

Address returned in R1.

For FREECORE:

```
L    1,address
L    15,=V(ILBDCMM1)
BALR 14,15
```

Output: User regulated blocks of storage chained into the storage chain.

TEST AND COMPARE SUBROUTINES

The subroutines described below test certain characteristics of items in virtual storage. Condition codes or return codes indicate the results of the test or comparison.

Class Test (ILBDCLS0)

Operation: This subroutine performs a test to determine whether a field is alphabetic, external decimal, or internal decimal. The field (operand) will be variable length or of a length greater than 256 bytes. The subroutine uses one of five tables:

ILBDATE0	(alphabetic)
ILBDETB0	(signed external decimal)
ILBDITB0	(signed internal decimal)
ILBDUTB0	(unsigned internal decimal)
ILBDWTB0	(unsigned external decimal, numeric edited, alphanumeric, and alphanumeric edited)

The address of the table is loaded into register 2. The tables are 256-byte translate tables which enable the subroutine to perform testing.

Linkage:

For fixed-length operands:

```
L      2,=V(Table)
L      15,=V(ILBDCLS0)
BALR   1,15
DC     XL1'Type code'
DC     XL3'Length of item'
DC     AL1(base code)
DC     AL3(displacement of pointer in TGT
          to data name
DC     XL2'Displacement of item
         from BL address'
```

For variable-length operands:

```
L      2,=V(Table)
L      15,=V(ILBDCLS0)
BALR   1,15
DC     XL1'Type code'
DC     AL3(displacement of the
         variable-length cell
         in the TGT)
DC     AL1(base code)
DC     AL3(displacement of item
         from above base)
DC     XL2'Displacement of item
         from BL address'
```

where the type code bits are:

Bit	Meaning, if on
0-1	Not used
2	Variable-length item
3	Direct pointer (for example, for a literal or TALLY)
4-7	Not used

Output: The condition code is set to 0 when the test is true, and to nonzero when the test is false.

Compare (ILBDVC00)

Operation: Compares two operands, one or both of which are variable in length or are greater than 4096 bytes in length. When control is returned to the object program, the condition code is set to indicate whether operand-A is less than, equal to, or greater than operand-B.

Linkage:

```
L      15,=V(ILBDVC00)
BALR   1,15
DC     XL10'Operand-A Information'
         (see note)
DC     XL10'Operand-B Information'
         (see note)
```

Note: Substitute one of the following:

For a variable-length operand:

```
DC     XL1'Type code'
DC     AL3(displacement of the variable-
         length cell in the TGT from the
         base register code)
DC     AL1(base register code)
DC     AL3(displacement of base locator
         from above base register)
DC     XL2'Displacement of item from
         BL address'
```

For fixed-length operand:

```
DC     XL1'Type code'
DC     XL3'Length of operand'
DC     AL1(base code)
DC     AL3(displacement of item from
         above base)
DC     XL'Displacement of item from BL
         address'
```

The type codes are:

Bit	Meaning, if on
0	Figurative constant
1	Not used
2	Variable length
3	Direct pointer to the Program Global Table (for a literal)
4-7	Not used

Output: The condition code is set to indicate whether Operand-A is less than, equal to, or greater than Operand-B.

Compare Figurative Constant (ILBDIVL0)

Operation: This subroutine compares a data-name operand and a figurative constant of more than one character. The figurative constant is always the second operand.

Linkage:

```
MVC    Param Cell-1,FIGCON
L      0,Length of figurative constant
L      1,Length of data name operand
LA     2,Param Cell-1
LA     3,Data name
L      15,=V(ILBDIVL0)
BALR   14,15
```

Output: The condition code is set to indicate whether the data-name operand is less than, equal to, or greater than the figurative constant.

Comparison with Alternate Collating Sequence (ILBDACS0, ILBDACS1) [CCD]

Operation: ILBDACS0 compares operand 1 to operand 2 (both unsigned display or group), where operand 1 is either an identifier or a literal (other than a figurative constant) and operand 2 is either an identifier or a literal (other than a multi-byte figurative constant). ILBOACS1 is similar, except that operand 2 is a multi-byte figurative constant (that is, all 'XX...' for XX with length 2 or greater).

Linkages:

ILBDACS0

```
LA   0,operand 1
L    1,length of operand 1
LA   2,operand 2 (omitted if a 1-byte
       figurative constant)
L    3,length of operand 2 (0 if a
       10byte figurative constant)
LA   13,TGT
L    15,=V(ILBDACS0)
BALR 14,15
```

In the TGT, #TB1PCS will have been set if an alternate collating sequence was specified; if so, #TPCSADR points to the transfer table.

ILBDACS1

```
L    0,length of figurative constant
L    1,length of identifier
LA   2,figurative constant
LA   3,identifier
LA   13,TGT
L    15,=V(ILBDACS1)
BALR 14,15
```

The TGT indicators are the same as for ILBDACS0.

Output: The condition code is set to high, equal, or low, according to the result of the comparison.

Calling Information: This re-entrant routine can be called by either the compiled code or by the ILBDINS0 subroutine. Calls no other subroutines. In the TGT of ILBDACS's caller, #TB1PCS will have been set on if the user specified an alternate collating sequence; if so, #TPCSADR points to the transfer table.

UPSI (ILBDUPS0)

Operation: This subroutine initializes the UPSI bytes in the TGT. It is called at the beginning of the program, if the user has specified UPSI in the SPECIAL NAMES paragraph.

Linkage:

```
L     15,=V(ILBDUPS0)
BALR  14,15
```

Output: If the UPSI bit is on, the corresponding byte in the TGT is set to X'F1'; otherwise, it is left at X'F0'.

Linkage (ILBDSET0)

Operation: This subroutine sets the switch byte of the Program Indicator subroutine (ILBDMNS0) to X'FF'. The linkage subroutine must be called by any program which is not an American National Standard or DOS/VS COBOL program before that program calls an American National Standard or DOS/VS COBOL subprogram. The name of this subroutine can be changed to any name specified by the user.

Linkage:

```
L     13,A(Savearea)
L     15,=V(ILBDSET0)
BALR  14,15
L     15,=V(COBOL subprogram)
BALR  14,15
```

Output: The switch byte of subroutine ILBDMNS0 is set to X'FF'.

Program Indicator (ILBDMNS0)

Operation: This subroutine contains a number of disparate items.

1. A one-byte switch used to indicate whether the program is a main program or a subprogram.
2. A flag byte containing:
   a. An alternate index build flag.
   b. A debug switch.
3. A pointer used as an anchor for the chain of storage obtained by the subroutine ILBDCMM0.
4. A pointer to the currently being used storage area.
5. The CSECT ILBDPRM0 used to access SYSPARM bytes using the COMRG macro, and to set the flags in the flag byte.

Linkage to ILBDPRM0:

```
L    15,=V(ILBDMNS0)
LA   15,16(15)
BALR 14,15
```

Output:

1. Main switch byte is set to X'FF' by a main program. Subprograms do not affect it.
2. Set by step 4.
3. Set by ILBDCMM0 to the address of the first block of storage obtained.
4. Sets the flags in step 2 above.

TIME-OF-DAY and CURRENT-DATE Subroutine (ILBDTOD0)

Operation: This subroutine, in response to the use of the TIME-OF-DAY special register, issues the GETIME macro instruction and calculates the time of day of the execution of the program. In response to the use of the CURRENT-DATE special register, the subroutine issues the COMRG macro instruction and calculates the date of the execution of the program.

Linkage:

TIME-OF-DAY

```
LA    2,receiving field
LH    3,length of receiving field
LNR   3,3
L     15,=V(ILBDTOD0)
BALR  14,15
```

CURRENT-DATE

```
LA    2,receiving field
LH    3,length of receiving field
L     15,=V(ILBDTOD0)
BALR  14,15
```

Output: The time in the form of hour/minute/second (HHMMSS) or the date in the form either of day/month/year (DD/MM/YY) or of month/day/year (MM/DD/YY) is stored in the receiving field. The form of the date is set at system generation time.

SYMDMP Address Test (ILBDADR0)

Operation: This subroutine tests the validity of an address calculated for a subscripted identifier or the validity of the starting and ending addresses of a variable-length identifier used as a receiving field in a MOVE instruction. The subroutine determines whether the address lies within a data area for any of the current programs in the run unit. Checking for valid addresses is only performed when all programs in the run unit are American National Standard or DOS/VS COBOL programs. The subroutine has two entry points.

It is called at entry point ILBDADR0 from the inline code generated to calculate the address of a subscripted item.

It is called at entry point ILBDADR1 from subroutines ILBDVM00 and ILBDMOV0 before a variable-length MOVE instruction.

Linkage:

From generated code:

```
LR    0,register containing data-name
        address
L     15,=V(ILBDADR0)
BALR  14,15
```

From subroutines ILBDMOV0 and ILBDMV00:

```
LR    0,register containing data-name
        address
LR    1,register containing length
L     15,=V(ILBDADR1)
BALR  14,15
```

Output: If the address or addresses are valid, control is returned to the caller. If the address or addresses are invalid, an error message (C170I - INVALID ADDRESS) is written on SYSLST and subroutine ILBDMP20 is called to produce a symbolic dump.

GENERAL DATA MANAGEMENT SUBROUTINES

The subroutines described below perform certain I/O operations, such as,, accepting and displaying information, opening and closing files.

DISPLAY (ILBDDSP0) [EA]

Operation: This subroutine is used (in conjunction with ILBDDSS0) to print, punch, or type data, usually in limited amounts, on an output unit. TRACE and EXHIBIT are special kinds of DISPLAY. The acceptable forms of data for this subroutine are:

1. Display
2. External decimal
3. Internal decimal
4. Binary
5. External floating-point

Internal decimal and binary are converted by the subroutine to external decimal. Internal floating-point numbers are converted to external floating-point before the subroutine is called and placed in the PARAM cells of the TGT.

Note: When OPT has been specified, subroutine ILBDDSS0 is sometimes called instead of subroutine ILBDDSP0. See "Optimizer DISPLAY (ILBDDSS0)" below.

When NOOPT is in effect or the ILBDDSS0 criteria cannot be met, ILBDDSP0 is called. This causes ILBDDSS0 to be included at link edit time. At object time the two subroutines act as a superset of the DISPLAY function. (See the ILBDDSP0 and ILBDDSS0 flowcharts for a visual representation of this interaction.)

Linkage:

For DISPLAY, the linkage is:

```
LA     2,=C(PROGRAM-ID)
         (If DISPLAY on SYSPCH)
L      15,=V(ILBDDSP0)
       or
L      15,=V(ILBDDSS0)
BALR   1,15
DC     XL2'Device code'
         (See Note 1)
operand information
         (See Note 2)
.
.      (Parameters)
.
DC     X'FFFF'
```

For TRACE, the linkage is:

```
L      15,=V(ILBDDSP0)
       or
L      15,=V(ILBDDSS0)
BALR   1,15
DC     XL2'Device code'
         (See Note 1)
DC     X'40'(Type code)
         (See Note 3)
DC     X'5'
DC     XL6 (EBCDIC generated card
         number)
```

For EXHIBIT, the linkage is:

```
.
.      (Test coding if CHANGED
.        case)
L      15,=V(ILBDDSP0)
       or
L      15,=V(ILBDDSS0)
L      2,A(Switch)
         (See Note 4)
BALR   1,15
DC     XL2'Device code'
         (See Note 1)
operand information
         (See Note 2)
.
.      (Parameters)
.
DC     X'FFFF'
```

Note 1: The device codes specify the device to be used. They are:

Code	Device
1	SYSLST
2	CONSOLE
3	SYSPCH
4	SYSIPT

Note 2: The operand information describes each item and has one of the following three formats:

1. Fixed length, ready to display:

```
DC     XL1'Type code'
         (See Note 3)
DC     XL3'Length of item'
DC     AL1(base code)
         (See Note 5)
DC     AL3(displacement of pointer
         in TGT to data-name or
         displacement of literal
```

General Data Management Subroutines

The following subroutine has been added.

ILBDTAB0

Operation: This subroutine contains a table of device-dependent information for tape or mass-storage devices and a search routine to get the table entry corresponding to the caller's parameter. The parameter may be either a PUB device code or a DTF device code.

Linkage:

```
         L     1,=A(search argument)
         L     15,=V(ILBDTAB0)
         BALR  14,15
         DC    AL1(indic)       (See Note 1)
         DC    AL1(default)     (See Note 2)
```

Note 1: indic is an index into branch vector, which indicates whether the parameter pointed to by register 1 is a PUB code (indic=0) or a DTF code (indic=4).

Note 2: default is an alternate search argument to use instead of the parameter in register 1 if the search does not yield a match for the parameter. Specify 255 if you do not want the default.

Output: Register 1 contains the address of the table entry containing the information corresponding to the caller's parameter. If the search is unsuccessful, register 1 is set to 0.

```
          text or TALLY)
     DC   XL2'Displacement'
```

2. Fixed-length binary or internal decimal (conversion is required):

```
     DC   XL1'Type code'
            (See Note 3)
     DC   XL1'Length of input item'
     DC   XL2'Length after conversion'
     DC   AL1(base code)
            (See Note 5)
     DC   AL3(displacement of item
            from above base)
     DC   XL2'Displacement'
```

3. Variable length:

```
     DC   XL1'Type code'
            (See Note 3)
     DC   AL3(displacement of the
            variable-length cell in
            the TGT)
     DC   AL1(base code)
            (See Note 5)
     DC   AL3(displacement of item
            from above base)
     DC   XL2'Displacement'
```

Note 3: The type code bits are:

Bit	Meaning, if on
0	Not used
1	TRACE item
2	Variable length
3	DIRECT pointer (for example, for a literal TALLY)
4-5	See below
6	Internal decimal item
7	Binary item

If bits 4 through 7 are all on, the item is numeric, ready to display. If bits 4 through 7 are all off, it is nonnumeric.

Note 4: The switch indicates whether or not an item should be exhibited. It is a 2-bit switch and corresponds to either one or two 10-byte operands. Figure 5 gives the switch codes, an indication of whether SEG1 (the first operand) is an alphanumeric literal, the meaning of codes, and the action that is taken.

An overriding situation occurs to the conditions in Figure 5 if register 3 contains a zero when the subroutine is called, indicating the first-time through requirement for the EXHIBIT CHANGED (NAMED) case. It is assumed that the second bit of the switch is on and only the first three conditions can occur.

Note 5: The base code indicates a register which contains a pointer to the TGT or the PGT.

Output: Lines of print via a PUT on the printer or the card punch, or via an EXCP on the console.

Optimizer DISPLAY (ILBDDSS0) [EB]

Operation: When OPT has been specified, this subroutine is used to print or type data of a certain kind on SYSLST or the console, respectively. Acceptable forms of data are the same as those listed for the ILBDDSP0 subroutine except the following:

- floating-pont data-names;
- floating-point literals;
- variable-length items;
- any DISPLAY verb where the sum of the operand lengths exceeds 120 bytes for SYSLST or 100 bytes for the console;
- any DISPLAY UPON SYSPCH.

Note: When any of the above items are to be printed, or typed, subroutine ILBDDSP0 is called together with Subroutine ILBDDSS0.

Switch	First Segment Alphanumeric Literal	Meaning	Action
01 or 00	--	Source literal or figurative constant	Display as 'SEG1' (up to 10 bytes)
11	yes	Named, changed	Display as 'SEG1 = SEG2' (up to 20 bytes)
11	no	Not named, changed	Display as 'SEG1' (up to 10 bytes)
10	yes	Named, not changed	Nothing displayed (up to 20 bytes)
10	no	Not named, not changed	Display n + 1 blanks when n is the length of SEG1 (up to 10 bytes)

Figure 5. Switch Codes for Display

Linkage: The linkage to this subroutine is the same as the linkage to subroutine ILBDDSP0.

Output: Lines of print via a PUT on the printer or via an EXCP on the console.

ACCEPT (ILBDACP0) [EC]

Operation: Services ACCEPT and STOP literal statements. For ACCEPT, a record is read from SYSIPT or the console. Lowercase alphabetic characters accepted from the console are translated to their uppercase equivalents. For STOP, the literal is typed on the console.

Linkage:

```
L     15,=V(ILBDACP0)
BALR  1,15
DC    XL2'Device code'
        (See Note 2)
DC    XL1'TYPE'
        (See Note 1)
DC    XL3'MNN' (If binary or internal
        decimal, M=length of input
        item and NN=length of conver-
        ted result.  If variable-length
        the three bytes are an ADCON
        pointing to the VLC-CELL.
        Otherwise, the three bytes are
        the length.)
DC    AL4(base locator)
        or
      AL4(operand-text)
        (if bit 3 of TYPE is set)
DC    XL2'Displacement of text
        from base'
```

Note 1: The TYPE bits are:

Bit	Meaning, if on
0-1	Not used
2	Variable-length
3	Pointer ADCON is direct
4-7	Not used

Note 2: The device codes specify the device to be used. They are:

Code	Device
X'0002'	CONSOLE
X'0004'	SYSIPT

For a STOP literal, the first byte of the device code is X'80'.

Output: The record accepted is placed in the operand specified. If it is a STOP literal, the message is typed on the console.

Checkpoint (ILBDCKP0) [ED]

Operation: This subroutine builds a table of pointers to DTF's of all magnetic tape units used in the problem program and its subprograms, and issues a CHKPT macro instruction, which will write checkpoint records on a user specified tape or disk checkpoint device.

Linkage: There are three sequences:

1. The first call, made during initialization is:

```
L     15,=V(ILBDCKP0)
CNOP  2,4
BALR  14,15
DC    XL8'0' (See Note)
DC    A(DTFPTR-1)
        (address of first DTF cell)
DC    A(DTFPTR-n)
        (address of last DTF cell)
```

2. When the specified number of records of the RERUN file has been read or written, the subroutine is called again, as follows:

```
L     15,=V(ILBDCKP1)
BALR  14,15
DC    X'N'
DC    XL7'External name'
```

3. During a sorting operation requiring checkpoints the SORT subroutine calls this subroutine as follows:

```
L     1,A(Physical IOCS list)
L     15,=V(ILBDCKP2)
BALR  14,15
```

Note: If SORT RERUN is specified, substitute the following two instructions:

```
DC    X'N'
DC    XL7'External name'
```

where:

N
: is the unit number of the checkpoint device, and

External name
: is the external name of the checkpoint file or SYSxxx if no external name is used.

Output: For DTFMT's, the DTF address is placed in a parameter list for CHKPT, and the macro instruction is issued. For more details on this macro instruction, refer to IBM DOS/VS Supervisor and I/O Macros Reference, Order No. GC33-5373.

OPEN ACCEPT File (ILBDASY0) [EE]

Operation: This subroutine ensures that SYSIPT is open. It is called if there is an ACCEPT FROM SYSIPT statement in a label declarative.

Linkage:

```
L     15,=V(ILBDASY0)
BALR  14,15
```

Output: SYSIPT is opened if it was not already opened.

OPEN DISPLAY File (ILBDOSY0) [EF]

Operation: This subroutine ensures that SYSLST or SYSPCH or both are open. It is called if there is a DISPLAY UPON SYSLST or a DISPLAY UPON SYSPCH or both in a label declarative.

Linkage:

```
L     15,=V(ILBDOSY0)
BALR  14,15
DC    X'NNNN'
```

where:

'NNNN' = '3000'
if both DISPLAY's are used

'NNNN' = '2000'
if the device is SYSLST

'NNNN' = '1000'
if the device is SYSPCH

Output: SYSLST and/or SYSPCH is opened if it was not already open.

Close With Lock (ILBOCLK0) [EG]

Operation: This subroutine receives control only when an OPEN is to be executed for a file and a CLOSE WITH LOCK for that file is specified anywhere within the program. The Pre-DTF switch is tested to determine if the file was closed with lock; a X'FF' indicates it was. If the file was not closed with lock, control is returned to the COBOL program. If it was closed with lock, the subroutine issues an error message and the job is terminated.

Linkage:

```
L     1,DTFPTR
L     15,=V(ILBDCLK0)
BALR  14,15
```

Output: A message is issued stating that an attempt has been made to reopen a file that was closed with a lock and the job is terminated.

User Standard Labels (ILBDUSL0) [EH]

Operation: This subroutine enables the user to write or check user standard labels. It determines the label condition (BOF, BOV, EOF, or EOV) and branches to the appropriate user procedure.

Linkage: None. The address of this subroutine is in the DTF and is branched to from LIOCS.

Output: Register 0 is set to decimal 8, 12, 16, or 20 depending on entry conditions (BOF, EOF, EOV, BOV, respectively). Register 4 points to the DTF. If there are tape output files and no user procedure, the COBOL label bit is turned on to indicate to LIOCS that no labels are to be written.

Nonstandard Labels (ILBDNSL0) [EI]

Operation: This subroutine reads and writes nonstandard labels and branches to the appropriate user label processing routine. It writes a tape mark after the last trailer label on each output reel.

Linkage: One of the following:

1. When the entry is from LIOCS, the entry point will be ILBDNSL0).

2. When the entry is from a user procedure and return is to the procedure:

```
L  15,=V(ILBDNSL1)
BR 15
```

3. When the entry is from a user procedure and return is to LIOCS:

```
L  15,=V(ILBDNSL2)
BR 15
```

Output: Register 4 points to the DTF and Register 1 to the label area.

Error Messages ($$BCOBER) [EJ]

Operation: This subroutine prepares input/output error messages to be printed by the Error Message Print Subroutine ($$BCOBR1). After preparing the error message, it places the address of the

message in register 0. Then it fetches $$BCOBR1, overlaying itself. See the descriptions of ILBDSAE0, ILBDISE0, and ILBDDAE0.

Linkage:

```
LA    4,DTF-8
LA    0,ERRCODE
SLL   0,24
OR    0,4
LA    1,=C'$$BCOBER'
SVC   2
```

Output: Register 0 contains the address of an error message to be written by $$BCOBR1 on SYSLOG or SYSLST.

Error Message Print ($$BCOBR1) [EK]

Operation: This subroutine prints the input/output error messages prepared by the Error Messages Subroutine ($$BCOBER) and provides a dump if the DUMP option is in effect. After the subroutine prints the error message, it tests the DUMP bit. If the bit is on, it calls the $$BPDUMP Subroutine via a SVC 2 instruction. If the DUMP bit is off, $$BCOBR1 returns control to the routine which fetched $$BCOBER. In either case the fetching routine determines if the job should be cancelled or control transferred to the Debug Control subroutine (ILBDDBG0). See the descriptions of subroutines ILBDSAE0, ILBDISE0, and ILBDDAE0 for further information on input/output error handling.

Linkage:

```
LA    1,=C'$$BCOBR1'
SVC   2
```

Note: Register 0 contains the address of the message to be printed.

Output: An error message on SYSLOG and/or SYSLST and, optionally, a PDUMP.

SYMDMP Error Message ($$BCOBEM) [EL]

Operation: This subroutine (a transient) puts the correct error message into the buffer of the PRINT routine (ILBDDBG1). When this subroutine is fetched by the PRINT routine, register 0 contains the error number in the high-order byte and the address of the buffer in the low-order byte and the address of the buffer in the low-order three bytes. If the error number does not fall within the range of errors contained in the subroutine, control is returned to the fetching program.

Linkage:

```
L     0,CURRBUFF Error code and buffer
        address
LA    1,C'$$BCOBEM'
SVC   2
```

Output: An error message on SYSLST.

3886 Optical Character Reader (OCR) Interface (ILBDOCR0) [Chart EM]

Operation: ILBDOCR0 handles all input/output operations with the 3886 Optical Character Reader and builds the OCR File Control Block required for this purpose. The subroutine receives control from the COBOL program via the CALL statement with the USING parameter for action requests and from the DOS/VS system for error recovery and end-of-file condition.

For an action request the functions of the subroutine are as follows:

- Validate operation code (See Note 1)
- Validate OCR-file identification by searching OCR File Control Block chain
- Test for valid sequence of WAIT and READO operations
- Call action routine to issue appropriate macro instruction for request (see Note 1)
- Build OCR File Control Block (via GETVIS) for OPEN requests and release OCR FCB (via FREEVIS) for CLOSE requests
- Set Status Key (See Note 2)

Note 1: Valid OCR operations and the DOS/VS macro instructions issued for each are listed below:

OCR-operation	DOS/VS macro instruction
OPEN	OPEN
CLOSE	CLOSE
READ	READ and WAITF
READO	READ
WAIT	WAITF
SETDV	SETDEV
MARKL	CNTRL
MARKD	CNTRL
EJECT	CNTRL

Note 2: The status key contains a completion code returned to the COBOL program. The codes, their meanings, and the action requests which generate them are listed below:

Code	Meaning	Action Request
00	Successful completion	OPEN, CLOSE, READ, READO, WAIT, MARKL, MARKD, EJECT, SETDV
10	End-of-file	READ, WAIT, MARKL, MARKD, EJECT, SETDV
31	Mark Check	EJECT
32	Nonrecovery error	READ, READO, WAIT, MARKL, MARKD, EJECT, SETDV
33	Incomplete Scan	READ, WAIT
34	Mark Check & Equipment Check	EJECT
39	Permanent error	READ, READO, WAIT, MARKL, MARKD, EJECT, SETDV
92	Logic error	OPEN, CLOSE, READ, READO, WAIT, MARKL, MARKD, EJECT, SETDV
93	Insufficient storage	OPEN
95	Invalid Parameter	OPEN, READ, READO, MARKL, MARKD, EJECT
99	Unrecognizable operation	

Linkage: Called by compiled code for the CALL statement.

Note: The user must set appropriate fields in the identifier data area before issuing the CALL statement with the USING option. (Refer to the 3886 statement in IBM DOS/VS COBOL Programmer's Guide, Order No. GC28-6478.)

Output: The OCR File Control Block is built in virtual storage; the indicated action request is performed; a return code is entered in the Status Key field of the OCR file data area.

SEQUENTIAL ACCESS DATA MANAGEMENT SUBROUTINES

The subroutines described below handle some special I/O operations for the sequential access method.

SAM I/O Subroutine (ILBDSIO0) [F]

Operation:

A single entry, ILBDSI01, for initialization of SAM XDTF entry, save registers, and sets the READ, WRITE, REWRITE, and WRITE ADVANCING entry points to the logic error internal address (RETLOGST) within ILBDSIO0. The ILBDSIO1 entry point is invoked from the inline code during INIT3 processing. Any non-SAM DTFs found during initialization processing are bypassed.

A single entry, ILBSDIO0, for all open/close requests, save registers, and then transfer to the appropriate action routine for the request, as follows.

On each OPEN/CLOSE request from the inline code, a single SAM DTF address and the specified OPEN/CLOSE Option are passed to entry point ILBDSIO0.

If the request is for OPEN, the OPEN PROLOG processing section is entered. There, the single XDTF control block for the file is obtained by using the SAM DTF address. The XDTF is examined to ensure that no logic error is involved in the request. If a logic error is encountered, this fact is noted in the status key (when present) and in the XDTF field XDTFSTAT.

If CLOSE with LOCK has occurred for this file and file status was not specified, ILBDSIO0 calls ILBDCLK0 to send a message and terminate the run unit.

If LINAGE was specified, those values are saved via ILBDSPA1.

Lastly, the DOS data management OPEN/CLOSE function is invoked. The DTF address is placed in registers 0 and 4, and the OPEN or CLOSE macro is issued after the inline registers are restored.

Upon successful return from the OPEN/CLOSE function, and either the OPEN or CLOSE EPILOG section of ILBDSIO0 is entered before returning to the inline COBOL code.

In the OPEN EPILOG, for any SAM DTF file opened for input or input/output, the appropriate EOF address in the DTF is set to an internal ILBDSIO0 address. This address is used for further EOF file processing when the EOF condition occurs. The user's EOF GN address is placed in the XDTF control block in line code for executions for READ.

If the file is DTFMT, multi-file reel with labels omitted, ILBDMVE0 will set the EOF status, and the address of ILBDMOE0 (set by phase 21) is not replaced.

If the file is DTFSD open for output, the internal address of the ILBDSIO0 code for end of extent processing is placed in the DTFSD at the appropriate address.

The invalid key address is placed in the XDTF control block during execution of the inline code for WRITE. If the end of extend condition occurs, ILBDSIO0 passes control to the specified user procedure.

Finally, the transfer address to be used by the inline READ or WRITE code expansion is set to the appropriate values corresponding to the OPEN mode requested.

For files with nonzero initial LINES-AT-THE-TOP, ILBDSIO0 invokes ILBDSPA1 to space the specified number of lines during processing of the first WRITE request for the file. At this point, OPEN processing is complete, the inline registers are restored, and control is returned to COBOL inline code.

The CLOSE EPILOG processing section frees the work area obtained during OPEN processing.

The inline code expansions for I/O action requested use the READ, WRITE, REWRITE, and WRITE with ADVABCUBG entry points set during OPEN processing in the XDTF.

The choice of entries set by OPEN is determined by the OPEN mode requested, file status specification, and the type of records in the file.

All I/O requests (non-advancing) for variable block records are handled via calls to ILBDVBL0. Other record types are handled by the appropriate DOS data management GET or PUT macro interface.

Requests for WRITE ADVANCING are routed via ILBDSPA1.

If EOF, end of extent, or logic errors occur, the appropriate action to exit to the user GN address is taken, after file status is set.

ILBDSIO0 also invokes the user error declaratives when the appropriate condition occurs. The user declarative addresses are found in pre-DTF control block.

ILBDSAE0 still invokes the user I/O error declaratives, but ILBDSIO0 sets the file status and determines the inline return point.

ILBDSIO0 Linkages

Entry Point ILBDSIO1

Purpose: Handles initialization of SAM XDTF.

Linkage: Called by inline code (from INIT3 code).

R14 = points to 4-byte inline parameter
R13 = loaded with TGT address
R12 = loaded with PGT address

The parameter list contains:

HWORD1 - displacement of first DTF cell in TGT
HWORD2 - number of DTF cells in TGT
- (end of parameter list return point)

Input: See Linkage above.

Output: Requested initialization operation is performed on various areas of XDTF. The I/O transfer address in the XDTF is set to its initial "logic error" values.

Entry Point ILBDSIO0

Purpose: Entry point for OPEN/CLOSE requests and CLOSE UNIT requests. Called by inline expansion of OPEN/CLOSE verbs.

R1 = address of parameter list for OPEN/CLOSE request
R14 = return address
R13 = TGT address
R12 = PGT address

Note: Where appropriate, R2 contains address of BL cell for the file.

The parameter list contains (word boundary):

Bytes 1-4	address of DTF	
Byte 5	option byte for OPEN	
	0123 4567	bit positions
	0000 0000	input
	0000 0001	input, no rewind
	0000 0010	input, reversed
	0000 0011	input, reversed no rewind
	0000 0100	output
	0000 0101	output, no rewind
	0000 1100	I/O
Byte 5	option byte for CLOSE	
	0123 4567	bit positions
	0000 0000	close rewind
	0000 0001	close no rewind
	0000 0010	close lock
	0000 1000	close reel rewind
	0000 1001	close reel, no rewind

```
Byte 6      command byte
            0123 4567  bit positions
            0001 0000  OPEN request
            0001 0100  CLOSE request
            0001 1000  CLOSE reel/unit
                       request
```

The above parameter list is generated inline and pointed to by register 1.

Note: The generated inline code still sets the appropriate DTF bits for the rewind function in the DTF fields, and the pre-DTF byte for open mode, in order to maintain existing interfaces to other library routines (i.e. label handling, etc.).

Linkage: Called by inline code expansions for I/O action verbs READ, REWRITE, WRITE, and WRITE WITH ADVANCING.

Purpose: I/O function support for action verbs. The action verbs are invoked by the inline code via a transfer vector that is up in XDTF, the DTF extension control block of SAM files at open and close.

For READ:

```
  R1  = address of DTF
  R2  = BL address of record (if
        appropriate)
  R4  = address of the XDTF
  R14 = return address
  R15 = address that is in XDTFRD
```

For REWRITE:

```
  R1  = address of DTF
  R2  = BL address of record (if
        appropriate)
  R4  = address of XDTF
  R14 = return address
  R15 = address that is in SDTFRW
```

For WRITE (without advancing):

```
  R1  = address of DTF
  R2  = BL address of record (if
        appropriate)
  R4  = address of XDTF
  R14 = return address
  R15 = address that is in XDTFWR
```

For WRITE (with ADVANCING clause):

```
 *R0  = address of record if AWO
 *R1  = address of DTF
 *R2  = BL address of record
 *R3  = length or record
 *R4  = address of XDTF
  R14 = return address
        +4 if ID not specified
        +8 if ID is specified
 *R14   +0 address parameter list
```

* The contents of the registers and the option byte setting are those specified for module ILBDSPA0, which is called by ILBDSIO0 to support WRITE WITH ADVANCING statements.

Note: In the case of the WRITE ADVANCING data name identifier, the address of the identifier is placed in the inline generated code parameter list. ILBDSIO0 loads the address of the identifier prior to the call to ILBDSPA1.

Parameter list for WRITE WITH ADVANCING:

```
Byte 1 parameter 1
       0123  4567  bit positions
               yy  remainder of integer/3
             00    integer
             01    identifier
             10    mnemonic
          0        S/370 control
                   characters
          1        ASA CC, no befores for
                   file at all
          0        BEFORE, this statement
          1        AFTER, this statement
       00          binary identifier
       01          packed decimal
                   identifier
       10          zoned decimal
                   identifier

Byte 2 parameter 2
       0123  4567  bit positions
         zz        either mnemonic skip
                   code, or quotient of
                   integer/3 or length
                   of identifier in
                   digits.
Byte 3 parameter 3
       0123  4567  bit positions
       ----  0000  fixed length record
       ----  0001  variable unblocked
       ----  0010  variable blocked
                   (not AWO)
       ----  0100  undefined
       ----  1000  apply write only

Byte 4 parameter 4
       0123  4567  bit positions
       ----  0001  with code (RW
                   specified)
       1000  ----  advance page
       0100  ----  EOP
       0010  ----  positioning

Bytes 5-8    address of identifier, if
             specified
```

Interface: Same as interface to ILBDSPA0 except for bytes 5-8, which contain the address of the identifier. This address is passed to ILBDSPA0 in register 4.

XDTF, PREDTF, and DTF Control Block Structure

Only one XDTF control block is generated for any SAM file. (This control block is generated in phase 21.) The XDTF control block contains those fields necessary to support SAM I/O with file status and linage clause.

The XDTF contains a transfer address for each of the following I/O action verbs: READ, REWRITE, WRITE, and WRITE WITH ADVANCING clause. The XDTF control block also contains status fields and the user-specified EOF and end of extent addresses.

For a file opened only one way in the program, a DTF control block is generated for the file. The SAM DTF control blocks are DTFCD, DTFMT, DTFPR, DTFSD, and DTFDU.

Preceding the DTF control block is the pre-DTF control block associated with that DTF. The pre-DTF control block contains the pre-DTF status byte, the error declarative address, and the label declarative address. These pre-DTF control blocks have been adjusted in size for all SAM DTFs, and an address field has been added that contains the address of the single XDTF control block for the file. The presence of the XDTF control is indicated by setting the XDTF bit in the PCE-DTE byte to one (this is done by ILBDSIO0 at open).

Since the address of the XDTF is now always at a known offset in the pre-DTF control block, the XDTF address can always be located from the DTF address.

On all I/O action requests from inline source code, the XDTF address is found in register 4 and the DTF address is found in register 1.

For any given SAM DTF, the pre-DTF and DTF control are in contiguous storage. If the file is opened in only one way in the COBOL program, the XDTF control precedes the pre-DTF, DTF pair in storage. For files opened in more than one way, there are multiple pre-DTF, DTF control blocks generated; a pre-DTF, DTF combination for each way the file is opened.

There is only one XDTF control block generated for each file. The address of this control block is found in each pre-DTF control block.

The primary DTF field in the DTF is initialized with the address of the DTF that has the XDTF control block associated with it. This is the only XDTF control block generated for that file.

For example, if the DTFMT file is opened for INPUT, OUTPUT, and INPUT-REVERSED in the same program, then the INPUT pre-DTF, DTF pair will have the XDTF control block associated with it in contiguous storage and that its DTF address is placed in the TGT DTF address slot.

The other pre-DTF, DTF structures for OUTPUT and INPUT-REVERSE will be generated and the address of each will be found in the pre-DTF control for each structure.

The address of these pre-DTF, DTF pairs is located via the secondary DTF pointers in the PGT. Note that the primary DTF combination also has a secondary cell associated with it. Prior to any OPEN/CLOSE request, the secondary DTF for the specified file is moved to the primary cell in the TGT. Register 1 is always loaded from the primary DTF cell.

SA Printer Spacing (ILBDSPA0) [FA]

Operation: This subroutine performs printer spacing; that is, it handles the WRITE statement with the ADVANCING option. It calls subroutine ILBDVBL0 to write variable-length blocked records.

Entry point ILBDSPA1 is called by ILBDSIO0 to handle WRITE ADVANCING together with any linage clause information from the XDTF for the file. If LINES-AT-THE-TOP is specified for the file, ILBDSPA1 writes the necessary blank lines together with the first write request.

In simple cases, ILBDSPA1 issues a PUT macro directly to write the line. Otherwise, it creates an appropriate parameter list and calls ILBDSPA0 to perform the writes. ILBDSPA0 is also called directly from inline code to process WRITE ADVANCING in LANGLVL(1) programs.

Linkage:

```
L     0,A(Record) (If APPLY WRITE-ONLY)
L     2,BUFPTR (If no APPLY WRITE-ONLY)
L     4,A(Identifier)
L     1,DTFPTR
L     3,RECORDLEN
L     15,=V(ILBDSPA0) or (ILBDSPA1)
BALR  14,15
DC    B'01234567' (see note 1)
DC    X'ZZ' (see note 3)
DC    B'01234567' (see note 2)
DC    B'01234567'(see note 4)
```

Note 1: Substitute binary digits as follows:

```
For 01:  00 if a binary indentifier
         01 if a packed decimal
            identifier
         10 if a zone decimal
            identifier

For 2:    0 if before
          1 if after

For 3:    0 if System/360 control
            character
          1 if ASA control characters
```

For 45: 00 if integer
01 if identifier
10 if mnemonic

For 67: The remainder of integer/3.

Note 2: Substitute binary digits as follows: (1, 2, and 3 are not used):

For 0: 1 if ASCII file

For 4567: 0100 if undefined
1000 if APPLY WRITE-ONLY
0000 if fixed
0001 if variable unblocked
0010 if variable blocked
(not APPLY WRITE-ONLY)

Note 3: ZZ = mnemonic skip code, or quotient of integer/3, or length of identifier.

Note 4: This byte is only used by the ILBDSPA1 entry point. Substitute binary digits as follows:

For 0: 1 indicates ADVANCING PAGE
For 1: END OF PAGE specified
For 2: AFTER POSITIONING specified
For 7: 1 indicates WITH CODE specified (REPORT WRITER)

Output: The user's record, with proper spacing, is written on his output file. IOREG (+4 if variable blocked records) is forwarded to main line.

SA Variable-Length Record Output (ILBDVBL0) [FB]

Operation: This subroutine writes variable-length blocked records. It calls ILBDMOV0 to move records into a buffer.

Linkage:

```
L     1,DTFPTR
L     2,A(record)
L     3,Record length
L     15,=V(entry point)
BALR  14,15
```

where:

entry point
is ILBDVBL0 if the subroutine was called by ILBDSPA0, or ILBDVBL1, if the subroutine was called by the main-line program.

Output: The record is written and the IOREG is advanced past the record length field.

SA Error (ILBDSAE0) [FC]

Operation: This subroutine handles errors on DTFMT and DTFSD files. If an XDTF is present and file status has been specified, then file status is set in the XDTF. If user error bytes are to be set, they are set and either an exit to a user error routine is made or, if file status has not been set, an error message is printed by fetching $$BCOBER. If $$BCOBER is fetched, an appropriate message is printed on SYSLOG and SYSLST by $$BCOBR1. If a dump is not required, return is made to ILBDSAE0; if it is, $$BPDUMP is called. $$PDUMP provides the dump and returns control to ILBDSAE0. If ILBDDBG2 (the STXIT routine) is in the load module, control is passed to it. If it is not, the job is cancelled.

Linkage: None. Control is transferred to this subroutine through LIOCS. The address of the subroutine is in the DTF.

Entry points are:

ILBDSAE0 (ADDR in ERROPT field of DTF)
ILBDSAE1 (ADDR in WRLERR field of DTF)

Output: Register 0 contains the error code and the address of DTF-8 when fetching $$BCOBER.

SA Tape Pointer (ILBDIML0) [FD]

Operation: This subroutine gets the pointer to the physical tape drive associated with the logical unit for a particular tape file.

Linkage:

```
LA    0,DTFPTR cell
L     15,=V(ILBDIML0)
BALR  14,15
```

Output: The current PUB pointer for this device is moved to DTF-8.

SA Position Multiple File Tapes (ILBDMFT0) [FE]

Operation: This subroutine positions an unlabeled or nonstandard labeled tape to the beginning of a desired file. Given a position integer greater than one, the subroutine rewinds and forward-spaces the tape, bypassing all files ahead of the desired one.

Linkage:

```
L      1,DTFPTR
LA     2,Position integer
L      15,=V(ILBDMFT0)
BALR   14,15
```

Output: The tape is positioned.

SA Test Tape File (ILBDMVE0) [FF]

Operation: This subroutine determines whether a multivolume unlabeled tape has reached EOF or EOV and acts accordingly. It sends a message reading, "C126D IS IT EOF?" to the operator. If the operator's answer is yes (Y or y), the subroutine exits to the AT END address; if it is no (N or n), the subroutine executes an FEOV instruction to switch to the next volume, executes a GET instruction to get the first record, and then returns.

Linkage:

```
L      5,A(AT END routine)
L      1,A(DTF)
BALR   15,0
LA     3,12(15)   (See Note)
L      15,16(1)
BAL    14,8(15)
```

For spanned records, a different linkage is required since register 3 is not available.

and the work area address is needed by the subroutine:

```
CNOP   2,4
L      5,A(AT END routine)
BALR   15,0
ST     5,20(15)
ST     0,24(15)
LA     5,20(15) (Register 5 points to the
         2 fullword constants below)
L      15,16(1)
B      8(5)
DS     F (Contains end-of-file address)
DS     F (Contains workarea address)
BAL    14,8(15)
```

Note: This is the same address as that in register 14.

Output: The message, "C126D IS IT EOF" is sent to the operator.

SA STXIT Macro Instruction (ILBDABX0) [FG]

Operation: This subroutine is called during the code generated for OPEN verbs. It issues a STXIT AB macro instruction specifying that an address within the subroutine is to be given control by the system in the event of abnormal termination. The secondary entry point is called if an error occurs on a unit record device, there is a standard error declarative for the device, and STXIT is requested on the CBL card. If the ILBDTC20 subroutine is in the load module, control is passed to it.

Linkage:

```
L      15,=V(ILBDABX0)
BALR   14,15
```

Output: The STXIT AB macro instruction is issued.

SA Reposition Tape ($$BFCMUL) [FH]

Operation: This subroutine resets the PUB pointer for a particular (SYSnnn) device to the same as that saved earlier (by subroutine ILBDIML0). It rotates the LUB/JIB pointers until the current PUB pointer is identical to the saved one.

Linkage:

```
L      0,A(DTF) (See Note)
LA     1,=CL8'$$BFCMUL'
SVC    2
```

Note: The saved PUB pointer is at DTF-8

Output: the LUB and JIB pointers may be changed.

INDEXED SEQUENTIAL ACCESS DATA MANAGEMENT SUBROUTINES

The subroutines described below handle some of the I/O operations for the indexed sequential access method.

ISAM READ and WRITE (ILBDISM0) [GA]

Operation: This subroutine handles all indexed sequential READ and WRITE instructions. It checks for invalid key and input/output errors and branches accordingly to the appropriate procedure.

Linkage:

```
L      1,DTFPTR
L      0,A(Record) BL for Sequential
         READ, REWRITE only
L      15,=V(entry point)
L      5,A(INVKEY or EOF)
BALR   14,15
```

For 'entry point,' substitute one of the following:

```
ILBDISM0 for LOAD or EXTEND (WRITE,
           Sequential)
ILBDISM1 for ADD (WRITE, Random)
ILBDISM2 for Random Retrieval (READ,
           Random)
ILBDISM3 for Random Retrieval (READ,
           Sequential)
ILBDISM4 for Random Update (REWRITE,
           Random)
ILBDISM5 for Sequential Update (REWRITE,
           Sequential)
```

Output: The record is read or written.

ISAM Error Routine (ILBDISE0) [GB]

Operation: This subroutine processes ISAM errors either by setting user error bytes (if any) and branching to a user error routine, or if there is no user error routine, by setting the error code and fetching $$BCOBER. If the exit is to the user routine, register 1 points to the error block. If $$BCOBER is fetched, an appropriate message is printed on SYSLOG and SYSLST by $$BCOBR1. Then, if a dump is not required, control returns to ILBDISE0;

if it is, $$PDUMP is called, provides the dump and returns control to ILBDISE0. If ILBDDBG2 (the STXIT routine) is in the load module, control is transferred to it. If it is not, the job is cancelled.

Linkage:

If this subroutine is called by ILBDISM0:

```
L     2,ERRBLKPTR
L     1,DTFPTR
L     15,=V(ILBDISE0)
BR    15
```

If this subroutine is called by the main line:

```
L     1,DTFPTR
L     15,=V(ILBDISE1)
BR    15
```

Output: User error bytes, if any, are set to reflect the error condition. Register 1 points to the error block for data transfer on input file. The error code and address of DTF-8 (for PDUMP) are forwarded in register 0 when fetching $$BCOBER.

ISAM START (ILBDSTR0) [GC]

Operation: This subroutine, in response to START or START with the KEY EQUAL TO option, issues the $$BSETL macro to initiate sequential retrieval. If the subroutine is called in response to the KEY EQUAL TO option, certain processing occurs prior to the issuance of the $$BSETL macro; after obtaining the address and length of the NOMINAL KEY data-name (KEYARG) from the DTF, this subroutine moves the generic key identifier to the NOMINAL KEY data-name and pads with zeros if the generic key identifier is shorter.

Linkage:

If the subroutine is called in response to START:

```
L     0,DTFPTR
L     15,=V(ILBDSTR1)    (Entry point in
BALR  14,15               ILBDSTR0)
```

If the subroutine is called in response to START with KEY EQUAL TO:

```
L     0,DTFPTR
LA    3,identifier       (Address of
                          identifier
                          which contains
                          key value
                          requested)
LH    5,=H'LENGTH'       (Length of
      OR 'VLC'            identifier)
L     15,=V(ILBDSTR0)
BALR  14,15
```

Output: For START, the file is positioned to the specific key within the file. For START with KEY EQUAL TO, the file is positioned to the beginning of the generic group within the file. The generic key identifier is moved to the NOMINAL KEY data-name and padded with zeros if necessary.

DIRECT-ACCESS DATA MANAGEMENT SUBROUTINES

The subroutines described below handle some of the I/O operations for the direct access method.

DA Close Unit (ILBDCRD0) [HA]

Operation: This subroutine implements a CLOSE UNIT instruction on a DA file which is read sequentially when absolute track (physical) addressing is used.

Linkage:

```
L     1,DTFPTR
L     15,=V(ILBDCRD0)
BALR  14,15
```

Output: The current extent bucket in the Extent Store Area (described under "DA Extent Processor") and the SEEK address are updated to the first extent on the next volume for subroutine ILBDDSR0.

DA Close Unit for Relative Track (ILBDRCR0) [HB]

Operation: This subroutine implements a CLOSE UNIT instruction for relative track addressing on a DA file which is read sequentially.

Linkage:

```
L     1,DTFPTR
L     15,=V(ILBDRCR0)
BALR  14,15
```

Output: The current extent bucket (in the high-order byte of DTF-16; followed by the 3-byte address of the extent table in the DTF) and the SEEK address are updated to the first extent on the next volume for subroutine ILBDRDS0.

DA Extent Processor (ILBDXTN0) [HC]

Operation: When absolute track addressing is used, this subroutine is called to store the extent limit information made available by an OPEN. A maximum of 7 extents can be stored. The Extent Store Area address is in DTF-16.

Linkage: None. The address of this subroutine is in the DTF.

Output: The extent limits are saved. The SEEK address is initialized for ILBDDSR0, and the first byte of the Extent Store Area is initialized to 0.

The Extent Store Area format is as follows:

Byte 0:	Current extent bucket. It is set at CLOSE UNIT time by subroutine ILBDCRD0 and used as an indicator by subroutine ILBDDSR0.
Byte 1:	Used by subroutine ILBDXTN0 to indicate the SYS-number of the applicable unit.
Bytes 2-8:	The lower limit of the first extent, in the form MBBCCHH.
Bytes 9-15:	The upper limit of the first extent, in the form MBBCCHH.
Bytes 16-n:	The lower and upper limits of any remaining extents, in the same form as the first.
Byte n + 1:	X'FF', to indicate the end of the extent store area.

DA Sequential Read (ILBDDSR0) [HD]

Operation: This subroutine reads a DA file sequentially when absolute track addressing is used. It generates a SEEK address from the extent information stored by subroutine ILBDXTN0 and from the IDLOC returned by LIOCS. It utilizes subroutine ILBDIDA0 to increase the SEEK address by one track.

Linkage:

```
L     1,DTFPTR
L     0,A(ACTKEY)  (If actual key
        specified)
SR    0,0 (If actual key not
        specified)
L     15,=V(ILBDDSR0)
L     5,A(EOF)
BALR  14,15
```

Output: The record is read and the track address is updated for the next READ.

DA Sequential READ for Relative Track (ILBDRDS0) [HE]

Operation: This subroutine reads a DA file with relative track addressing sequentially. The relative track address is initialized at OPEN time by the main-line code or, at CLOSE UNIT time, by subroutine ILBDRCR0. The address of the next record, which has been stored in the IDLOC field by the LIOCS module, is stored in the track address field.

Linkage:

```
L     1,DTFPTR
L     0,A(ACTKEY)  (If actual key
        specified)
SR    0,0  (If actual key not
        specified)
L     15,=V(ILBDRDS0)
L     5,A(EOF)
BALR  14,15
```

Output: The record is read and the track address is updated for the next READ.

DA RZERO Record (ILBDFMT0) [HF]

Operation: When absolute track addressing is used, this subroutine writes Record 0 onto each track of a DA output file.

Linkage:

```
L     1,DTFPTR
L     15,=V(ILBDFMT0)
BALR  14,15
```

Output: The RZERO record is written.

DA RZERO for Relative Track (ILBDRFM0) [HG]

Operation: This subroutine writes Record 0 onto each track of a DA output file with relative track addressing.

Linkage:

```
L     1,DTFPTR
L     15,=V(ILBDRFM0)
BALR  14,15
```

Output: The RZERO record is written.

DA Increase SEEK Address (ILBDIDA0) [HH]

Operation: This subroutine increases a SEEK address by one track when absolute addressing is used.

Linkage:

```
L     1,DTFPTR
L     15,=V(ILBDIDA0)
BALR  14,15
```

Output: The SEEK address is increased.

DA READ and WRITE (ILBDDIO0) [HI]

Operation: When absolute track addressing is used, this subroutine reads or writes records on random access DTFDA files in response to READ or WRITE instructions using absolute addressing. It also checks for invalid key and input/output errors and branches, if necessary, to the appropriate procedure.

Linkage:

```
LH    3,RECSIZE (Undefined and spanned
        records only)
AH    3,=H'4' (Spanned records only)
L     0,A(ACTKEY)
L     15,=V(Entry point)
L     5,A(INVKEY)
BALR  14,15
```

For 'entry point', substitute as follows:

ILBDDIO0 for WRITE AFTER or WRITE key (American National Standard and DOS/VS COBOL WRITE/REWRITE)
ILBDDIO1 for READ key and SAVE key READ (American National Standard and DOS/VS COBOL READ)
ILBDDIO2 for READ key
ILBDDIO3 for WRITE key
ILBDDIO4 for WRITE AFTER

Output: The record is read or written.

DA READ and WRITE for Relative Track (ILBDRDI0) [HJ]

Operation: This subroutine reads or writes records an random access DTFDA files in response to READ or WRITE instructions using relative track addressing. It also checks for invalid key and input/output errors and, if necessary, branches to the appropriate procedure.

Linkage:

```
LH    3,RECSIZE
L     1,DTFPTR
L     0,A(ACTKEY)
L     15,=V(entry point)
L     5,A(INVKEY)
BALR  14,15
```

For 'entry point', substitute as follows:

ILBDRDI0, for WRITE AFTER or WRITE KEY (American National Standard and DOS/VS COBOL WRITE/REWRITE)
ILBDRDI1, for READ KEY and SAVE KEY READ (American National Standard and DOS/VS COBOL READ)
ILBDRDI2, for READ KEY
ILBDRDI3, for WRITE KEY
ILBDRDI4, for WRITE AFTER

Output: The record is read or written.

DA Error Routine (ILBDDAE0) [HK]

Operation: This subroutine handles errors on DTFDA files either by setting user error bytes (if any) and branching to a user error routine, or if there is no user error routine, by setting the error code and fetching $$BCOBER. If $$BCOBER is fetched, an appropriate message is printed on SYSLOG and SYSLST. Then, if a dump is not required, control returns to ILBDDAE0; if it is, $$BPDUMP is called. $$BPDUMP provides the dump and returns control to ILBDDAE0. If ILBDDBG2 (the STXIT routine) is in the load module, control is passed to it. If it is not, the job is cancelled.

Linkage:

```
L     2,A(DTF-24)
L     15,=V(ILBDDAE0)
BR    15
```

Output: Register 1 points to the data in the error block when exiting to user if there has been input data transferred. The user error bytes, if any, are set to reflect the error condition.

VSAM DATA MANAGEMENT SUBROUTINES

The subroutines described below are the interface between the IBM DOS/VS COBOL object program and the VSAM system control subroutines.

VSAM Initialization (ILBDINT0) [HL]

Operation: This subroutine issues the GETVIS macro instruction to obtain virtual storage for the VSAM File Control Block (FCB) associated with each VSAM File Information Block (FIB). It initializes the FCB to zeros, sets some initial values, and stores the address of the FCB in the object program's TGT area. It also acquires work space for the VSAM subroutines.

Linkage:

```
L     R15,V(ILBDINT0)
BALR  R14,R15
DC    XL2'DISPL IN TGT OF 1ST FIB CELL'
DC    XL2'NUMBER OF FIB's'
```

Output: Storage is acquired for the VSAM FCB and VSAM work space.

VSAM Open and Close Subroutine (ILBDVOC0) [HM]

Operation: This subroutine handles all VSAM OPEN and CLOSE requests.

For OPEN, the subroutine fills in FCB fields, obtains workspace for the file, constructs three control blocks for each file to be opened (ACB, EXLST, RPL), and fills in fields in these control blocks. It sets up the STATUS KEY and RERUN integer and checks the CLOSE option for LOCK. If opened OUTPUT, it checks the high relative byte address for zero. It then branches to the appropriate VSAM system control subroutine.

For CLOSE, the subroutine issues a FREEVIS for all space used for the file being closed and sets up the STATUS KEY as well as the CLOSE options in the FCB. It tests for RERUN and, if required, takes a checkpoint. It then branches to the appropriate VSAM system control subroutine.

Linkage--For OPEN Request

The following code is generated for each file to be opened:

```
L     R1,FIB-CELL(R13)
ST    R1,SAV3+DISP(R13)
MVC   FOPENOPT(4,R1),
      =XL4'OPEN-OPTIONS'  (See Note 1)
MVC   FUSEERR(4,R1),      If USE...ERROR
      USERRPN(R12)        Declarative
```

The following code is generated last:

```
MVI   SAV3+LASTDISP       Indicate end of
      (R13),X'80          list.
LA    R1,SAV3(R13)
L     R15,=V(ILBDVOC0)
BALR  R14,15
```

Note 1: See FOPENOPT field of the FCB in "Section 3: Data Areas" for bit assignments for each option.

Linkage--For CLOSE Request

The following code is generated for each file to be closed:

```
L     R1,FIB-CELL(R13)
ST    R1,SAV3+DISP(R13)
MVC   FCLOSOPT(4,R1),     (See Note 1)
      =XL4'CLOSE-OPTIONS'
```

The following code is generated last:

```
MVI   SAV3+LASTDISP(R13),X'80'
LA    R1,SAV3(R13)
L     R15,=V(ILBDVOC1)
BALR  R14,R15
```

Note 1: See FCLOSOPT field of the FCB in "Section 3: Data Areas" for bit assignments for each option.

VSAM Action Request Subroutine (ILBDVIO0) [HN]

This subroutine handles all requests for START, READ, REWRITE, WRITE, and DELETE verbs with VSAM files.

Each request is routed to the code handling the particular verb. This code passes the request to VSAM. Upon execution of the request, it checks the return code from VSAM for errors. Depending on the return code and conditions set in the FCB, it returns control to the calling subroutine.

For more specific meanings for each of the STATUS KEY entries, see IBM DOS/VS COBOL Programmer's Guide.

Linkage:

```
MOVE RECORD-AREA,          (If FROM option
     FROM-AREA              specified for
                            WRITE and
                            REWRITE)
L    R4,FIB-CELL(R13)
L    R14,return-GN
MVC  FENDINV(4,R4),        (If INVALID KEY,
     ENDINVGN(R12)          AT END, or AT
                            ECP specified)
```

```
MVI   FRECKEY(R4),          (If KEY clause
      RECORD-KEY-#           specified for
                             READ and START)
LH    R0,=H'RECORD-         (If WRITE or
      LENGTH'                REWRITE speci-
                             fied for fixed
                             length record)
      or
LH    R0,RECORD-VLC(R13)    (If WRITE or
                             REWRITE speci-
                             fied for vari-
                             able length rec-
                             ord)
LA    R0,KEY-LENGTH         (If START speci-
                             fied with key)
L     R15,FCOBRTN(R4)
BALR  R1,R15
DC    XL1'COMMAND-CODE'     (See Note 1)
DC    XL3'OPTIONS'          (See Note 2)
MOVE
L     R5,NEXT-sentence GN   (if INTO option
BR    R5                     for READ)
```

Note 1: Command Codes-
(4=READ,8=WRITE,12=REWRITE,16=START,20=
DELETE)

Note 2: Option bytes have the following bit information:

Byte 0 -

Bit	Meaning
0	Invalid Key
1	At End
2-5	Unused
6	Next
7	Key (For READ or START)

Byte 1 - Search Condition for START

Code	Meaning
X'80'	Greater
X'40'	Equal
X'20'	Not less

Byte 2 - X'80' called from ILBDSRT0

Output: The requested input/output instruction is performed.

ASCII SUPPORT SUBROUTINE

The subroutine described below handles two of the functions necessary for handling files written in ASCII code. Other functions are handled by code in the COBOL program or by subroutine ILBDSPA0.

Separately Signed Numeric Subroutine (ILBDSSN0) [IA]

Operation: This subroutine is called whenever a data-name is involved in an arithmetic operation or in certain move operations and has a TRAILING SEPARATE CHARACTER or LEADING SEPARATE CHARACTER clause in the source program. The subroutine checks the sign for validity. If the sign is not a valid sign,the subroutine issues a message and abnormally terminates the job. The subroutine has two entry points, ILBDSSN0 and ILBDSSN1.

The subroutine is called at entry point ILBDSSN0, when an internal decimal number is to be produced. It places the proper sign in the low-order four bits of the receiving byte.

The subroutine is called at entry point ILBDSSN1, when a separately signed external decimal number is to be produced. It places the proper EBCDIC sign in the receiving byte and replaces the converted sign in the high-order four bits with a X'F'.

Linkage:

```
LA    0,Sign
LA    1,Receiving byte
L     15,=V(ILBDSSN0) or (ILBDSSN1)
BALR  14,15
```

Output: The output of this routine is an internal decimal number, or a separately signed external decimal number.

Three options are available for object-time debugging. These are the statement number option (STATE), the flow trace option (FLOW), and the symbolic debug option (SYMDMP). The subroutines for the first two options provide debugging information at abnormal termination of a program; the subroutines for the third option provide debugging information either at abnormal termination or dynamically during the execution of a program. All of the subroutines are under the control of and are serviced by the Debug Control subroutine (ILBDDBG0). This chapter discusses (1) the Debug Control subroutine (ILBDDBG0), and (2) the subroutines that are called in response to each of the three debug options.

Note: Diagram 6 in "Section 2: Program Organization" illustrates the calling dependencies among these routines.

DEBUG CONTROL SUBROUTINE (ILBDDBG0)

This subroutine is included by the linkage editor whenever the CBL control card for a program contains at least one of the debug options or the CCUNT option. It is a single CSECT, consisting of eight routines and one common area. These are, with their entry points:

- Test routine (ILBDDBG0)
- Print routine (ILBDDBG1)
- STXIT routine (ILBDDBG2)
- TGT Address routine (ILBDDBG3)
- Save Register 14 routine (ILBDDBG4)
- Dynamic Dump routine (ILBDDBG5)
- Range routine (ILBDDBG6)
- Debug common area (ILBDDBG7)
- Close Debug File routine (ILBDDBG8)

The routines are described below. The debug common area is described in "Section 3: Data Areas."

TEST ROUTINE (ILBDDBG0) [JA]: A call is generated to the TEST routine (ILBDDBG0) in INIT 3. This routine tests for the debug options that have been specified by checking bits 4, 5, and 6 of SWITCH in the TGT table, or for the COUNT option specified through bit 20.

The subroutine calls FLOW (ILBDFLW0) for the flow trace (FLCW) option, and loads and branches to SYMINIT (ILBDMP10) for the symbolic dump (SYMDMP) option. These subroutines perform initialization processing for the respective options. The initialization process varies for each option and is discussed below.

The TEST subroutine calls the execution statistics initialization subroutine (ILBDTC00) to begin implementation of the COUNT option.

The TEST routine issues the STXIT macro instruction specifying that the STXIT routine (ILBDDBG2) is to receive control when abnormal termination occurs. It also computes the load point for SYMDMP modules and issues the LOAD macro instruction to load ILBDMP10. ILBDMP10 is then given control so that it can read in and process the SYMDMP control cards.

Diagrams 4 and 5 in "Section 2: Prograi Organization" show the flow of control for the Symbolic Dump (SYMDMP) subroutines. Diagram 6 shows control flow for the Debug Control Subroutine (ILBDDBG0) through five levels.

Linkage:

```
L     15,=V(ILBDDBG0)
BALR  14,15
```

If the COUNT option has been specified, th following is added:

```
DC    H'number-of-count-blocks'
```

Input: Register 13 contains the address o the TGT.

PRINT ROUTINE (ILBDDBG1) [JB]: The PRINT routine is called by each of the subroutines associated with the debugging operations. Its function is to print either the debugging information requested or any error messages about the debug option subroutines themselves.

Linkage:

```
L     15,=V(ILBDDBG1)
BALR  14,15
```

Input:

1. DBG1CODE in the communication area in ILBDDBG0 module. This code indicates to ILBDDBG1 how the output is to be printed.

2. Buffer, containing information to be written on SYSLST.

Output:

1. Register 2 contains the address of the next buffer.

2. A line of output on SYSLST.

STXIT ROUTINE (ILBDDBG2) [JC]: This routine gets control from the system when an abnormal termination has occurred. This routine may also get control from a COBOL library I/O module when a termination type of error is recognized. It traces all COBOL programs in the run unit.

For those programs compiled with the COUNT option, the STXIT routine calls subroutine ILBDTC20 to write execution statistics on SYSLST. For those that are compiled with SYMDMP, STATE, or FLOW options, the subroutine calls the corresponding subroutines to record the requested debugging information.

Therefore, if the interrupted program was itself called by another program in the same load module, the STXIT routine also supervises debugging operations for the calling program if one of the debug options has been specified for that program. It uses data area FIRST-LAST, for this purpose (see "TGT Address routine (ILBDDBG3))". The debugging operations are completed when the highest level calling program which has been compiled with a debug option (SYMDMP, STATE, or FLOW) has been given debug information. Diagram 6 shows the control flow for the STXIT routine through five levels. Diagram 11 in "Section 2: Program Organization" shows the doubleword data-area (FIRST-LAST) which is used to trace the COBOL programs at abnormal termination.

Linkage:

This routine is given control directly from the System at abnormal termination. It returns to the System by issuing an EOF macro instruction.

Input: STXIT save area, containing the PSW and registers 0-15 at the time of abnormal termination.

TGT ADDRESS ROUTINE (ILBDDBG3) [JC]: The TGT Address routine (ILBDDBG3) is called by the COBOL program following the return of control to the COBOL program after a branch outside the current program. The TGT Address routine stores in a fullword (LAST), the address of the current TGT upon return from a called program. This data area is used by the STXIT routine at abnormal termination to trace the calling programs of an interrupted program so that debugging information may be provided for each of them. Diagram 11 in "Section 2: Program Organization" shows the pointer connections between the FIRST-LAST data area and the TGT's of the programs that are link edited together.

Linkage:

```
L     15,=V(ILBDDBG3)
BALR  14,15
```

Input: Register 13 contains the address of the current TGT.

SAVE REGISTER 14 ROUTINE (ILBDDBG4) [JC]: The Save Register 14 routine (ILBDDBG4) is called by the COBOL program just before any instruction which passes control outside the COBOL program. It stores the address of this instruction. If an abnormal termination occurs and the PSW points outside the current COBOL program, it is this address and not the PSW address that is used to determine the number of the source statement that caused the program error.

Linkage:

```
L     15,=V(ILBDDBG4)
BALR  14,15
```

Input: Register 14 contains the address of the instruction that transfers control outside the current program.

DYNAMIC DUMP ROUTINE (ILBDDBG5)[JD]: The function of this routine is to signal SYMDMP that a dynamic dump is to be given. Upon return from SYMDMP, register 10 contains the address of the instruction that was overlaid with the BALR instruction that called ILBDDBG5. (See "Program Modification" under "Symbolic Dump (SYMDMP) Subroutine".) The overlaid instruction is then executed and control is returned to the COBOL program.

Linkage:

```
L     15,=V(ILBDDBG5)
BALR  14,15
```

Input:

1. Register 3 contains the TGT address.

2. Upon return from SYMDMP, register 10 points to the instruction that was

overlayed with the BAIR instruction that invoked ILBDDBG5.

RANGE ROUTINE (ILBDDBG6)[JE]: This routine is called from the GOBACK code. Its function is to indicate that a branch (GOBACK) to a program that is higher than the highest COBOL program compiled with SYMDMP, STATE, or FLOW has been taken. Such a program is outside the range of the Debug Control Subroutine. That is, an abnormal termination in such a program will be intercepted by the STXIT routine (ILBDDBG2). The STXIT routine's only function in this case is to issue the EOJ macro instruction.

Linkage:

```
L     15,=V(ILBDDBG6)
BALR  14,15
```

Input: Register 13 contains the current TGT address.

CLOSE DEBUG FILE ROUTINE (ILBDDBG8) [JE]: This routine is called by ILBDTC20 to close the debug file when object-time execution statistics have been written, but there are no debugging options specified.

Called by: ILBDTC20

Linkage:

```
L     15,=V(ILBDDBG8)
BALR  14,15
```

Calls: $$BCLOSE

Input: None

Output: DTF closed

SUBROUTINES FOR THE DEBUG OPTIONS (STATE, FLOW, SYMDMP)

The statement number (STATE) and flow trace (FLOW) options each require a separate subroutine. They are the Statement Number subroutine (ILBDSTN0) and the Flow Trace subroutine (ILBDFLW0). The symbolic dump option (SYMDMP) requires a subroutine made up of 13 modules or phases, whose entry point from the Debug Control Subroutine is ILBDMP10.

The debugging information provided by the Statement Number subroutine (ILBDSTN0) consists of the number of the COBOL statement and the number of the verb within the statement being executed when abnormal termination occurred. The debugging information provided by the Flow Trace subroutine (ILBDFLW0) consists of the source card numbers that represent the COBOL procedures executed before abnormal termination occurred.

When a dynamic dump is requested, the Symbolic Dump subroutines provide a formatted symbolic dump of specified areas of the Data Division just prior to the execution of each of the specified COBOL statements. When SYMDMP is specified, the symbolic dump subroutines provide at abnormal termination a formatted symbolic dump consisting of the following parts:

1. an abnormal termination message identifying the source statement causing the error,
2. selected areas in the TGT, and
3. all the data items from the Data Division.

STATEMENT NUMBER SUBROUTINE (ILBDSTN0)[JF]

Operation: When the subroutine receives control from the STXIT routine (ILBDDBG2) at abnormal termination, it provides the number of the COBOL statement and the number of the verb within the statement that was being executed when abnormal termination occurred. If abnormal termination occurs during execution of an instruction outside of the COBOL program, the statement number that is provided is that of the last COBOL statement executed. The subroutine uses the information stored by the Save Register 14 routine (ILBDDBG4) for this purpose. The subroutine calls the PRINT routine (ILBDDBG1) to write the debugging information on SYSLST.

This subroutine is called from the STXIT routine (ILBDDBG2) using the following sequence:

```
L     15,=V(ILBDSTN0)
BALR  14,15
```

Input: Register 13 points to the communication area in the Debug Control Subroutine (ILBDDBG0) from which the address of the current TGT and other information can be obtained.

Output: Statement number message on SYSLST.

FLOW TRACE SUBROUTINE (ILBDFLW0)[JG]

Operation: This subroutine is entered at entry point ILBDFLW0 by the TEST routine (ILBDDBG0) for initialization and at entry point ILBDFLW2 by the STXIT routine (ILBDDBG2) at abnormal termination. It is also called at entry point ILBDFLW1 by compiled code upon encountering each COBOL PN. Calls are not generated for dummy PNs. When the subroutine is called for initialization at entry point ILBDFLW0, it obtains the address of the area allocated for the flow trace table. The number of traces specified by the user is a factor in determining the table size at compile time. This table is at a fixed displacement in the TGT of the COBOL program. After initialization each time that the subroutine receives control from the COBOL program, it inserts the executing program's 8-character Program Identification as well as the card number of the current COBOL Procedure into the next available position in the table. The address of the next available position in the table is stored at location NXTAVL. Pointers for physical end (PEND) and logical beginning (LBEG), which indicates table wraparound, are also employed and are located just before the 80-byte PROGRAM-ID area of the table.

When the end of the table is reached, location NXTAVL points once again to the beginning of the table; and subsequent entries into the table overlay previous entries. The procedure is repeated until the end of the main COBOL program or until abnormal termination. If abnormal termination occurs, the subroutine receives control from the STXIT routine; and it calls the PRINT routine (ILBDDBG1) to print each entry of the table beginning with the earliest entry.

Linkage:

From the TEST routine (ILBDDBG0):

```
L     15,=V(ILBDFLW0)
BALR  14,15
```

From compiled code:

```
L     15,=V(ILBDFLW1)
BALR  14,15
```

From the STXIT routine (ILBDDBG2)

```
L     15,=V(ILBDFLW2)
BALR  14,15
```

SYMBOLIC DUMP (SYMDMP) SUBROUTINE [JH]

The symbolic dump subroutine, referred to mnemonically as SYMDMP, consists of 13 load modules or phases. Of these, two (ILBDMP01 and ILBDMP02) service I/O requests for the remaining modules; five (ILBDMP10 through ILBDMP14) constitute what is here termed Pass 1; and six (ILBDMP20 through ILBDMP25) constitute Pass 2. The first digit in the load module name identifies the pass, the second digit the module within the pass.

The 13 modules of SYMDMP are arranged in an overlay structure under the control of SYMDMP itself, with the modules of Pass 2 overlaying those of Pass 1 after initialization is complete. (See Diagrams 3 and 7 in "Section 2: Program Organization."

PASS1: The function of Pass 1 is to scan control cards and translate them into tables for the use of Pass 2. Pass 1 is entered during INIT3 before execution of a program compiled with the SYMDMP option or, when several programs compiled with the SYMDMP option have been link edited together, before execution of the first program. Pass 1 is entered only once per run unit.

PASS 2: The function of Pass 2 is to produce the output requested by the control cards. After Pass 2 has overlaid Pass 1, it is present during the entire run and may be entered many times. Pass 2 may be entered at the following times:

- During INIT3 before execution of each program
- Before each entry to any independent program segment
- At abnormal termination
- Each time a dynamic dump request is to be satisfied.

COMMON DATA AREA: The SYMDMP modules communicate with one another by means of a block of cells initialized by Pass 1 and kept intact (not overlaid) when control is turned over to Pass 2. Register 12 is reserved in all modules as the base register for this area. The first portion of the common data area contains four standard register save areas, and data needed by both passes. The data needed by both passes include: addresses of tables; addresses of buffers; cells used by the two I/O modules; information about storage allocation; etc. The second portion contains data used to communicate between the modules of either pass, but not between the passes. This includes: load addresses

for the modules of the pass; addresses of the table entries currently being processed; parameters for subroutines; etc.

OBJECT-TIME TABLES: Three tables are built in Pass 1 to facilitate communication among the modules of SYMDMP. These are:

- The PCONTROL table, which contains one entry for each program in a run unit; it preserves information about the program's debug file, the program-control card options, the other tables, and critical locations in the COBOL program itself.
- The DYNAMTAB table, which contains one entry for each dynamic dump request; it preserves card/verb number, virtual storage location and machine instruction corresponding to the request, and pointers which are used to locate on the debug file the data-names specified.
- The DATADIR table, which is an index to the blocks of the debug file that are needed for dynamic dumping.

For detailed descriptions of the PCONTROL, DYNAMTAB, and DATADIR tables, see "Section 3: Data Areas."

INPUT: SYMDMP receives information from four sources external to itself:

- The communication area of ILBDDBG0, containing, in particular, in LAST the address of the COBOL program's TGT.
- The COBOL program's TGT and INIT1 cells, its instructions, and its Data Division.
- The control cards on SYSIPT.
- The debug file built by the COBOL compiler.

Control cards: There are two types of control cards, program-control and line-control.

Each program for which any SYMDMP service is requested must be identified by a program-control card. PROGRAM-ID, debug file information, the ENTRY option, and the HEX option for abnormal termination dumps are specified on this card. Each dynamic dump request is identified by a line-control card. Card/verb number, the Data Division items to be dumped, and the ON and local HEX options are specified on this card.

The SYMDMP control cards are described in detail in the publication IBM DOS/VS COBOL Compiler and Library Programmer's Guide, Order No. SC28-6478.

Debug File: When the SYMDMP option is specified on the CBL card, Phases 25 and 65 of the compiler create a file for use by SYMDMP at object time. The file contains information about the items of the Data Division and about the location of the machine instructions corresponding to each Procedure Division source statement.

The program-control card identifies the debug file for SYMDMP at object time by specifying device type (MT or SD), logical unit number, and, for a disk file, filename. These three items of information are saved in the PCONTROL table. Device type is used to determine which of the two I/O modules to invoke; logical unit number and file-name are stored in the DTF before the file is opened. Thus, the single DTF contained in each of the I/O modules can serve any number of files used one at a time.

The format and contents of the debug file are described in "Data Areas" under "Program Organization". Diagrams 8, 9, and 10 in "Section 2: Program Organization" show the relations between the debug file and the object-time tables.

OUTPUT: SYMDMP generates the following types of information:

- Output on SYSLST consisting of: a copy of all control cards; diagnostic messages; dynamic dumps; the abnormal termination statement number message; the complete abnormal termination dump
- Modifications to the COBOL program in virtual storage if dynamic dumping is requested for the program

Program modification: The mechanism by which SYMDMP intervenes in the COBOL program to produce a dynamic dump is as follows:

Pass 1 searches the Procedure Division tables of the debug file for the specified card number. It stores, in the DYNAMTAB entry for the card, the address (relative to the beginning of the Procedure Division or of the transient area) of the corresponding instruction.

Pass 2, when entered during INIT3, relocates this address to its true current value and saves the instruction itself in the DYNAMTAB entry. The first two bytes of the instruction in virtual storage are then replaced with BALR 0,12, that is, a branch to the PGT. Since, in a program compiled with the SYMDMP option, the first cells of the PGT contain a call to ILBDDBG5, the

effect is to invoke SYMDMP each time control flows through the modified instruction.

After it has issued the requested dumps, SYMDMP returns to ILBDDBG5 the address of the DYNAMTAB cell which contains the saved original instruction. This instruction is executed in ILBDDBG5 before control is returned to the following instruction in the program. (Note that when abnormal termination occurs, SYMDMP restores the original instruction to the program so that, if the user obtains a system dump, the dump will reflect the COBOL program as it was compiled.)

LINKAGE TO SYMDMP:

```
L     15,=A(ILBDMP10) (See Note A.)
BALR  14,15
DC    H'n' (See Note B.)
```

Note A: the address is computed by ILBDDBG0 before the first call to SYMDMP.

Note B:
'n' = 0 in a call for initialization from ILBDDBG0
= 4 in a call for dynamic dumps from ILBDDBG5
= 8 in a call for abnormal termination dumps from ILBDDBG2

Processing (Sequence of Events)

The sequence of events when SYMDMP services are requested for a run unit is, in general, as follows:

- Initialization for the first COBOL program in a run unit
- Initialization for all other COBOL programs in a run unit
- Initialization for independent program segments
- Processing for dynamic dump requests
- Processing for abnormal termination dumps

The load names, mnemonic names, and functions of the individual SYDMP modules are as follows:

1. I/O modules:

   ILBDMP01 (IODISK) - I/O operations for a debug file on disk.

   ILBDMP02 (IOTAPE) - I/O operations for a debug file on tape.

2. Pass 1 modules:

   ILBDMP10 (SYMINIT) - initialization and Pass 1 control.

   ILBDMP11 (SCANP) - program-control card scan.

   ILBDMP12 (SCAND) - line-control card scan.

   ILBDMP13 (FINDNAMS) - resolution of identifiers.

   ILBDMP14 (FINDLOCS) - resolution of card/verb numbers.

3. Pass 2 modules:

   ILBDMP20 (SYMCNTRL) - Pass 2 control.

   ILBDMP21 (SEGINIT) - program and segment initialization.

   ILBDMP22 (DMPCNTRL) - control for the two dump modules.

   ILBDMP23 (DUMP1) - group and elementary item dump.

   ILBDMP24 (DUMP2) - FD, SD, RD, VSAM FD, and TGT dump

   ILBDMP25 (SYMSTATE) - abnormal termination statement number processing.

The overlay structure and the hierarchy of loading responsibility are detailed in Diagrams 3 and 7 in "Section 2: Program Organization." The flow of control among the modules of Pass 1 and Pass 2 is shown in Diagrams 4 and 5, respectively. The operation of the individual modules is summarized in "Processing (Routines)" in this chapter.

INITIALIZATION - FIRST COBOL PROGRAM:
During INIT3 of the first program encountered with the SYMDMP option, ILBDDBG0 loads and calls ILBDMP10 (SYMINIT).

1. SYMINIT initializes the common data area and reads the first program-control card.

2. SYMINIT loads and calls SCANP.

3. SCANP builds the PCONTROL table, reads the next card, and returns to SYMINIT.

4. If the card starts with a number (line-control card), SYMINIT loads and calls SCAND; otherwise, SYMINIT skips to step 11 below.

5. SCAND builds the DYNAMTAB table; collects data-names in the QUALNAMS area for the batch search of the debug file; reads the next card and, if it starts with a number, repeats the process.

6. SCAND loads and calls FINDNAMS, overlaying itself.

7. FINDNAMS searches the debug file for names collected in the QUALNAMS are and fills in identifier information in the DYNAMTAB table; FINDNAMS then loads and returns to SCAND, overlaying itself.

8. SCAND enters DYNAMTAB and DATADIR pointers in the PCONTROL table, and returns to SYMINIT.

9. SYMINIT loads and calls FINDLOCS.

10. FINDLOCS searches the debug file for addresses corresponding to card/verb numbers and enters these in the DYNAMTAB table, FINDLOCS then returns to SYMINIT.

11. If end-of-file has not bee reached on SYSIPT, SYMINIT returns to step 2 above.

12. At end-of-file, SYMINIT calculates the total size of SYMDMP for the rest of the run unit and stores this value in the ILBDDBG0 cell SYMSIZE for use by the SORT subroutine; SYMINIT also stores information in the common data area for use by the Pass 2 space allocation routines.

13. SYMINIT loads ILBDMP20 overlaying itself and transfers to Pass 2; ILBDMP20 continues normal initialization processing. (See Initialization - All Other COBOL Programs.")

INITIALIZATION - ALL OTHER CCBOL PROGRAMS: During INIT3 of all COBOL programs after the first,, ILBDDBG0 calls SYMDMP at its original address, which is now occupied by ILBDMP20 (SYMCNTRL).

1. SYMCNTRL analyzes the calling parameter and determines that it has been called for initialization.

2. SYMCNTRL loads and calls SEGINIT.

3. SEGINIT, by analyzing PROGRAM-ID, determines that a fresh program is being entered.

4. SEGINIT stores ACURPC (pointer to the current PCONTROL entry) and frequently referenced addresses in COBOL program; SEGINIT also saves the root segment priority of zero.

5. If this is the first time that SEGINIT has been entered (that is, SEGINIT has been entered from SYMINIT), and the DYNAMTAB table exists for any program in the entire run unit, SEGINIT computes the load addresses for DUMP1/DUMP2, IODISK/IOTAPE, and the debug file buffers.

6. If there is no DYNAMTAB table for the current program, SEGINIT skips to step 8 below.

7. SEGINIT loads and calls IODISK or IOTAPE to open the debug file; relocates addresses in the PCONTROL and DYNAMTAB tables; saves the original instructions and modifies them in virtual storage to effect calls to SYMDMP for dynamic dumping.

8. SEGINIT returns to SYMCNTRL.

9. SYMCNTRL returns to ILBDDBG0.

INITIALIZATION - INDEPENDENT PROGRAM SEGMENT: Before entry to an independent program segment, ILBDDBG0 calls SYMDMP at ILBDMP20 (SYMCNTRL).

1. SYMCNTRL analyzes the calling parameter and determines that it has been called for initialization.

2. SYMCNTRL loads and calls SEGINIT.

3. SEGINIT, by analyzing the PROGRAM-ID, determines that the program is the same program as at the previous entry.

4. SEGINIT compares the priority in the TGT with the saved priority; if they are equal, SEGINIT skips to step 7 below.

5. SEGINIT saves the new priority; then, if there is no DYNAMTAB table for the program, SEGINIT skips to step 7 below.

6. SEGINIT saves and modifies instructions in the current independent segment to effect calls to SYMDMP for dynamic dumps.

7. SEGINIT returns to SYMCNTRL.

8. SYMCNTRL returns to ILBDDBG0.

DYNAMIC DUMP REQUEST: ILBDDBG5, called through the program modifications made by SYMDMP (see step 7 under "Initialization - All Other COBOL Programs," step 6 under "Initialization - Independent Program Segment" and "Program Modification" under "Output" above) , calls SYMDMP at ILBDMP20 (SYMCNTRL).

1. SYMCNTRL analyzes the calling parameter and determines that it has been called for dynamic dumps.

2. SYMCNTRL loads and calls DMPCNTRL.

3. DMPCNTRL searches the DYNAMTAB table for all entries with current priority and address fields which match the value of register 0 in the COBOL program; stores the instruction address for ILBDDBG0; updates and analyzes ON counters (if any) for the entries to determine if a dump is required at this execution of the COBOL statement specified on the line-control card. If no dump is required, DMPCNTRL skips to step 9 below; otherwise, DMPCNTRL gets the first (or only) active DYNAMTAB entry for the current request.

4. DMPCNTRL determines from the DYNAMTAB entry the limits of the dump requested; and gets the dump's starting item from the DATATAB table on the debug file.

5. DMPCNTRL loads and calls DUMP1 if the item is a group or elementary item; otherwise, DMPCNTRL loads and calls DUMP2.

6. DUMP1 analyzes the item's attributes which are contained in the DATATAB entry and issues a formatted dump of its contents in virtual storage; gets the next DATATAB entry. If it is beyond the limits of the requested dump, DUMP1 returns to DMPCNTRL; if itis a group or elementary item, DUMP1 repeats the process described above; if it is other than a group or elementary item, DUMP1 requests DMPCNTRL to load and transfer control to DUMP2 to process the item.

   Similarly, DUMP2 dumps information about FD, RD, SD, or index items; and gets the next DATATAB entry. If it is beyond the limits of the requested dump, DUMP2 returns to DMPCNTRL; if it is a group or elementary item, DUMP2 requests DMPCNTRL to load and transfer control to DUMP1.

7. When DUMP1 or DUMP2 returns after satisfying a dump request, DMPCNTRL examines the current DYNAMTAB entry; if it specifies further identifiers for the same card/verb number, DMPCNTRL returns to step 4 above.

8. DMPCNTRL continues the search of the DYNAMTAB table for further entries of equal address and priority; when it finds any such entries, it returns to step 4 above.

9. DMPCNTRL returns to SYMCNTRL.

10. SYMCNTRL returns to ILBDDBG5.

ABNORMAL TERMINATION: ILBDDBG2 calls SYMDMP at entry point ILBDMP20 (SYMCNTRL) to produce abnormal termination dumps for the abnormally terminating program, and, on subsequent calls, for all other SYMDMP-compiled programs encountered in its backward chain to the main COBOL program.

1. SYMCNTRL analyzes the calling parameter and determines that it has been called for abnormal termination dumps.

2. The BOMB switch is turned on.

3. SYMCNTRL loads and calls SEGINIT.

4. SEGINIT, finding BOMB on, performs special abnormal termination processing: examines all DYNAMTAB entries in the run unit and restores the modified instructions to their original state; if the run unit included no dynamic dumping requests, searches all PCONTROL entries for a record of Procedure Divisions large enough to be overlaid by as yet unused SYMDMP modules (DUMP1/DUMP2, IODISK/IOTAPE, and debug file buffers); may also use SORT and DISPLAY subroutines if present; as a last resort may use space remaining between end of tables and end of partition.

5. SEGINIT loads and calls IODISK or IOTAPE to open (or "rewind") the debug file; and relocates addresses in the PCONTROL table if the entry has never been used.

6. SEGINIT returns to SYMCNTRL.

7. SYMCNTRL loads and calls SYMSTATE.

8. If the STATEOUT switch is on, SYMSTATE skips to step 9 below, since the statement number message is only produced for an abnormally terminating program; otherwise, SYMSTATE turns on STATEOUT; gets the address in

ILBDDBG0's STXIT program status word (PSW), or, if this is not within the program's limits, gets the contents of register 14, which were saved by ILBDDBG4; uses this address to search Procedure Division tables of the debug file; identifies the most closely matching card/verb number and issues the statement number message.

9. SYMSTATE returns to SYMCNTRL.

10. SYMCNTRL loads and calls DMPCNTRL.

11. DMPCNTRL, finding BOMB on, sets the dump limit at the last entry in the DATATAB table; turns ALLSW on; and gets the first entry in the DATATAB table.

12. DMPCNTRL loads and calls DUMP2.

13. DUMP2 dumps the TGT and returns to DMPCNTRL.

14. DMPCNTRL loads and calls either DUMP1 or DUMP2 depending on the attributes of the initial DATATAB item (see step 5 under "Dynamic Dump Request").

15. DUMP1 and DUMP2 jointly dump the virtual storage contents of all DATATAB items (see step 6 under "Dynamic Dump Request").

16. DUMP1 returns to DMPCNTRL after dumping the final Data Divisiion entry in the DATATAB table.

17. Since ALLSW is on, indicating that the entire Data Division has been dumped, there can be no further dump request to fill and DMCNTRL returns to SYMCNTRL.

18. SYMCNTRL returns to IILBDDBG2.

Processing (Routines)

IODISK (ILBDMP01)[JI]

Operation: Contains DTFSD, SDMOD, and routines to open, close, read, read and note, point and read, for a debug file on disk.

Linkage:

```
L     15,AIOMOD
LA    1,='ILBDMP01'
LOAD  (1),(15)
BALR  14,15
DC    H'nn' (See note.)
```

Note: 'nn' = 00 to open
04 to read
08 to point before reading
12 to close

Output: Address of current debug file buffer is returned in register 3 and in ADBGBUF. If note was requested, block identification is returned in NOTEADR.

Calling Information: Called by the SCAND, FINDLOCS, and FINDNAMS subroutines in Pass 1, and by the SEGINIT, DMPCNTRL, and SYMSTATE subroutines in Pass 2. It overlays IOTAPE.

IOTAPE (ILBDMP02)[JI]

Operation: Identical with IODISK (ILBDMP01) except that it contains DTFMT and MTMOD for a debug file on tape.

Linkage: Identical with IODISK (ILBDMP01) except that the loadname is 'ILBDMP02'.

Output: See IODISK (ILBDMP01).

Calling Information: See IODISK (ILBDMP01).

SYMINIT (IILBDMP10)[JJ-JK]

Operation: Controls Pass 1 operations; contains 3 common subroutines (CALLFIND, ERROR, and READIPT) for Pass 1 modules.

Linkage:

```
L     15,=A(ILBDMP10)
BALR  14,15
DC    H'00'
```

Output: Table addresses and virtual storage limits are passed in common data area to Pass 2. SYMSIZE cell is set in ILBDDBG0 for use by the SORT subroutine.

Calling Information: Called by ILBDDBG0 during INIT3 of the first program compiled with the SYMDMP option. It is overlaid by SYMCNTRL (ILBDMP20) after completion of Pass 1.

CALLFIND (COMMON PASS 1 SUBROUTINE CONTAINED IN SYMINIT)

Operation: Effects linkage between SCAND and FINDNAMS.

Linkage:

```
L     15,ACALLFND
BALR  14,15
```

Output: None.

Calling Information: Called by SCAND when the DYNAMTAB table is complete.

ERROR (COMMON PASS 1 SUBROUTINE CONTAINED IN SYMINIT)

Operation: Issues Pass 1 error messages.

Linkage:

```
MVI   ERR,message-number
L     15,AERROR
BALR  14,15
```

Output: Error message on SYSLST.

Calling Information: Called by SYMINIT, SCANP, SCAND, FINDNAMS, and FINDLOCS.

READIPT (COMMON PASS 1 SUBROUTINE CONTAINED IN SYMINIT)

Operation: Reads and calls ILBDDBG1 to list control card on SYSIPT; scans card.

Linkage:

```
L     15,AREADIPT
BALR  14,15
```

Output:

Current input card in INBUF;
AELM, address of start of element;
COL, card column of start of next element;
LEN, length of element;
EOCSW, on if no more elements on card;
EOFSW, on if end-of-file found;
NUMSW, on if element is number;
PARENSW, on if element starts with left parenthesis.

Calling Information: Called by SYMINIT, SCANP, and SCAND.

SCANP (ILBDMP11)[JL]

Operation: Calls the READIPT subroutine of SYMINIT to scan program-control card; builds the PCONTROL table entry.

Linkage:

```
L     0,ASCANP
L     1,='ILBDMP11'
LOAD  (1),(0)
BALR  14,1
```

Output: PCONTROL entry and its pointer ACURPC, NXTBYTE, free area pointer, updated to byte following this entry.

Calling Information: Called by SYMINIT when program-control card has been found. Overlays QUAINAMS area used by SCAND and FINDNAMS.

SCAND (ILBDMP12)[JM]

Operation: Calls the READIPT subroutine of SYMINIT to scan line-control card; reads the next card and scans until it comes to a card which does not start with a card-number. Builds a DYNAMTAB entry for each line-control card. Collects data-names specified in QUALNAMS area. Reads in the first block of the debug file.

Linkage:

```
L     0,ASCAND
L     1,='ILBDMP12'
LOAD  (1),(0)
BALR  14,1
```

Output: DYNAMTAB table with fields to be completed by FINDNAMS and FINDLOCS. QUALNAMS area containing all names requested on line-control cards. Pointers to the DYNAMTAB and DATADIR tables in the PCONTROL table. NXTBYTE cell updated to byte following last DYNAMTAB entry.

Calling Information: Called by SYMINIT when card starting with a number is found by SCANP. Overlays FINDNAMS and FINDLOCS.

FINDNAMS (ILBDMP13)[JN]

Operation: Searches the DATATAB table on the debug file for identifiers collected in the QUALNAMS area. Builds the DATADIR table containing block identification for each distinct DATATAB block required. Enters the table locators for identifiers in the DYNAMTAB table.

Linkage:

```
L     0,ASCAND
L     1,='ILBDMP13'
LOAD  (1),(0)
BALR  14,1
```

Output: Locators in the DYNAMTAB table, which permit Pass 2 to point directly, without search, to the requested data-names in the debug file. NXTBYTE cell updated to byte following last DATADIR entry.

Calling Information: Called by SCAND via CALLFIND subroutine in SYMINIT when last line-control card for program has been scanned. Overlays SCAND and FINDLOCS.

FINDLOCS (ILBDMP14)[JO]

Operation: Searches the PRCCTAB table on the debug file for the card/verb numbers specified on line-control cards. Enters corresponding relative addresses in the DYNAMTAB table.

Linkage:

```
L      0,SCAND
L      1,='ILBDMP14'
LOAD   (1),(0)
BALR   14,1
```

Output: Priority and relative address fields in the DYNAMTAB table.

Calling Information: Called by SYMINIT when SCAND returns to it with DTABOK switch on. Overlays SCAND and FINDNAMS.

SYMCNTRL (ILBDMP20)[JP-JQ]

Operation: Controls Pass 2 processing. Contains 1 common subroutine (HEXDUMP) for Pass 2 modules.

Linkage:

```
L      15,=A(ILBDMP10)
BALR   14,15
DC     H'n' (See note.)
```

Note: 'n' = 0 for initialization
4 for dynamic dump
8 for abnormal termination

Output: BOMB switch is turned on in the event of an abnormal termination. This switch is checked by SEGINIT and DMPCNTRL.

Calling Information: Called by ILBDDBG0, ILBDDBG5, and ILBDDBG2. Overlays SYMINIT.

HEXDUMP (COMMON PASS 2 SUBROUTINE CONTAINED IN SYMCNTRL).

Operation: Calls ILBDDBG1 to print hexadecimal dumps.

Linkage:

```
L      15,AHEXDUMP
BALR   14,15
```

Note: Caller places address in ADTODUMP and length in LENTODMP; places desired starting column for address in ADCOL and desired starting column for contents in CORECOL. If address is to be printed, caller turns on PRINTLOC switch.

Output: Hexadecimal dump on SYSLST.

Calling Information: Called by DUMP1 and DUMP2.

SEGINIT (ILBDMP21)[JR]

Operation: Opens the debug file; reads the OBODOTAB table into virtual storage; relocates table addresses; initializes virtual storage for dynamic dumping; performs space allocation at abnormal termination.

Linkage:

```
L      2,ASEGINIT
LA     1,='ILBDMP21'
LOAD   (1),(2)
BALR   14,2
```

Output: Program modifications for dynamic dump calls. The pointer contained in ACURPC is updated to the current PCONTROL entry. LASTSEG is updated to contain the current priority.

Calling Information: Called by SYMCNTRL for initialization and in the event of an abnormal termination. Overlays DMPCNTRL and SYSMSTATE.

DMPCNTRL (ILBDMP22)[JS-JT]

Operation: Controls dumping, identifies current dynamic request in the DYNAMTAB table, and provides service and control for DUMP1 and DUMP2. Contains 2 subroutines (CALLD1D2 and NXTENTRY) common to DUMP1 and DUMP2.

Linkage:

```
L      2,ASEGINIT
LA     1,='ILBDMP22'
LOAD   (1),(2)
BALR   14,2
```

Output: Heading line on SYSLST, before a dynamic dump, to identify card/verb number of request. Cells and switches filled in by NXTENTRY subroutine.

Calling Information: Called by SYMCNTRL at each dynamic request and after SYMSTATE at abnormal termination. Overlays SEGINIT and SYMSTATE.

CALLD1D2 (COMMON PASS 2 SUBROUTINE CONTAINED IN DMPCNTRL)

Operation: Serves as linkage between DUMP1 and DUMP2. Loads whichever of the two is not in virtual storage when it is entered and passes control to it.

Linkage:

```
L     15,ACALLD
BR    15
```

Output: None.

Calling Information: Called by DUMP1 and DUMP2.

NXTENTRY (COMMON PASS 2 SUBROUTINE CONTAINED IN DMPCNTRL)

Operation: Gets and analyzes the next DATATAB entry on the debug file.

Linkage:

```
L     15,ANXTNTRY
BALR  14,15
```

Output: Address of the current DATATAB entry is returned in register 3 and ADATNAME; address of its attributes field is returned in ADATTR; LEV, MAJ, MIN, and other fields are also set.

Calling Information: Called by DMPCNTRL to get the first item of a dump, called by DUMP1 and DUMP2 to get subsequent items.

DUMP1 (ILBDMP23) [JU]

Operation: Formats the contents of group and elementary items; calls ILBDDBG1 to print dumps.

Linkage:

```
L     15,ADUMP1
LA    1,='ILBDMP2'
LOAD  (1),(15)
BALR  14,15
```

Output: The following is written on SYSLST:

For group items: name, level, and card-number. Hexadecimal dump as required.

For elementary items: name, level, card number, location in virtual storage, type code (for example, B for "binary," P for "packed decimal," etc.). Contents of alphabetic and alphanumeric fields in normal print characters. Contents of numeric fields in scaled decimal form.

Every occurrence of each subscripted elementary item is dumped, preceded on the line by its subscripts. Every collection of subscripted elementary items belonging to a variable-length group is preceded by the name(s) and current value of the applicable object(s) of the OCCURS...DEPENDING ON clause.

Calling Information: Called by DMPCNTRL and DUMP2. Overlays DUMP2.

DUMP2 (ILBDMP24) [JV]

Operation: Formats the contents of FD's, SD's, RD's, index-names, and fields of the TGT. Calls ILBDDBG1 to print dumps.

Linkage:

```
L     15,ADUMP1
LA    1,='ILBDMP24'
LOAD  (1),(15)
BALR  14,15
```

Output: The following is written on SYSLST:

TGT fields in hexadecimal format.

For an SD: name, type, and card-number.

For an index-name: name, type, and contents converted to decimal.

For an RD: name, type, card-number, and contents of PAGE-COUNTER and LINE-COUNTER, if present (Note: Report line is printed by DUMP1.)

For an FD: name, type, card-number, and DTF information including contents of DTF in hexadecimal format.

For a VSAM FD: whether the file is open or closed, file organization, access method, the file status key, and the last I/O statement.

Calling Information: Called by DMPCNTRL and DUMP2. Overlays DUMP2.

SYMSTATE (ILBDMP25)[JW]

Operation: Calls ILBDDBG1 to issue statement number message in the event of abnormal termination. Calls the FLOW subroutine (ILBDFLW0), if FLOW is specified, before the first Data Division dump is issued.

Linkage:

```
L     2,ASEGINIT
LA    1,='ILBDMP25'
LOAD  (1),(2)
BALR  14,2
```

Output: Statement number message on SYSLST. STATEOUT switch is set on.

Calling Information: Called by SYMCNTRL after SEGINIT in the event of abnormal termination. Overlays SEGINIT and DMPCNTRL.

SRCHPUBS (ILBDMP04)[JX]

Operation: Searches the PUB table for the device type and then completes the SYS005 DTF by entering the device type, track capacity, and upper head limit.

Linkage:

```
L     R0,ADBGBUF
LA    R1,'ILBDMP04'
LOAD  (1),(0)
LA    R0,ERREXIT
BALR  R10,R1
```

Output: Three bytes beginning at DTF + X'1D' are filled in; the first byte contains the device type and the next two bytes contain the device-type track capacity. DTF + X'27' contains the maximum head limit for a cylinder of that device.

Calling Information: Called by IODISK for each request to open SYS005.

USE-FOR-DEBUGGING Subroutine (ILBDBUG0) [JY]

Operation: ILBDBUG0 is called to handle invocations of USE-FOR-DEBUGGING declaratives, including the setting up of the debug item.

Linkages

```
L    15,V(ILBDBUG0)
BALR 14,15

Branch bypass--GN (4,6, or 8 bytes)

DC   X'FF'
DC   XL2'card number of this verb'

(following fields repeated for each
declarative invocation)
DC   X'description of DEBUG object'
DC   XL3'displ. of DBG-NM literal'
DC   XL2'length of DBG-NM literal'
DC   XL3'displ. of USDBG PN cell'

(optional fields)
DC   XL3'displ. of base for DBG-CONTT'
DC   XL2'displ. from the base'
DC   XL2'length of DBG-CONTENT data'
```

Output: The debug item is allocated and filled in as specified by the declarative.

Calling Information: This subroutine is called by the compiled code. Calls ILBDCMM0 (for GETCORE operations).

OBJECT-TIME EXECUTION STATISTICS SUBROUTINES

Programmers can specify three options in the PARM field of the EXEC statement to generate statistics for helping them make their programs more efficient. The VERBSUM and VERBREF options are implemented by the compiler, producing statistics on the design of the programs. The COUNT option is implemented by the compiler and object-time execution statistics subroutines, producing statistics on the frequency with which sections of the programs are executed.

RELATIONSHIP TO THE DEBUG CONTROL SUBROUTINE

The object-time execution statistics subroutines are controlled and supported by the debug control subroutine, ILBDDBG0 (see "Diagnostic Aid Subroutines").

The debug initialization subroutine is called by INIT3 in the object module whenever the COUNT option has been specified, regardless of whether any debugging options have also been specified. The debug initialization subroutine calls COUNT subroutines to perform COUNT initialization. The debug control subroutines also provide the following functions for the object time execution statistics subroutines:

- Call COUNT subroutines at abnormal termination of object module execution (ILBDDBG2)
- Write on the debug print file (SYSDBOUT) if count errors are found (ILBDDBG1)

COUNT DATA AREAS

The object-time execution statistics subroutines use a number of tables:

- The count table, built by the compiler as part of the object module. The table contains each procedure-name and verb as it is encountered in the source program, each verb being in Pl-code form.
- The verb translate table, verb table, and verb text table -- parts of subroutine ILBDTC30 -- which enable the subroutine to translate the verb codes into EBCDIC form for listing, and also enables the subroutines to locate verbsum table entries.
- The COUNT chain, space for which is obtained by ILBDTC00. This table is modified by the object-time execution statistics subroutines and contains the program-ids, pointers, and the node count table.
- The node count table contains the current number of times each count-block is entered. A count-block is a set of COBOL verbs such that (exclusive of ABENDs) each verb in the block is executed if, and only if, the first verb is executed.
- The verbsum table, space for which is obtained by subroutine ILBDTC30. This table is built at termination of object module execution and contains a summary of the information in the count tables and node count tables.

The COUNT subroutines use the count common area (ILBDTC01) to control the monitoring process. It also uses the debug common area (DBG0COM) for printing. These tables, chains, and common areas are described in "Section 3: Data Areas." "Section 2: Program Organization" shows how the tables are used.

COUNT OPERATIONS

At the start of object module execution the debug control subroutine calls the ILBDTC00 subroutine to begin implementation of the COUNT option.

During object module execution subroutine ILBDCT10 is called by compiled code to update the counts of the frequency with which count-blocks of object module statements are executed. A count-block is determined by the compiler on the basis that any statement in it is executed if and only if all statements in the block are executed. The start of a block is called a node.

An example of what constitutes a count-block is as follows:

Statement Number	Statement Type
1	ADD
2	SUBTRACT
3	MOVE
4	IF...GO TO...
5	ADD

Statement 1 is a node for the first count-block, which consists of statements 1 through the IF in statement 4. The GO TO part of statement 4 is the node for a second count-block. Statement 5 is the node for the third count-block.

Each count-block is assigned a unique number. At each node in the object module is embedded a call to ILBDCT10 with a parameter consisting of the appropriate count-block number.

At termination of load module execution, abnormal or otherwise, the ILBDTC20 and ILBDTC30 subroutines write the COUNT option statistics on SYSCOUNT.

Diagram 12 in "Section 2: Program Organization" show COUNT operations in more detail. The subroutines themselves are described individually below.

COUNT Initialization Subroutine (ILBDTC00) [KA]

Operation: Initializes the count common area, gets space for and initializes the count chain, and initializes the count chain pointer in the object module TGT.

Called by: ILBDDBG0, which was called by INIT3.

Linkage:

```
L     R15,=V(ILBDTC00)
L     R1,A(parameter list)
BALR  R14,R15
```

where the parameter list is:

```
DC    1H'number-of-count-blocks'
```

Calls: GETVIS
ILBDDBG1

Input:

1. Register 1 points to the number of entries for the count table
2. Register 8 points to the TGT
3. Register 13 points to the debug common area (ILBDDBG7)
4. Registers 14, 15: standard linkage

Output:

1. Count chain generated and initialized
2. Count common area initialized and/or bits set in count common
3. Object module TGT points to the count chain

Count Frequency Subroutine (ILBDCT10) [KB]

Operation: Updates the appropriate node counter by one and saves the caller's count-block number in the count chain.

Called by: Generated code in the object module.

Linkage:

without SYMDMP option

```
BALR  1,12
DC    H'count-block number' (Goes to
      the COUNT linkage area in
      the PGT)
```

with SYMDMP option

```
BAL   1,8(12)
DC    H'count-block number' (Goes to
      the COUNT linkage area in
      the PGT)
```

where the COUNT linkage area of the object module PGT contains:

```
L     15,=V(ILBDCT10)
BCR   15,15
DC    1H'0'
```

Calls: None

Input:

1. Register 1 points to the block number, and the return address is at 002(register 1).
2. Register 12 points to the PGT
3. Register 13 points to the TGT, where are contained the save area and a pointer to the appropriate count chain
4. Register 15 points to this subroutine

Output:

1. Appropriate node counter updated by one
2. Count chain contains the last count-block number

COUNT Termination Subroutine (ILBDTC20) [KC]

Operation: Called at termination of object module execution to determine if there are programs being monitored. If so, it calls subroutine ILBDTC30 to write execution statistics, and if the termination is normal, calls ILBDDBG8 to close the debug print file. If the termination is not normal, the debug print file is left open for debugging information.

Called by: Generated code in the object module, ILBDDBG2, ILBDABX0.

Linkage:

from ILBDDBG2

```
LA     1,=X'FFFFFFFF'
L      15,=V(ILBDTC20)
BALR   14,15
```

from all other callers

```
SR     1,1
L      15,V(ILBDTC20)
BALR   14,15
```

Calls: ILBDTC30
ILBDDBG8

Input:

Register	Contents	Meaning
1	Zero	Close debug print file after ILBDTC30 executes.
	Pointer to X'FFFFFFFF'	Do not close debug print file after ILBDTC30 executes.
13	Pointer to save area	
14,15	Standard linkage	

Output: None

COUNT Print Subroutine (ILBDTC30) [KD]

Operation: This subroutine computes and writes execution statistics on the debug print file upon termination of the program being monitored.

Called by: ILBDTC20

Linkage:

```
L      R15,V(ILBDTC30)
BALR   R14,R15
```

Calls: ILBDDBG1
ILBDDBG8
FREEVIS
GETVIS

Input:

Register	Contents	Meaning
1	Zeros or X'FFFFFFFF'	From ILBDTC20 input
9	Points to count common area	
13	Points to save area	
14,15	Standard linkage	

Output:

1. Printed execution statistics
2. Space for the count chains released
3. Count common area updated

Chart HM. VSAM OPEN And CLOSE Subroutine (ILBDVOC0) (Part 1 of 2)

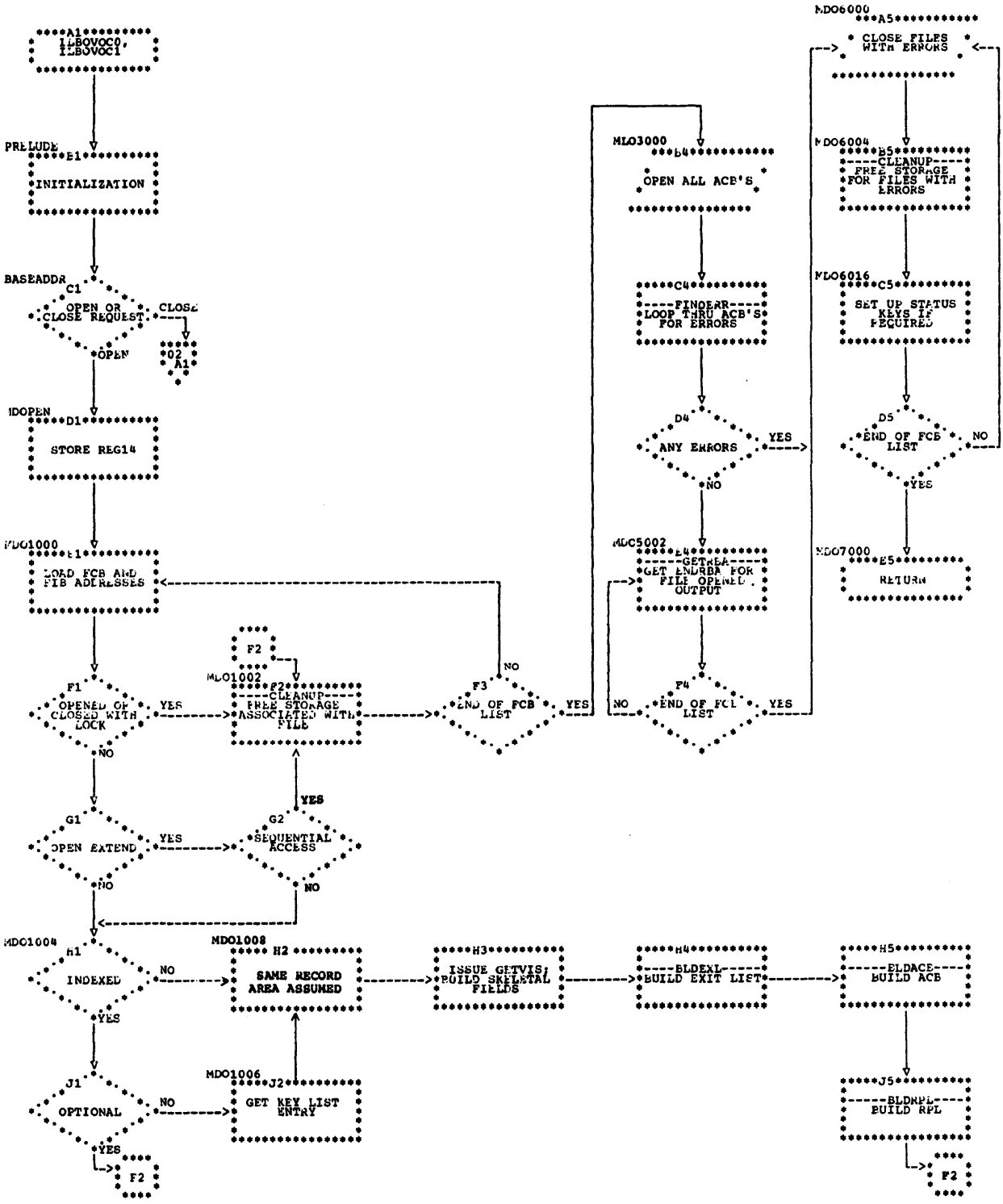


Chart HM. VSAM OPEN And CLOSE Subroutine (ILBDVOC0) (Part 2 of 2)

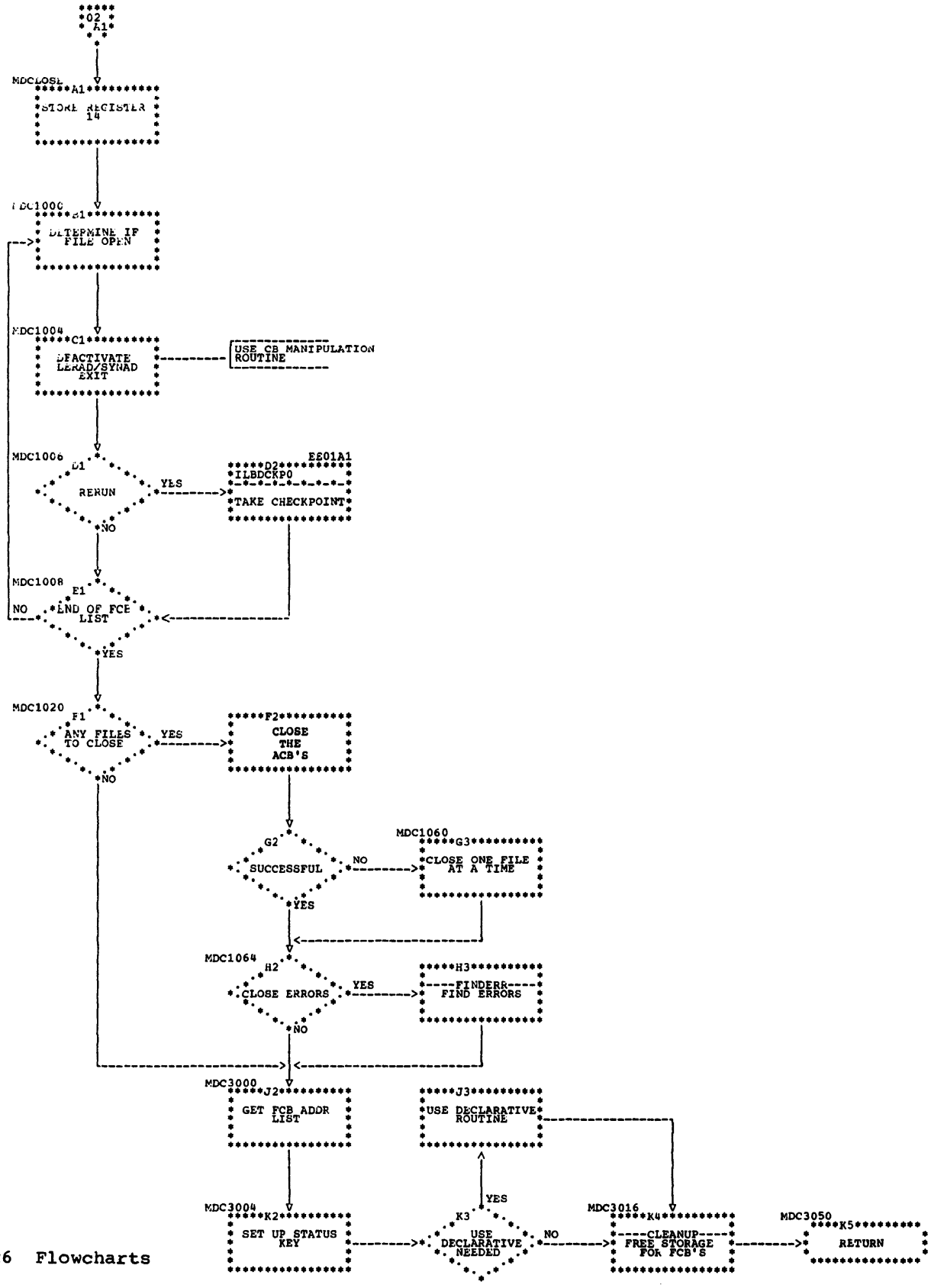


Chart HN. VSAM Action Request Subroutine (ILBDVIO0)

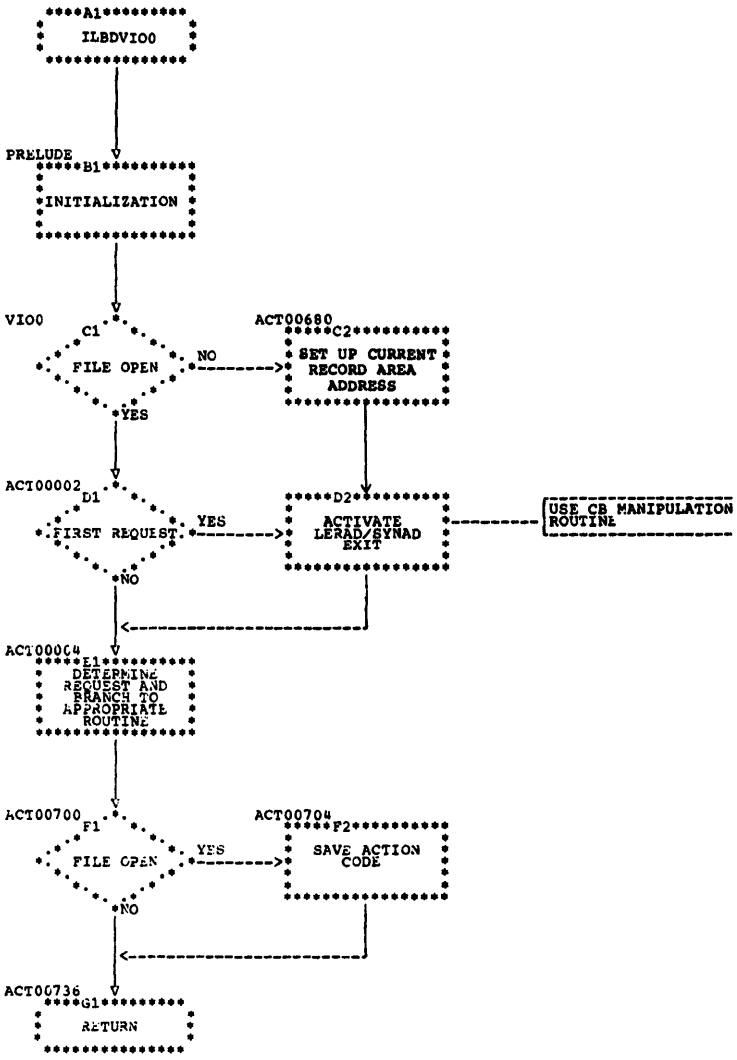


Chart IA. Separately Signed Numeric (ILBDSSN0)

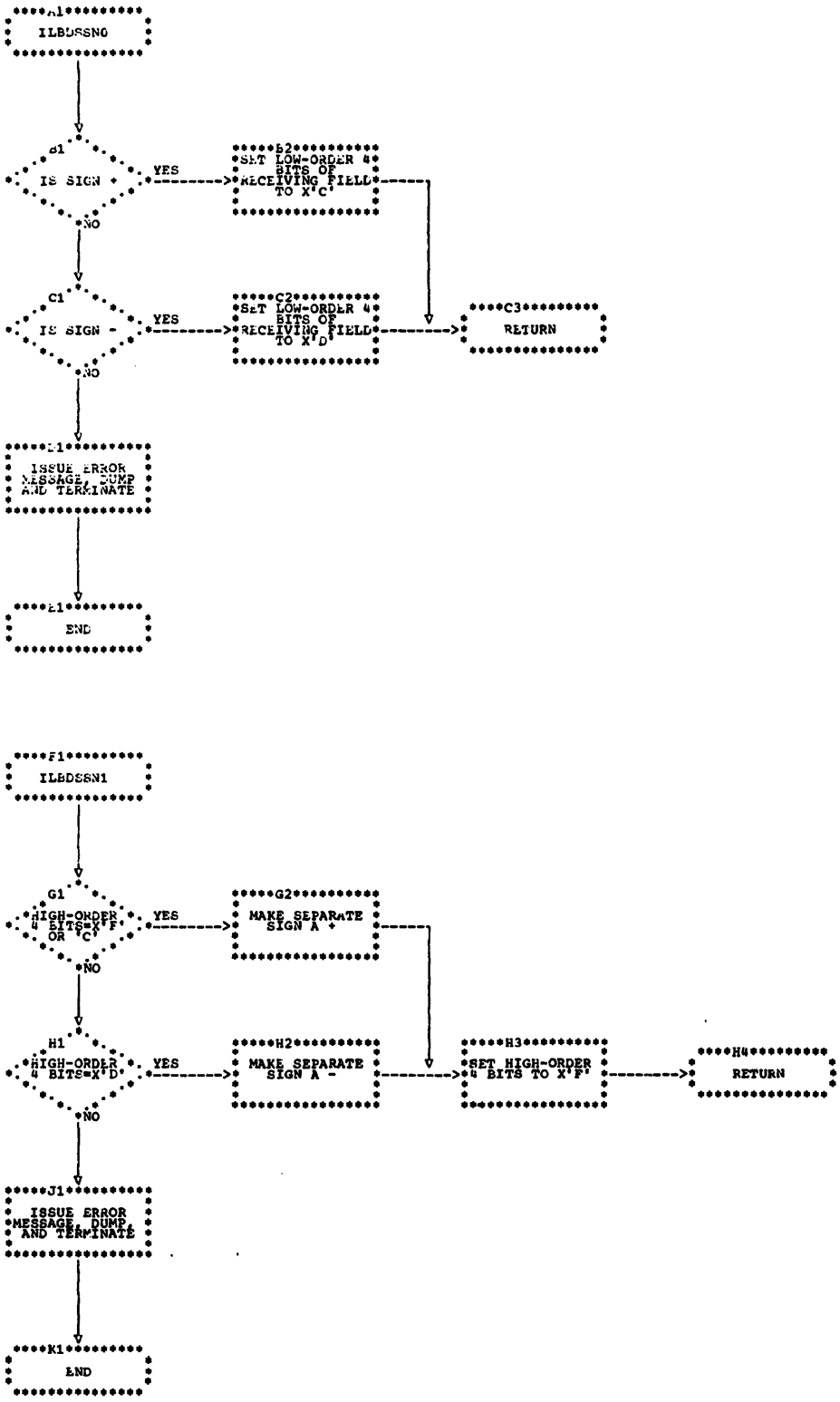


Chart JA. Test (ILBDDBG0) (Part 1 of 2)

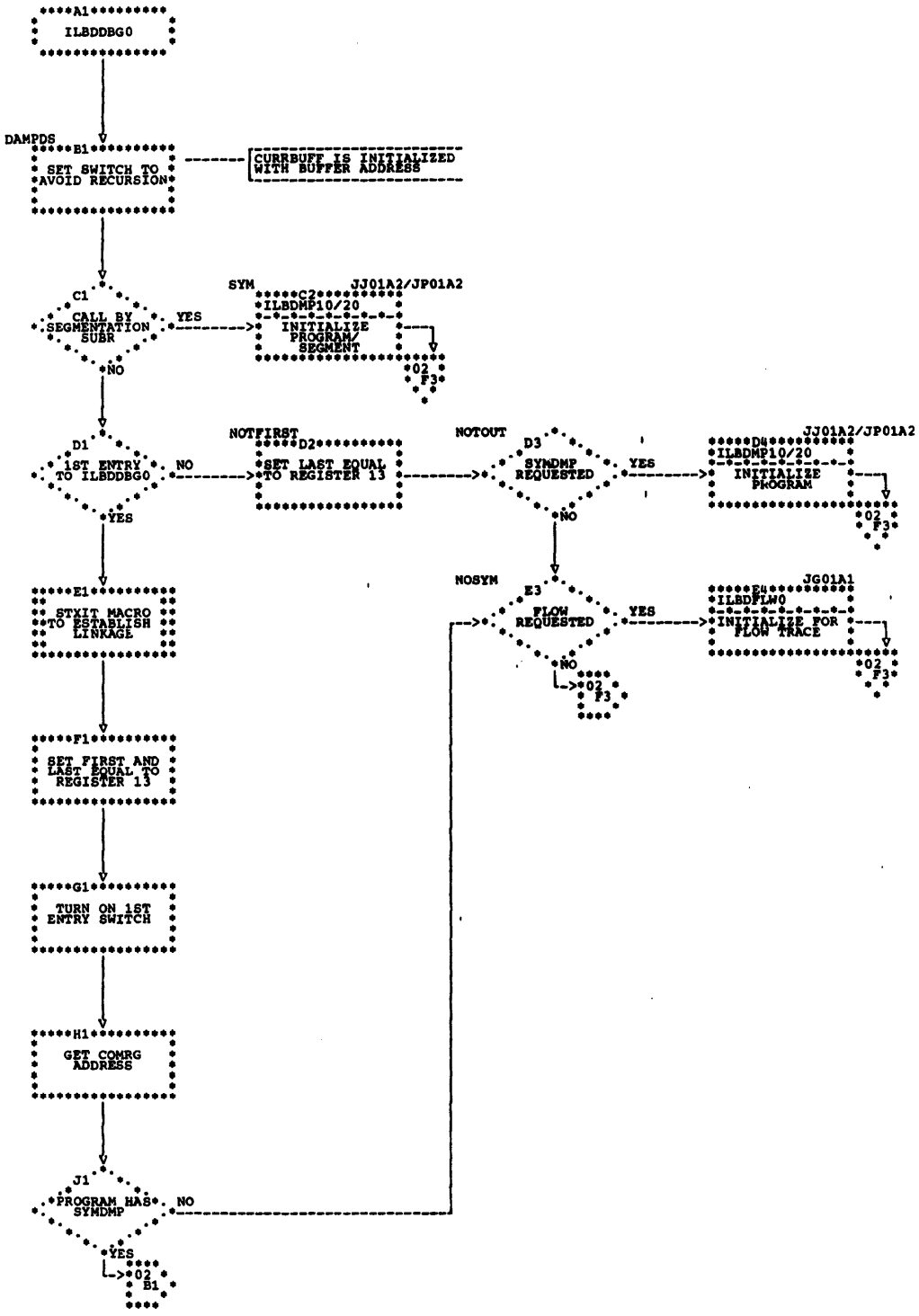


Chart JA. Test (ILBDDBG0) (Part 2 of 2)

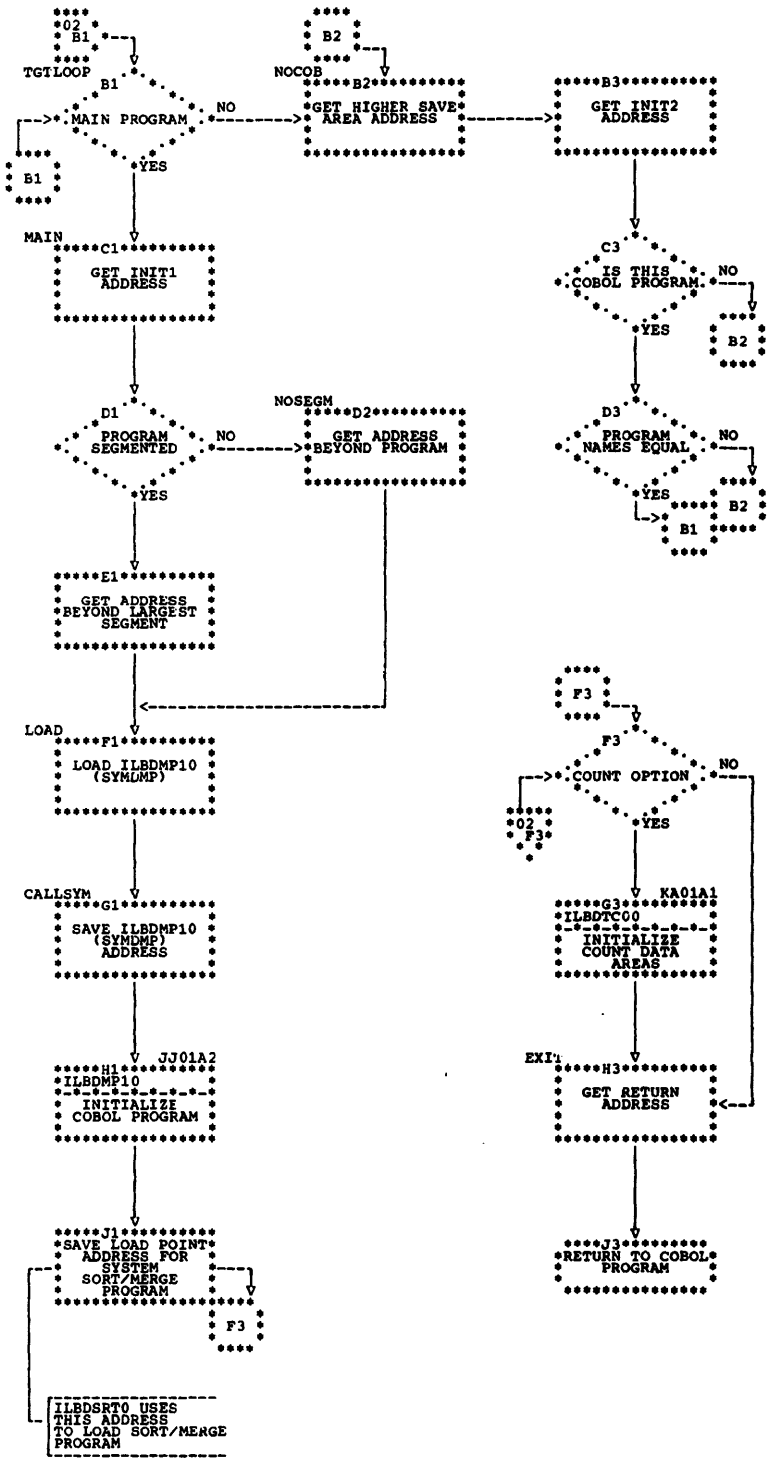


Chart JB. Print (ILBDDBG1)

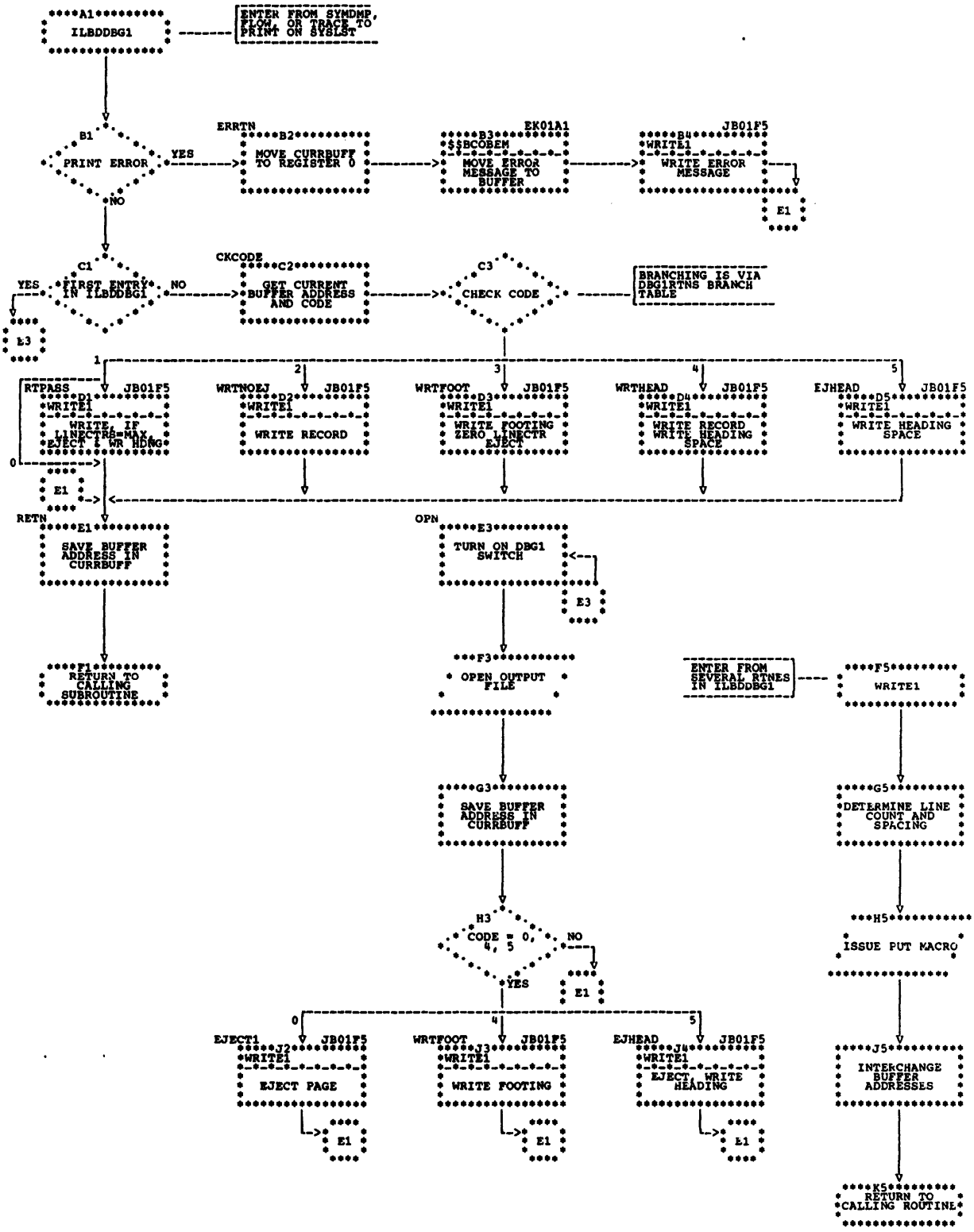


Chart JC. STXIT (ILBDDBG2), TGT Address (ILBDDBG3), and Save Register 14 (ILBDDBG4)

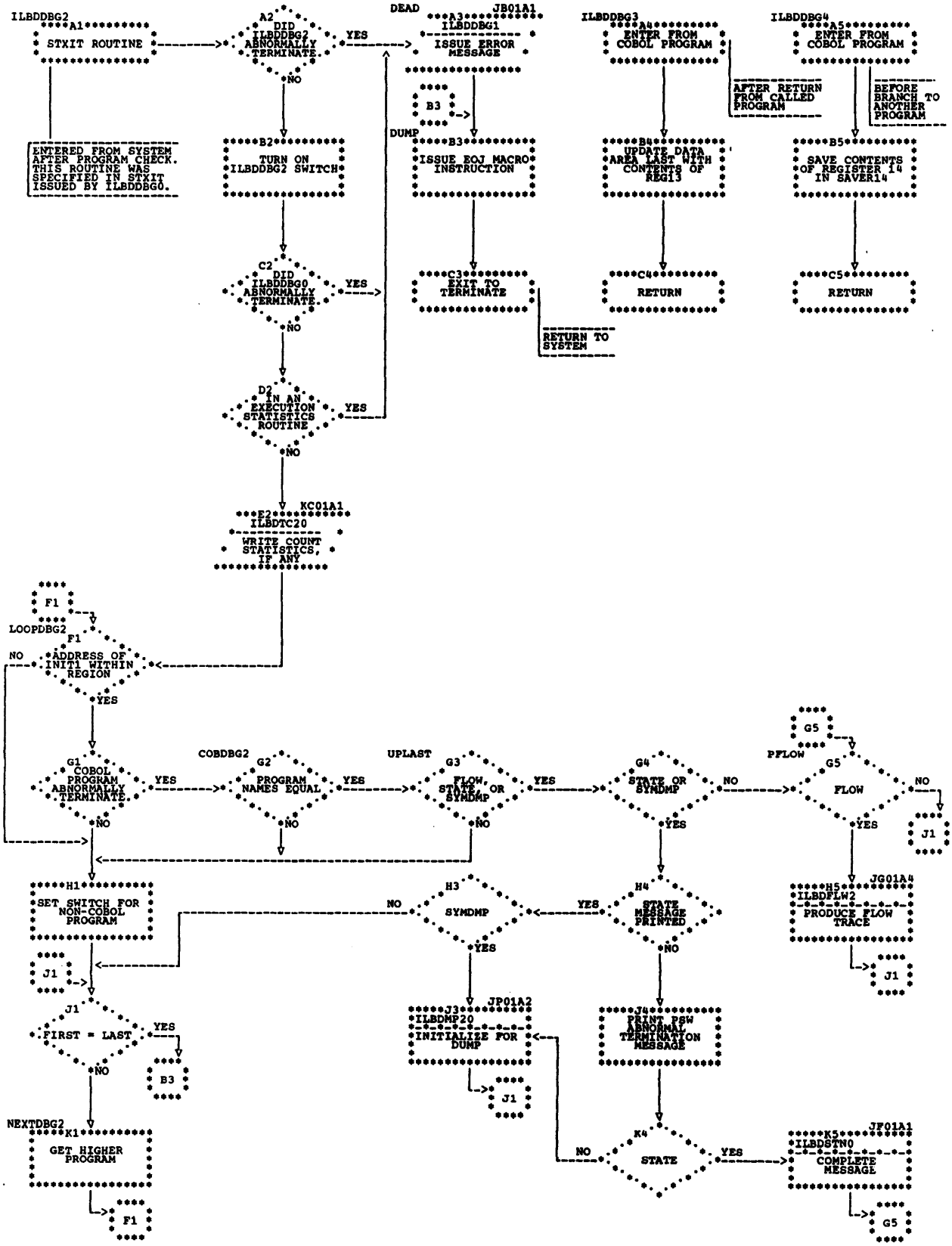


Chart JD. Dynamic Dump (ILBDDBG5)

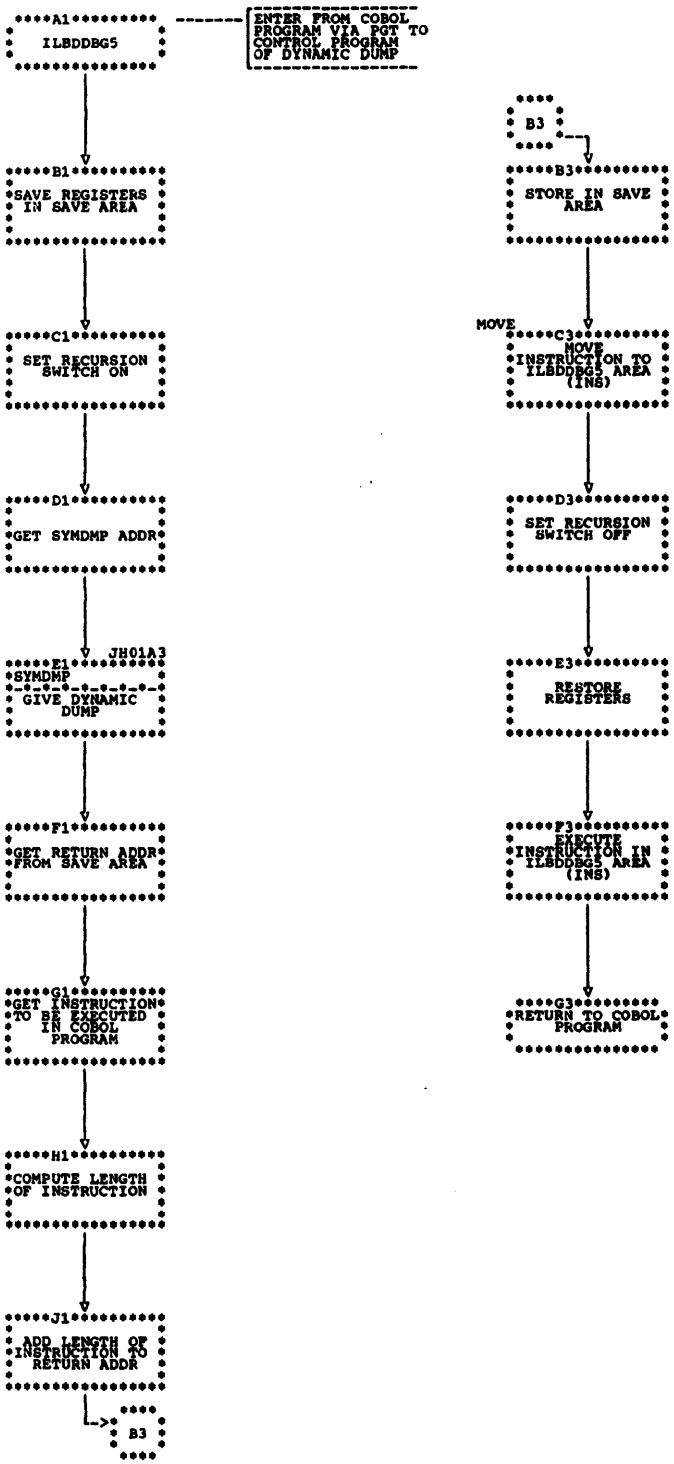


Chart JE. Range (ILBDDBG6) and Chose Debug File (ILBDDBG8) Subroutines

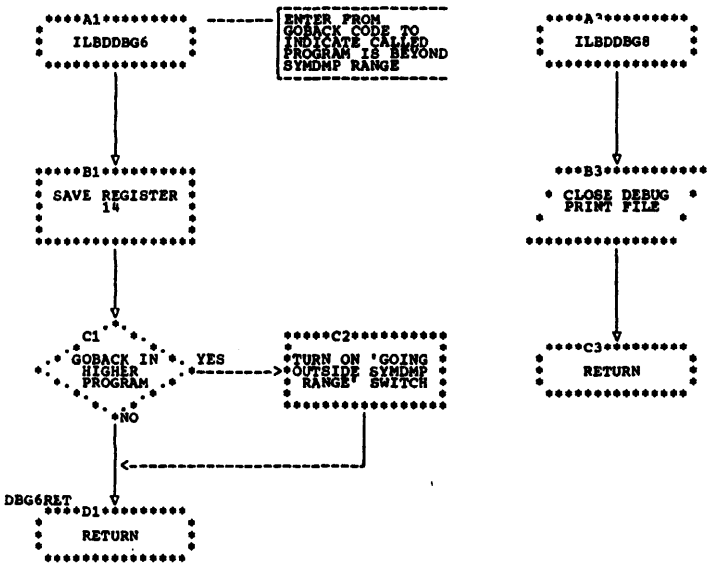


Chart JF. Statement Number (ILBDSTN0) (Part 1 of 2)

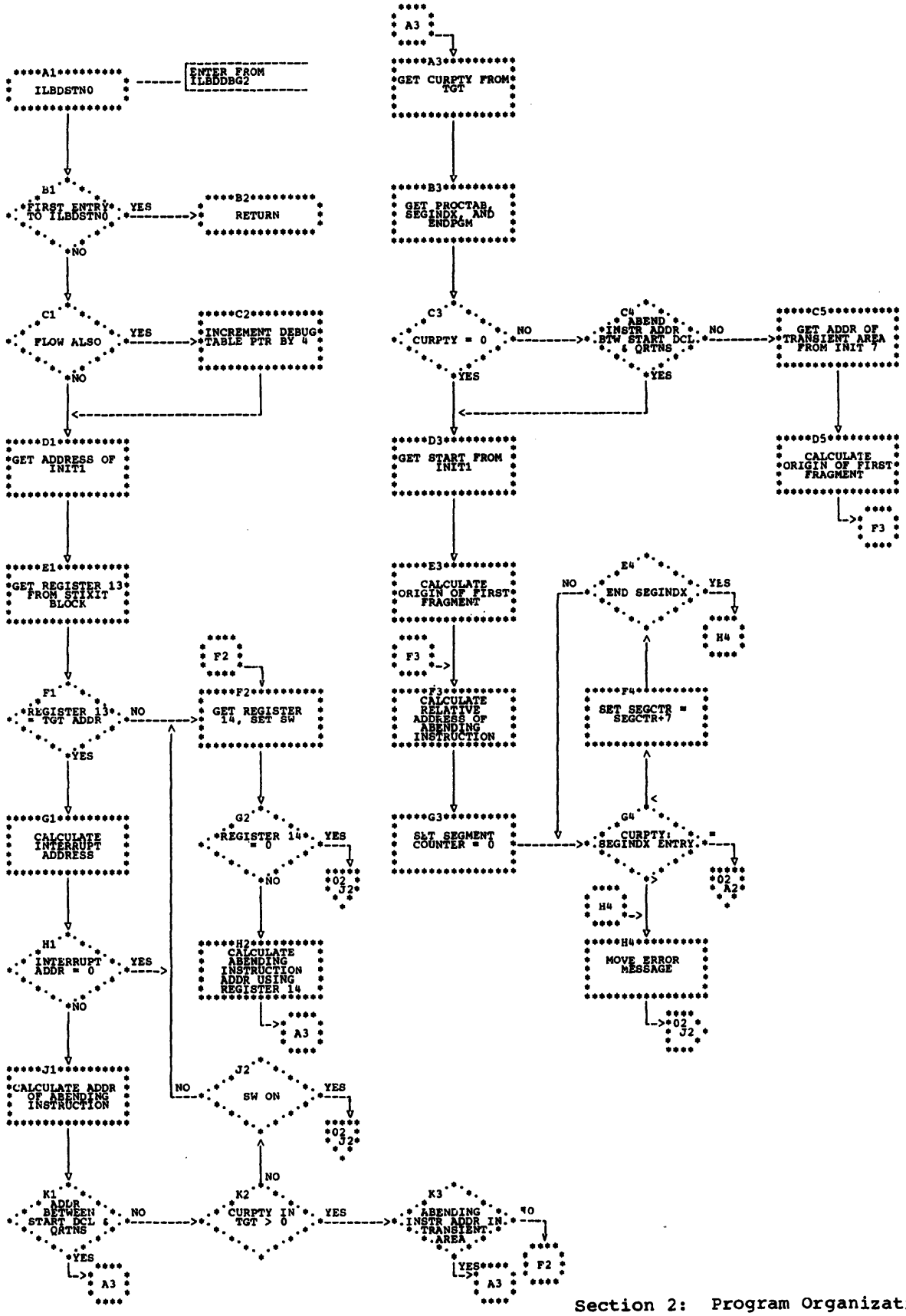


Chart JF. Statement Number (ILBDSTN0) (Part 2 of 2)

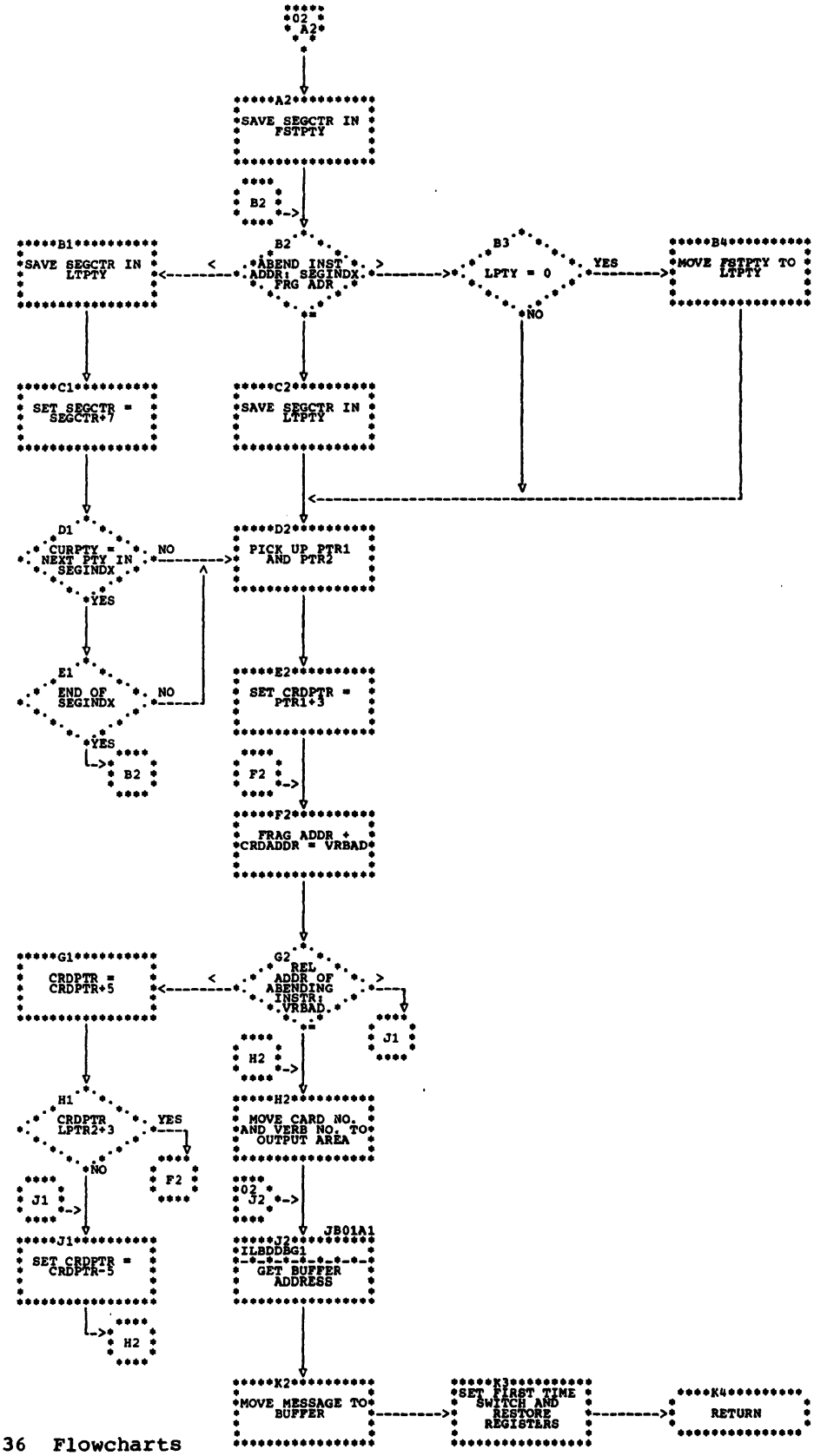


Chart JG. Flow Trace (ILBDFLW0)

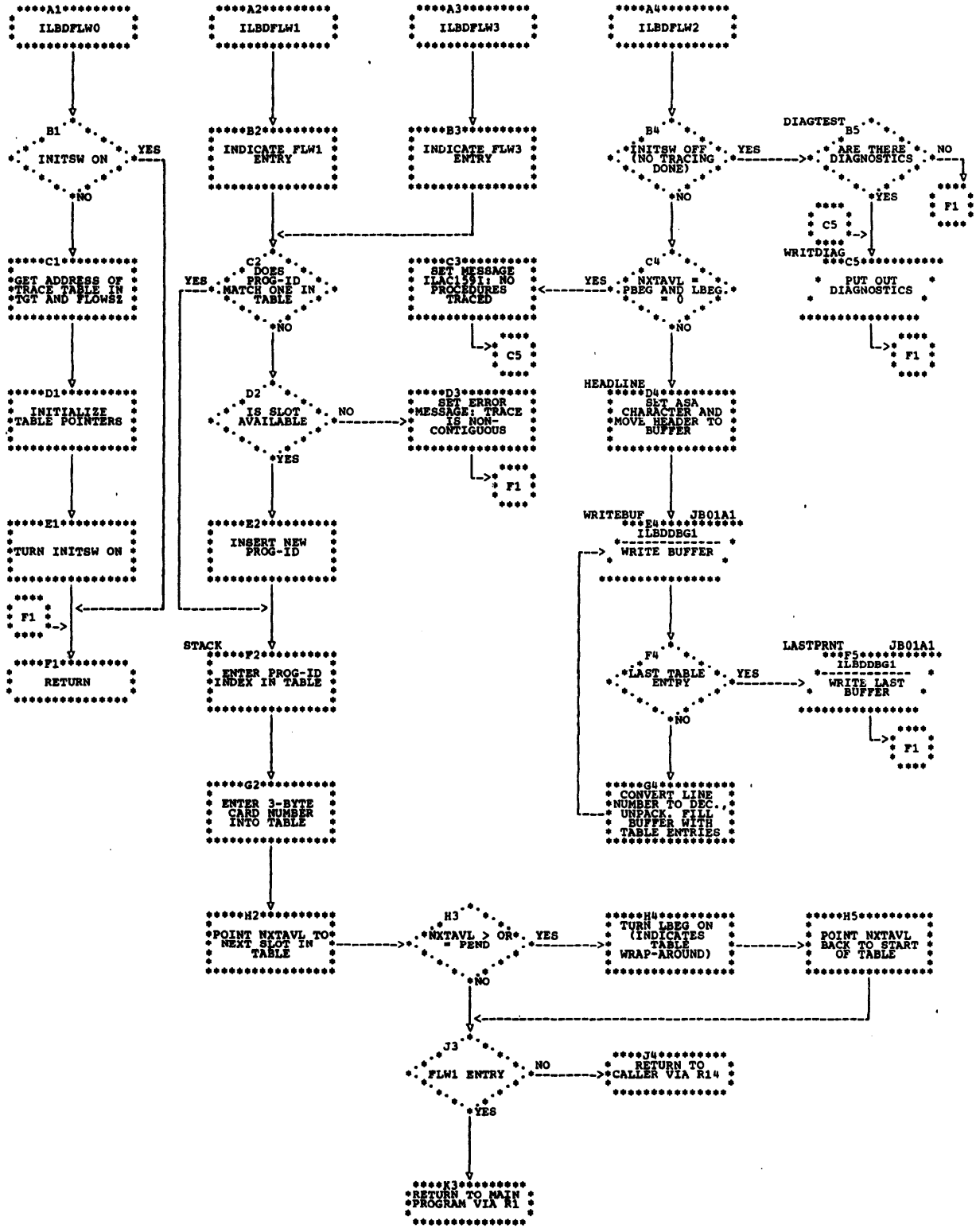


Chart JH. SYMDMP - Overall

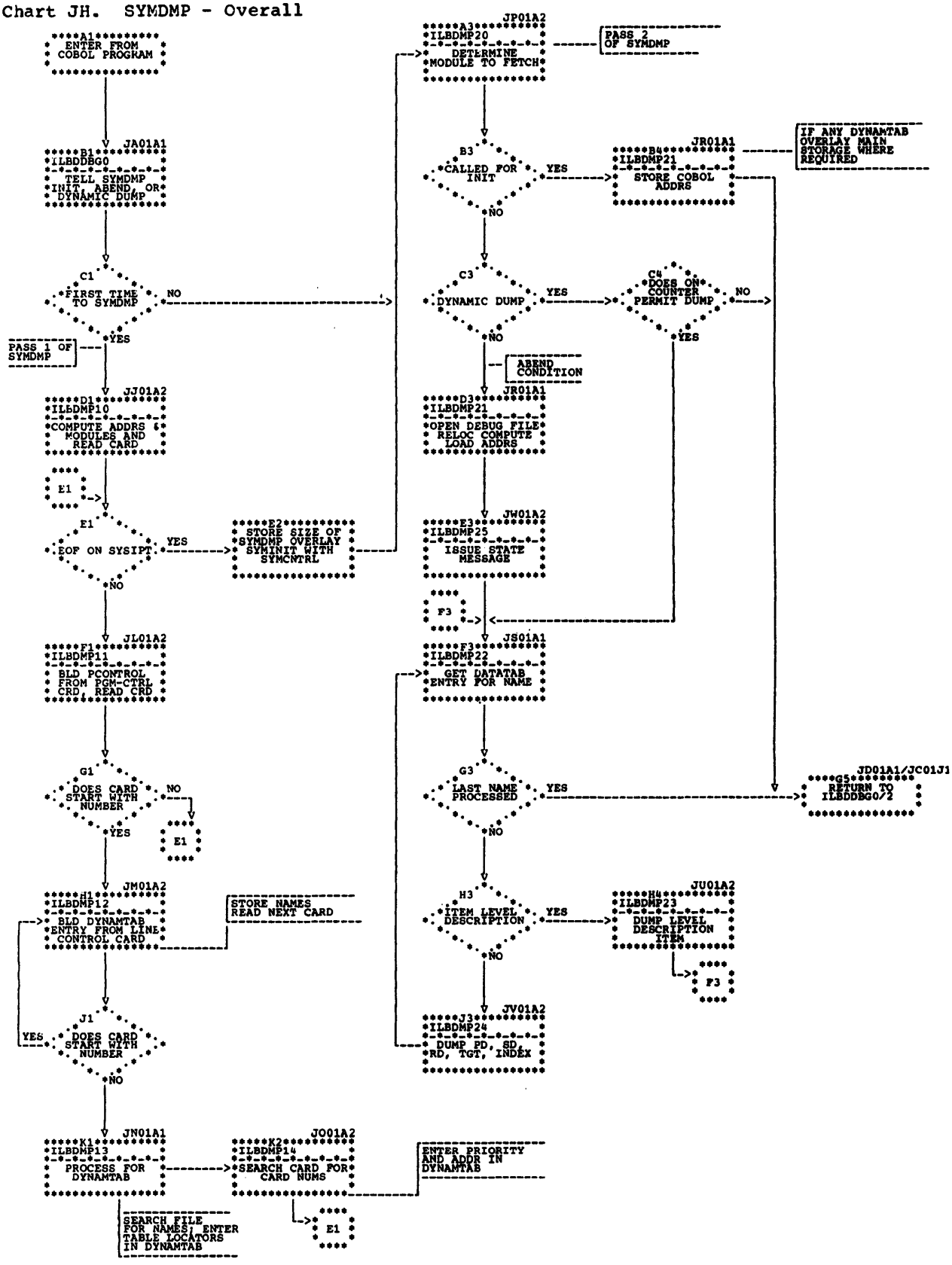


Chart JI. IODISK/IOTAPE (ILBDMP01/ILBDMP02)

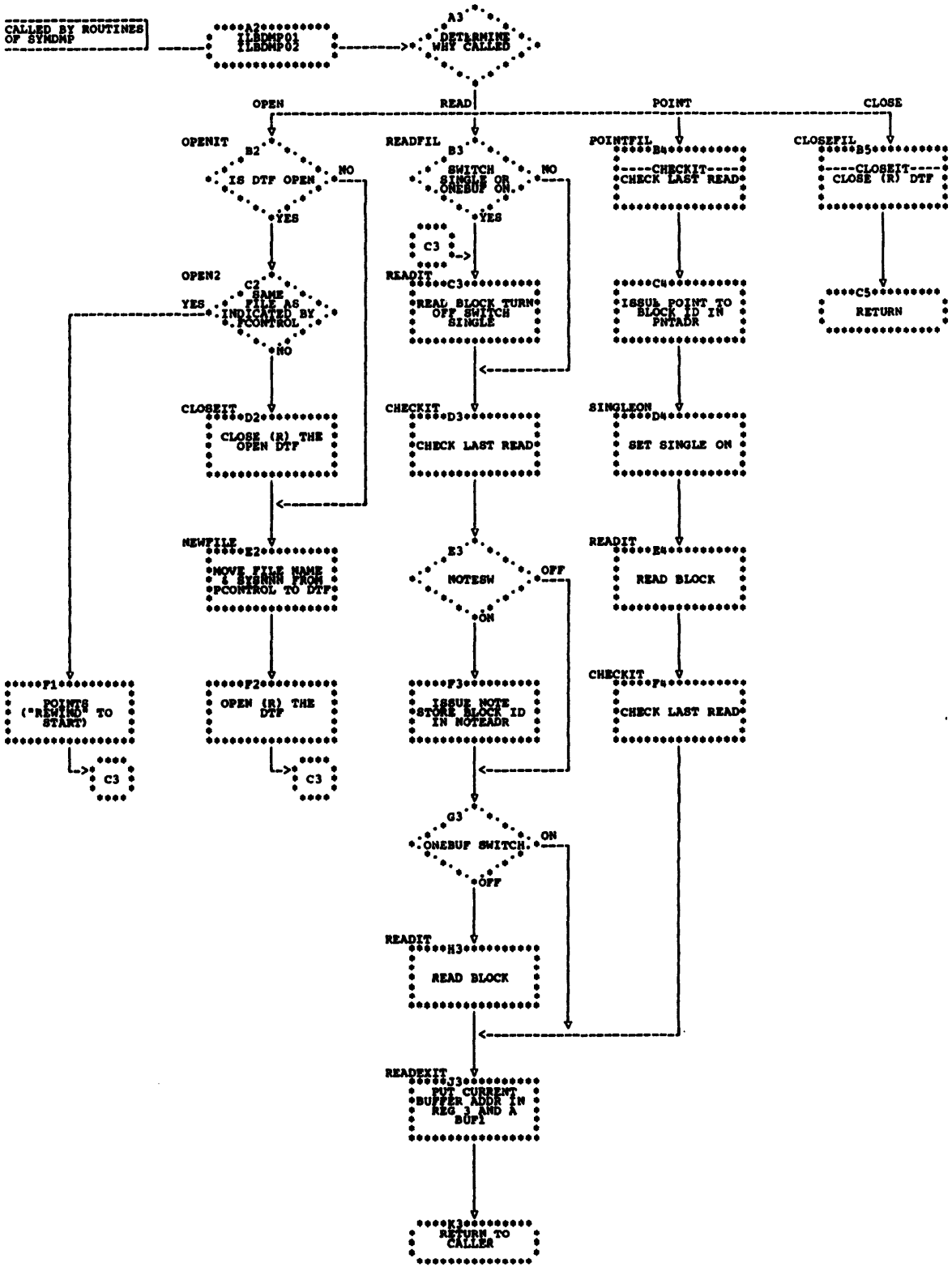


Chart JJ. SYMINIT (ILBDMP10)

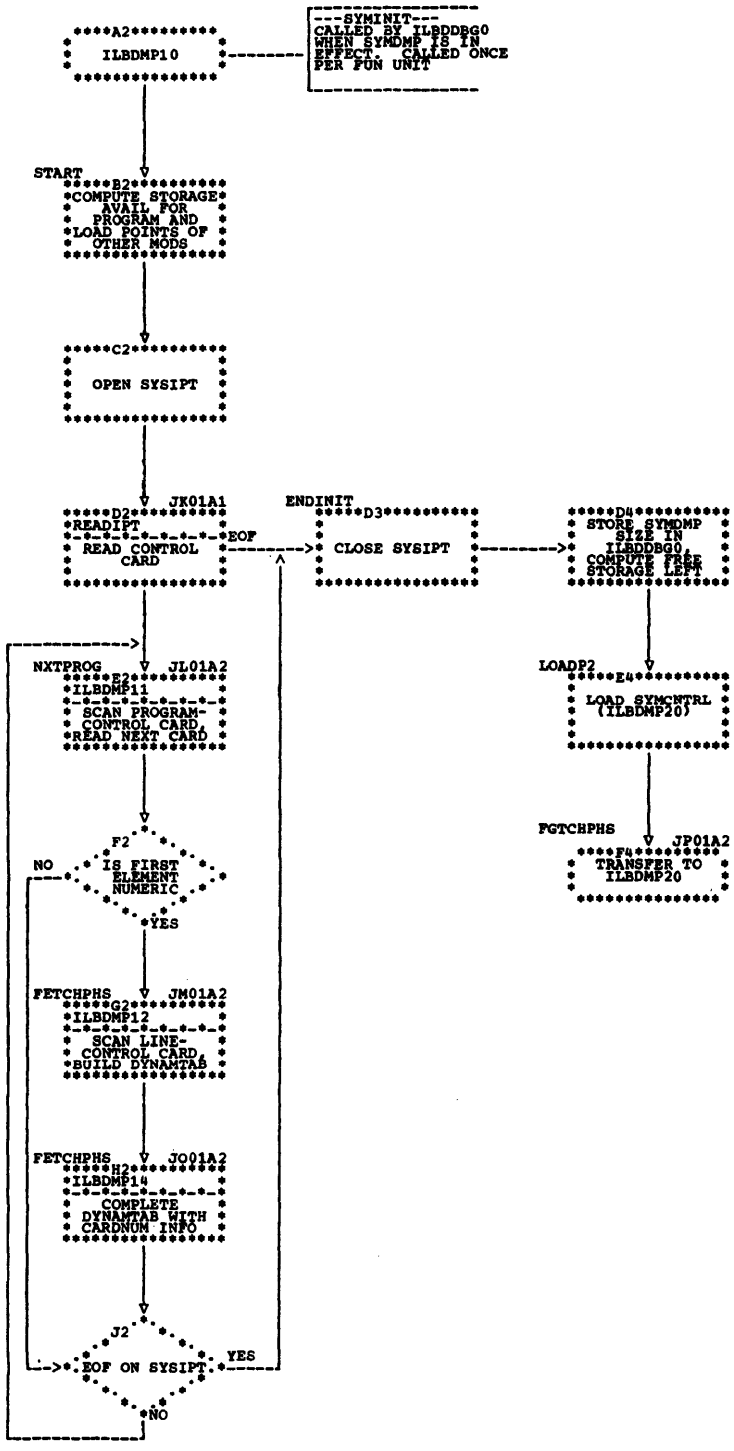


Chart JK. READIPT/ERROR (in ILBDMP10)

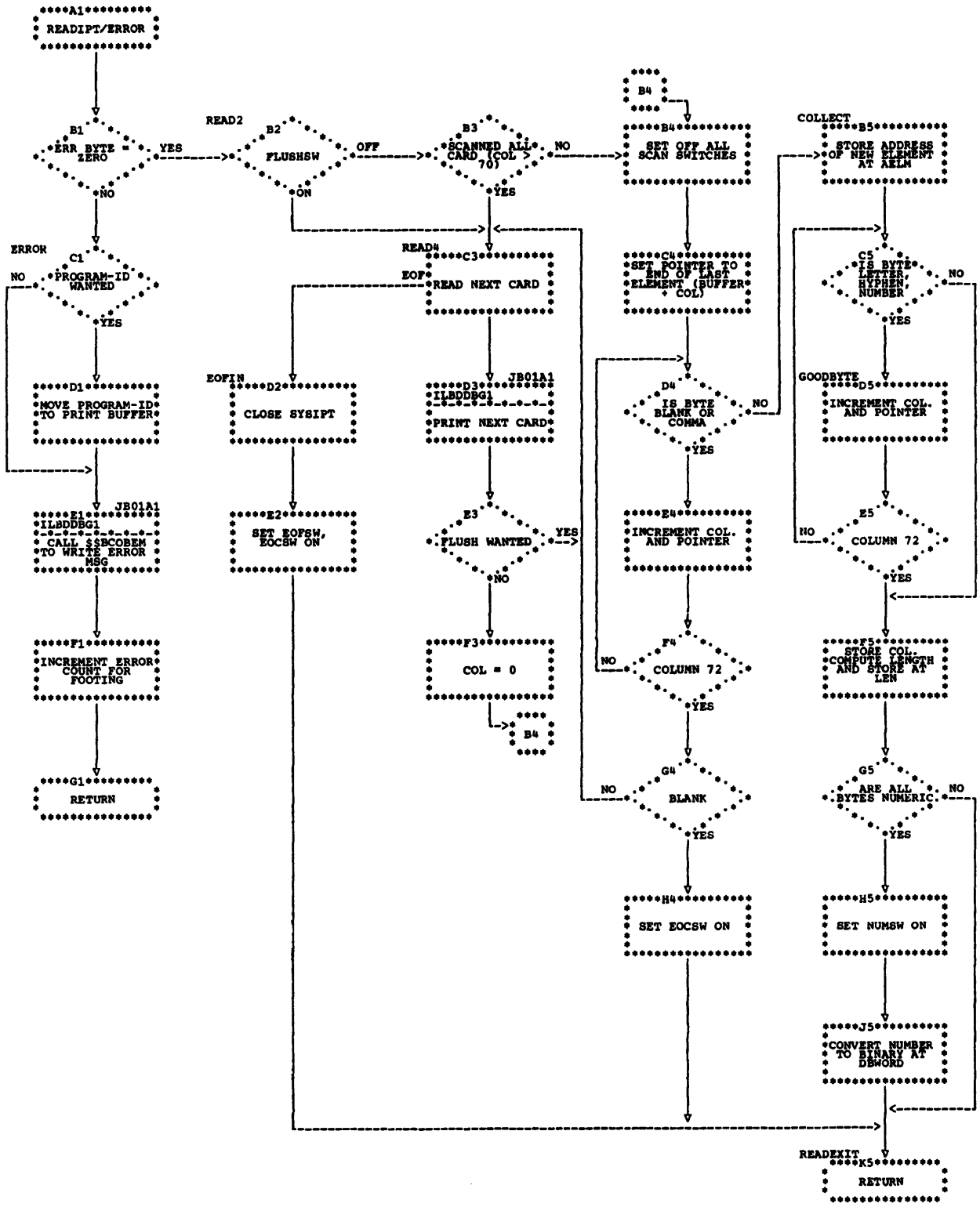


Chart JL. SCANP (ILBDMP11)

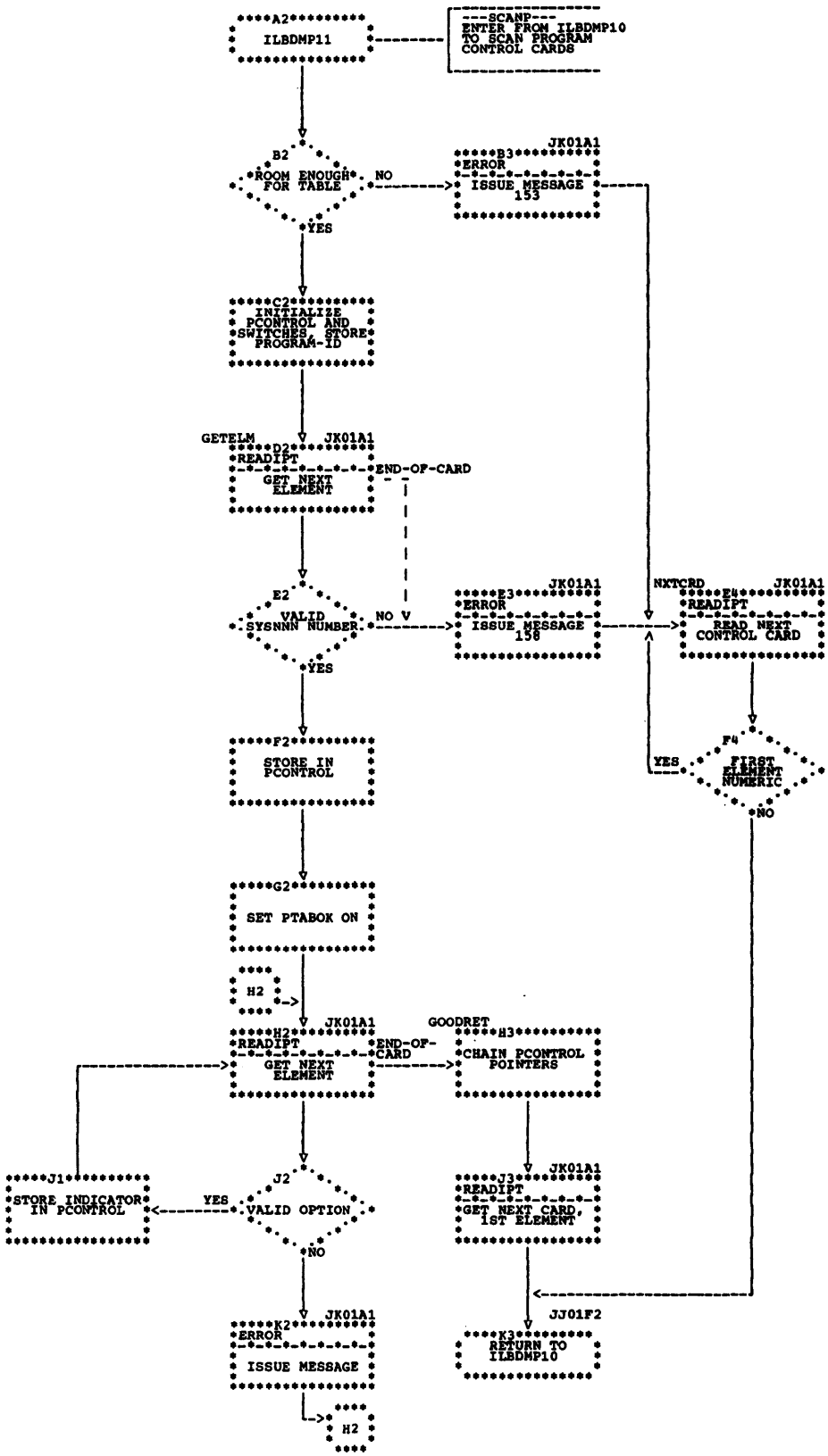


Chart JM. SCAND (ILBDMP12)

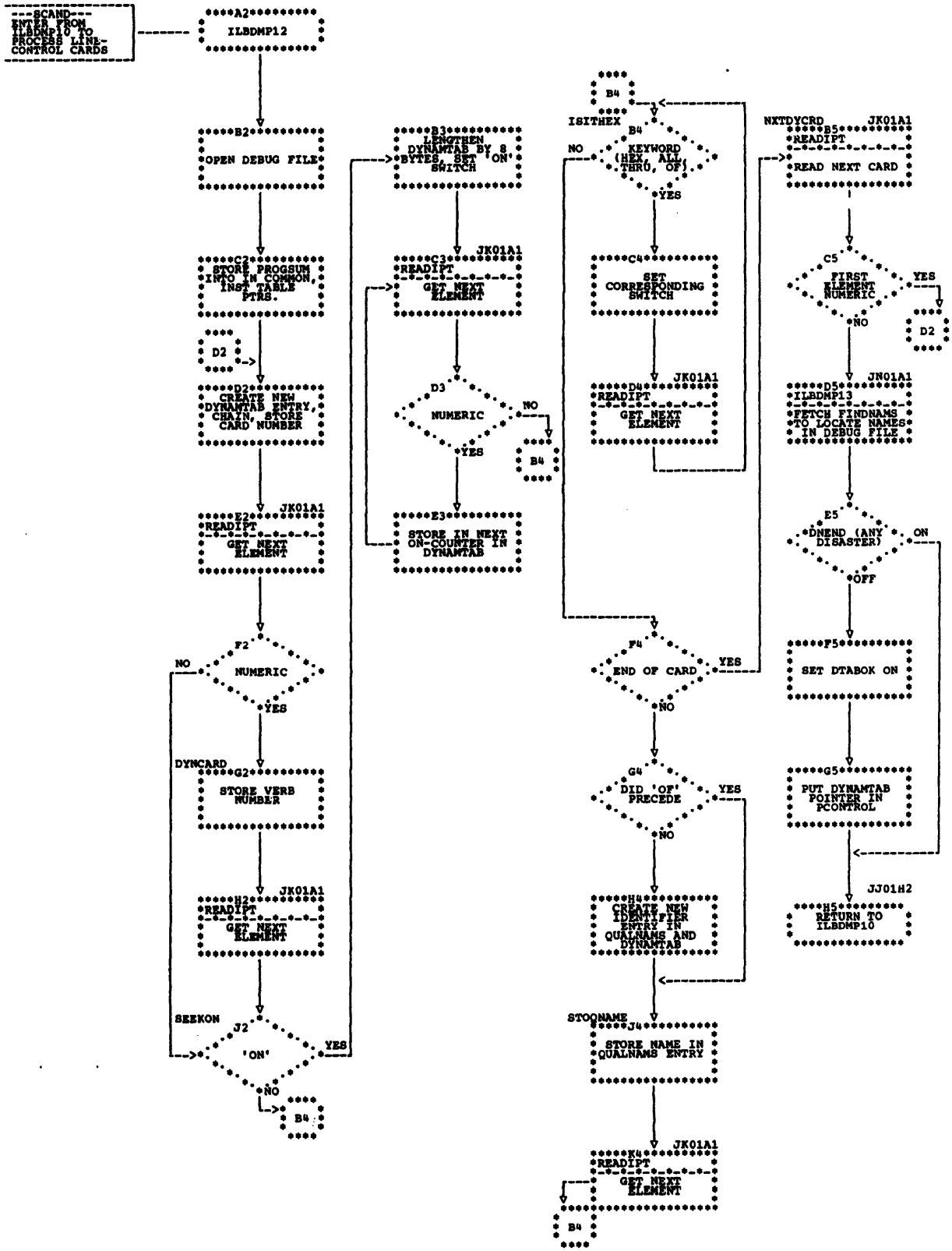


Chart JN. FINDNAMS (ILBDMP13)

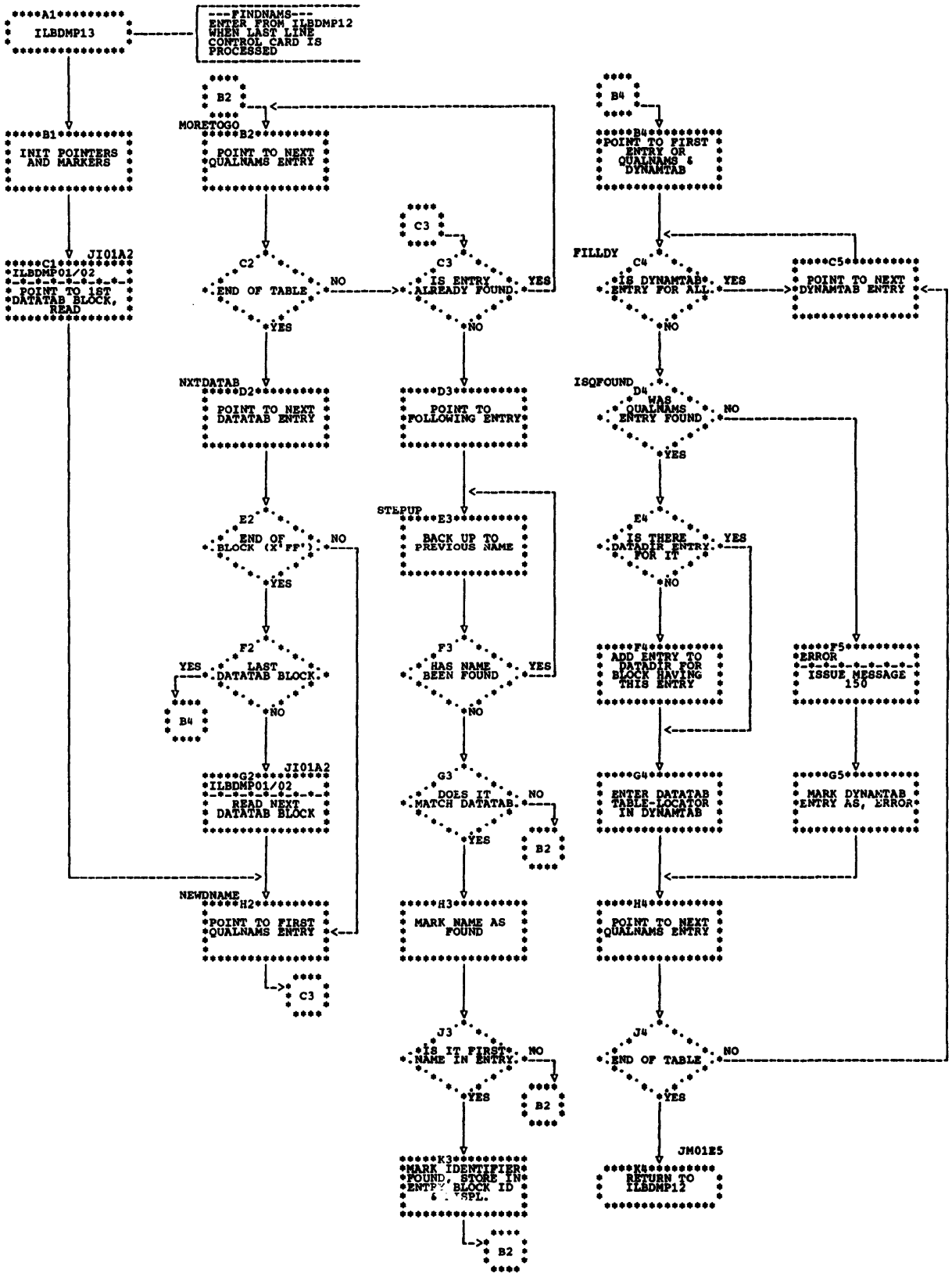


Chart JO. FINDLOCS (ILBDMP14)

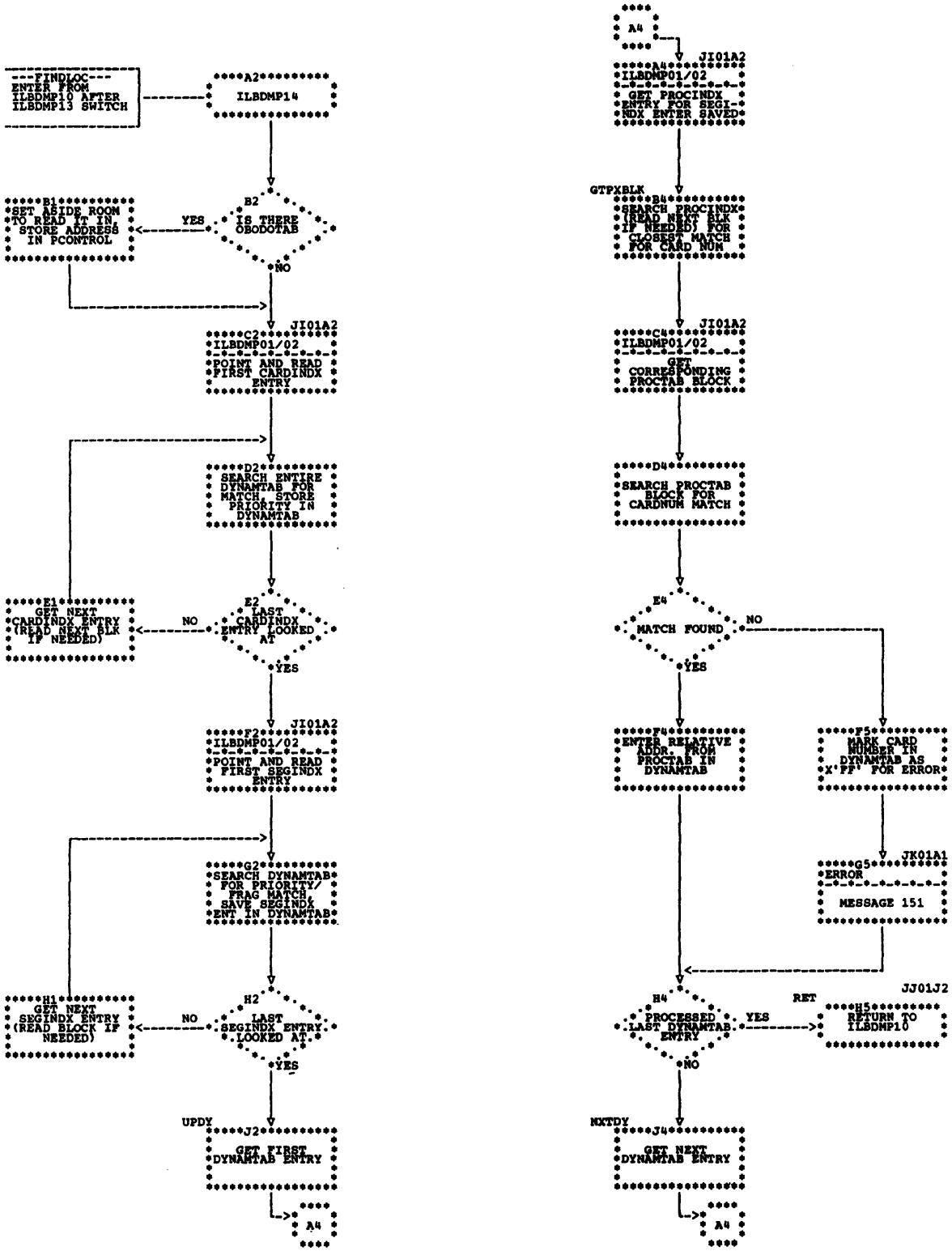


Chart JP. SYMCNTRL (ILBDMP20)

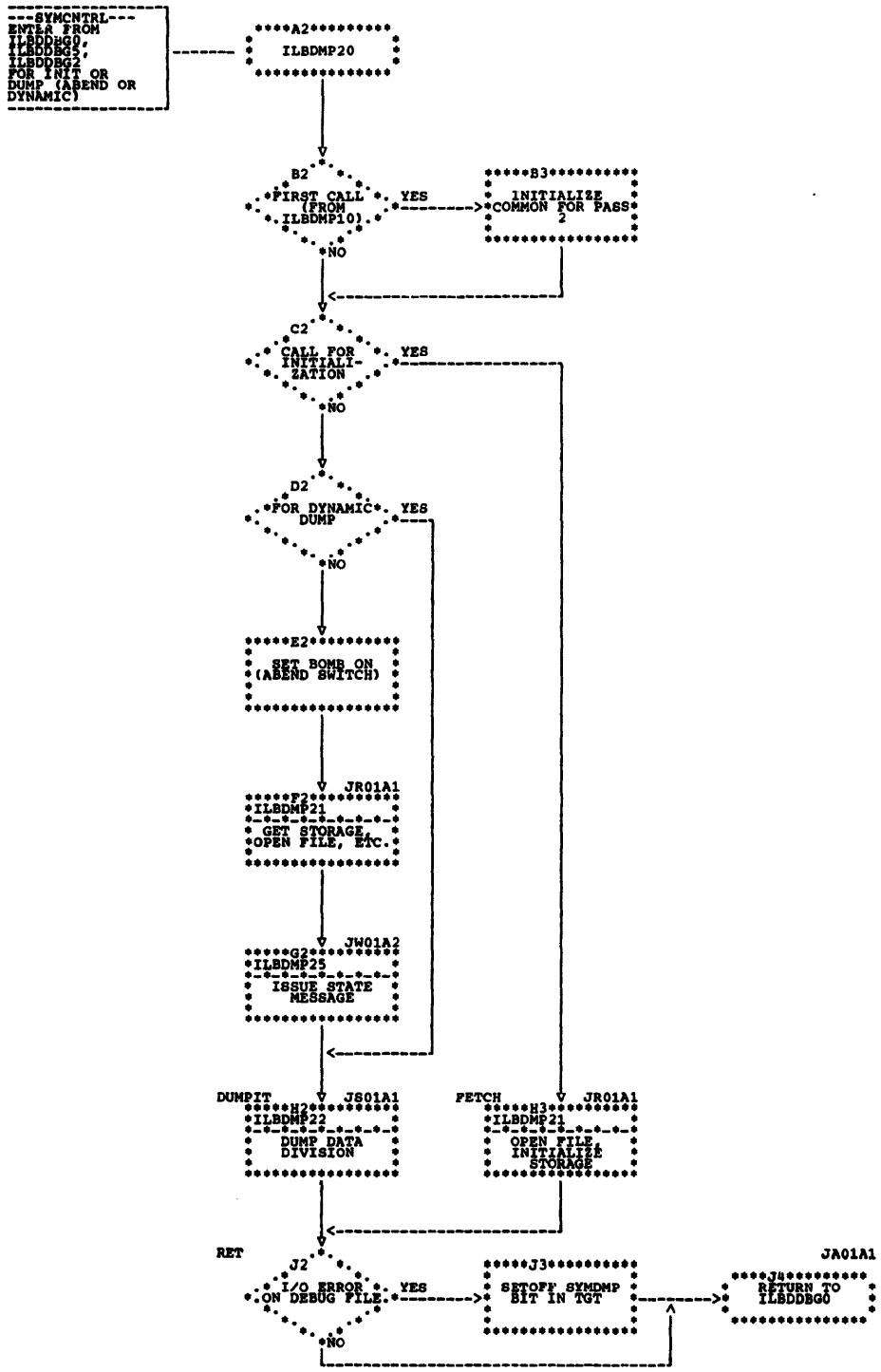


Chart JQ. HEXDUMP (in ILBDMP20)

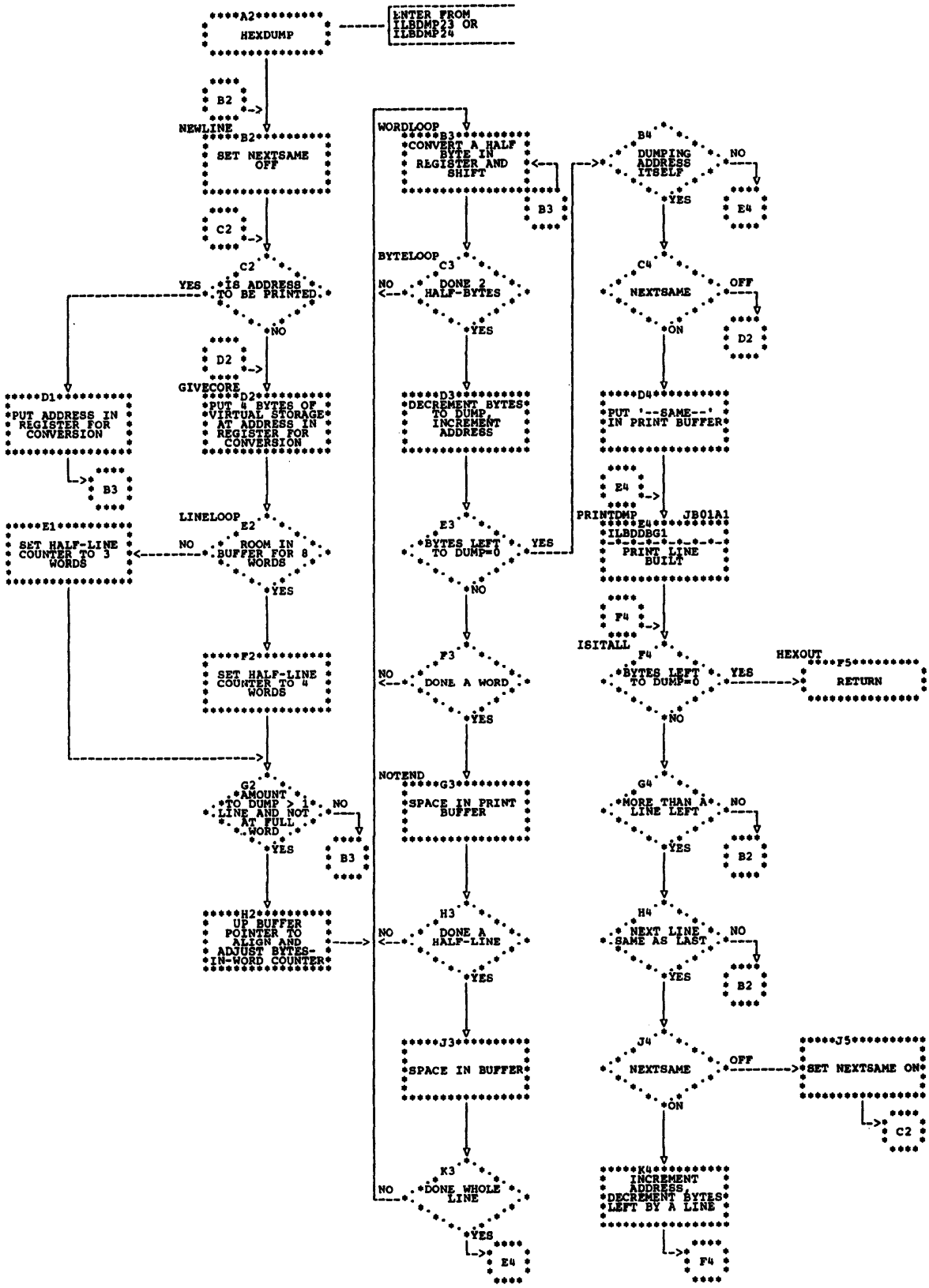


Chart JR. SEGINIT (ILBDMP21)

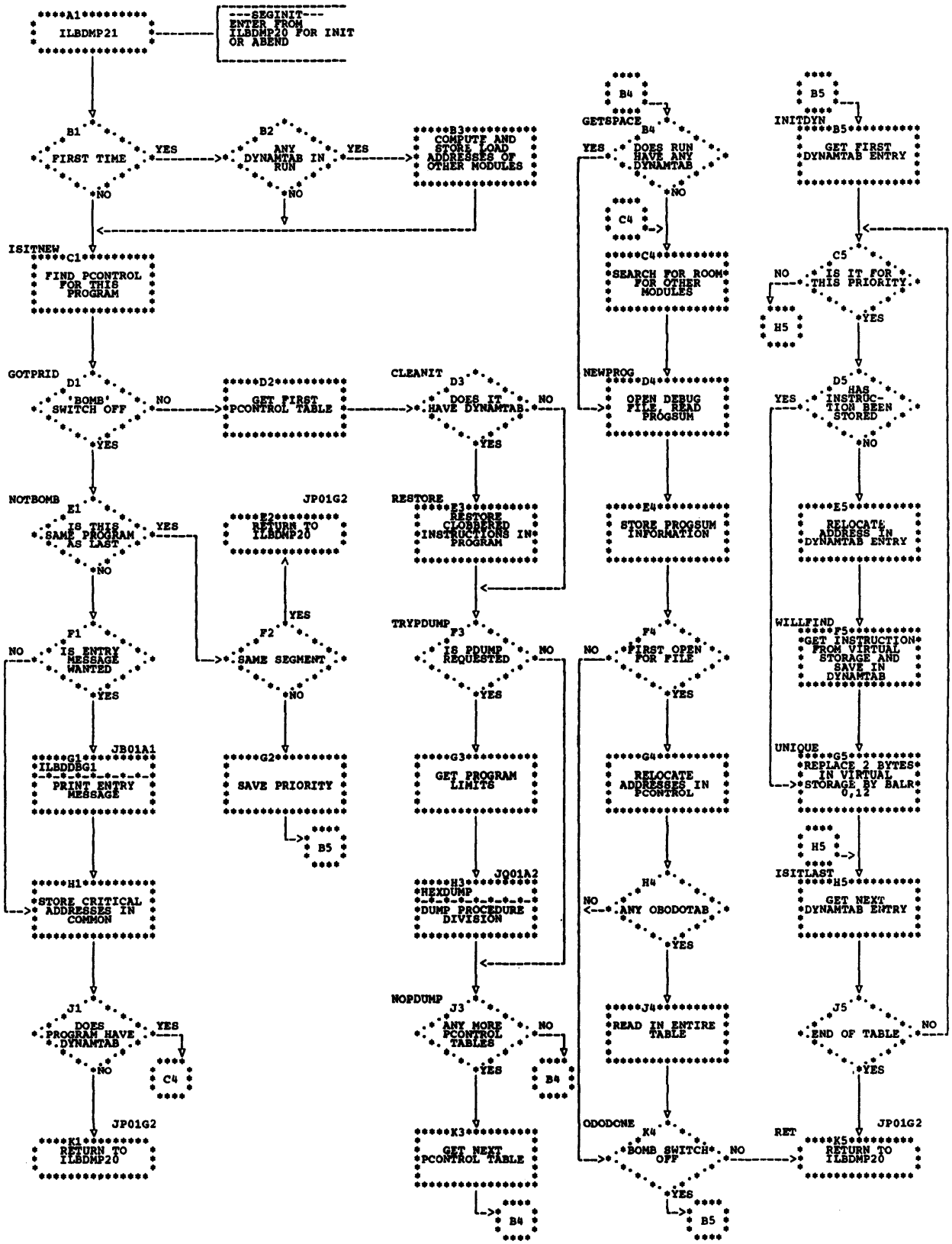


Chart JS. DMPCNTRL (ILBDMP22)

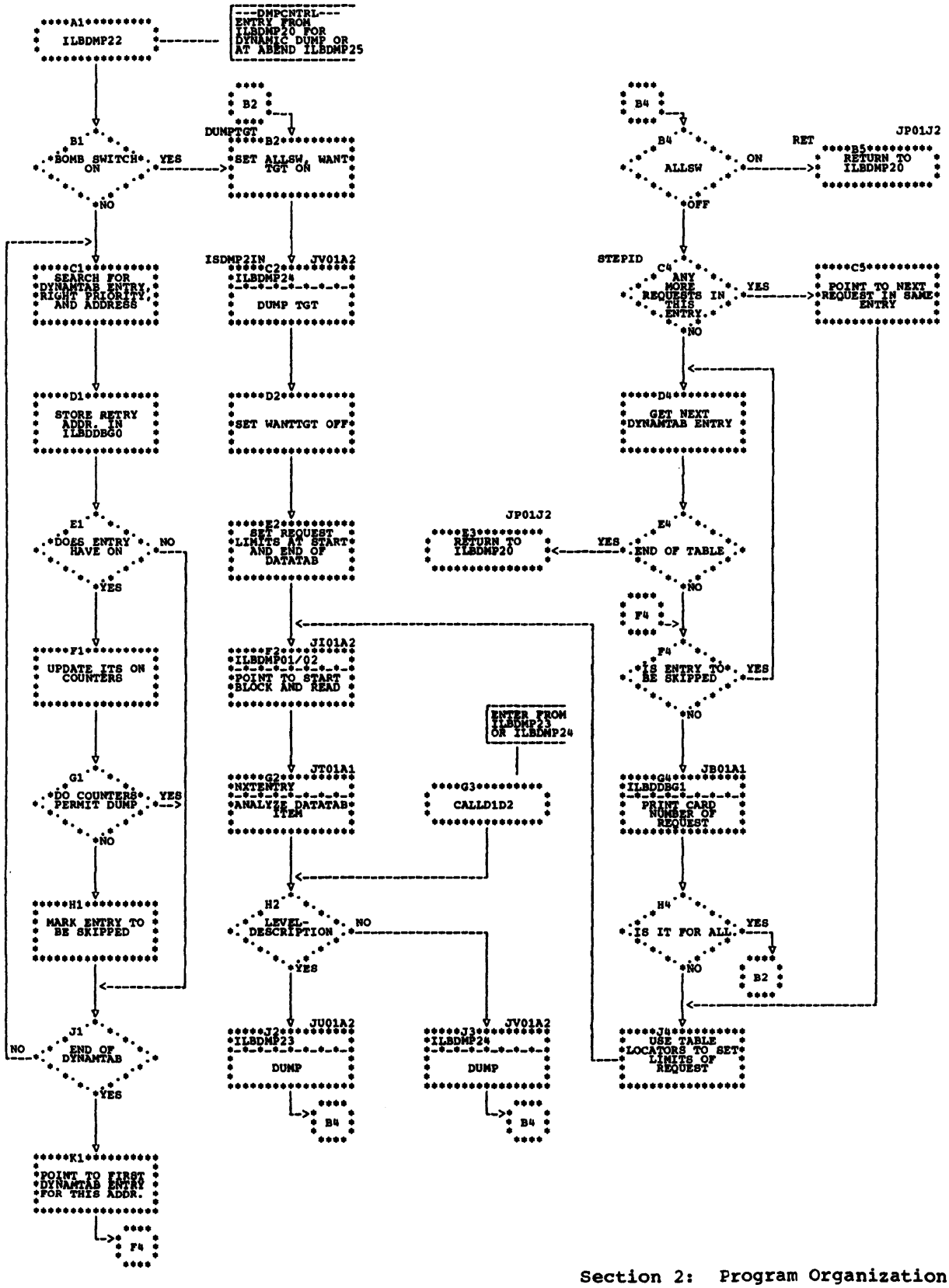


Chart JT. NXTENTRY (ILBDMP22)

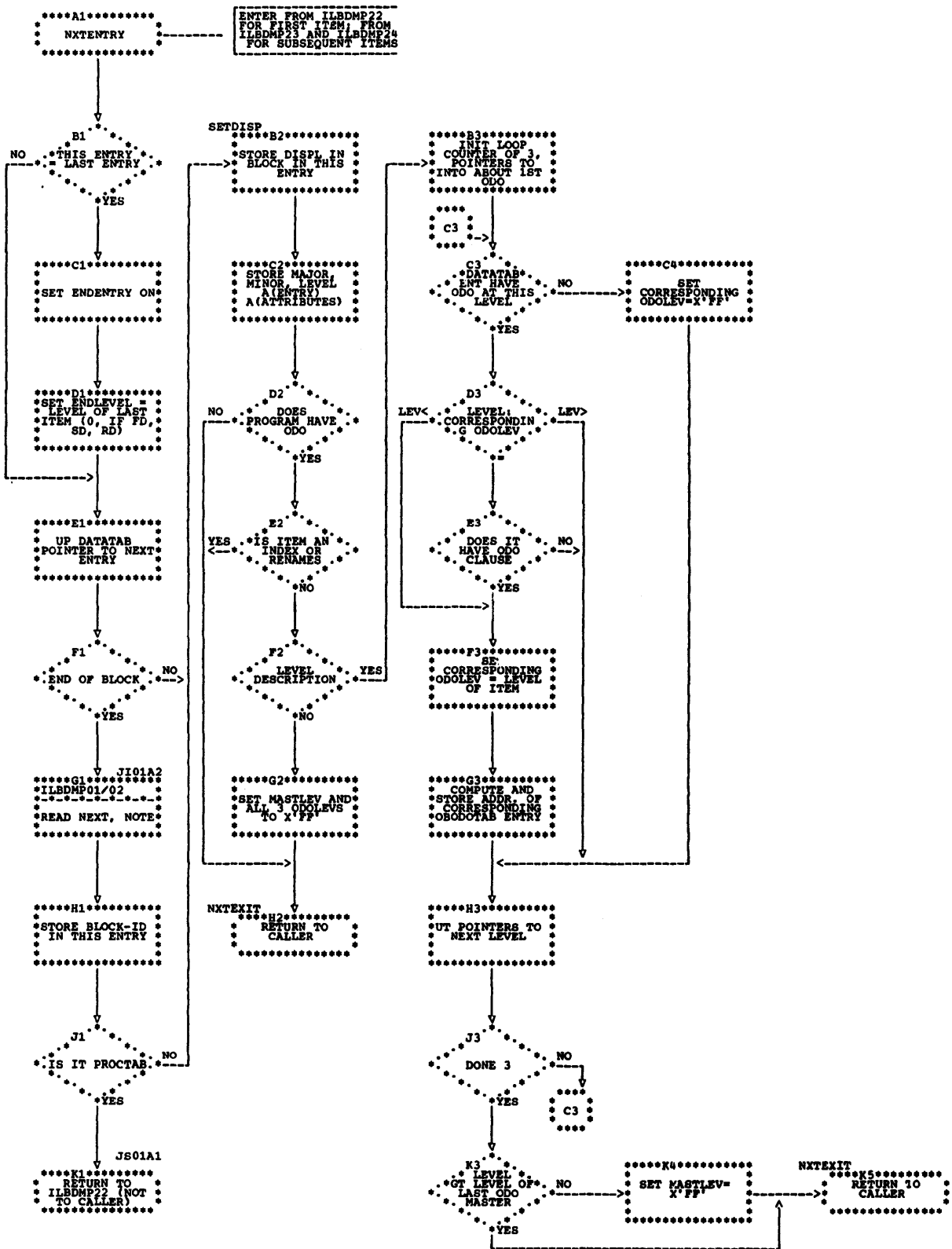


Chart JU. DUMP1 (ILBDMP23)

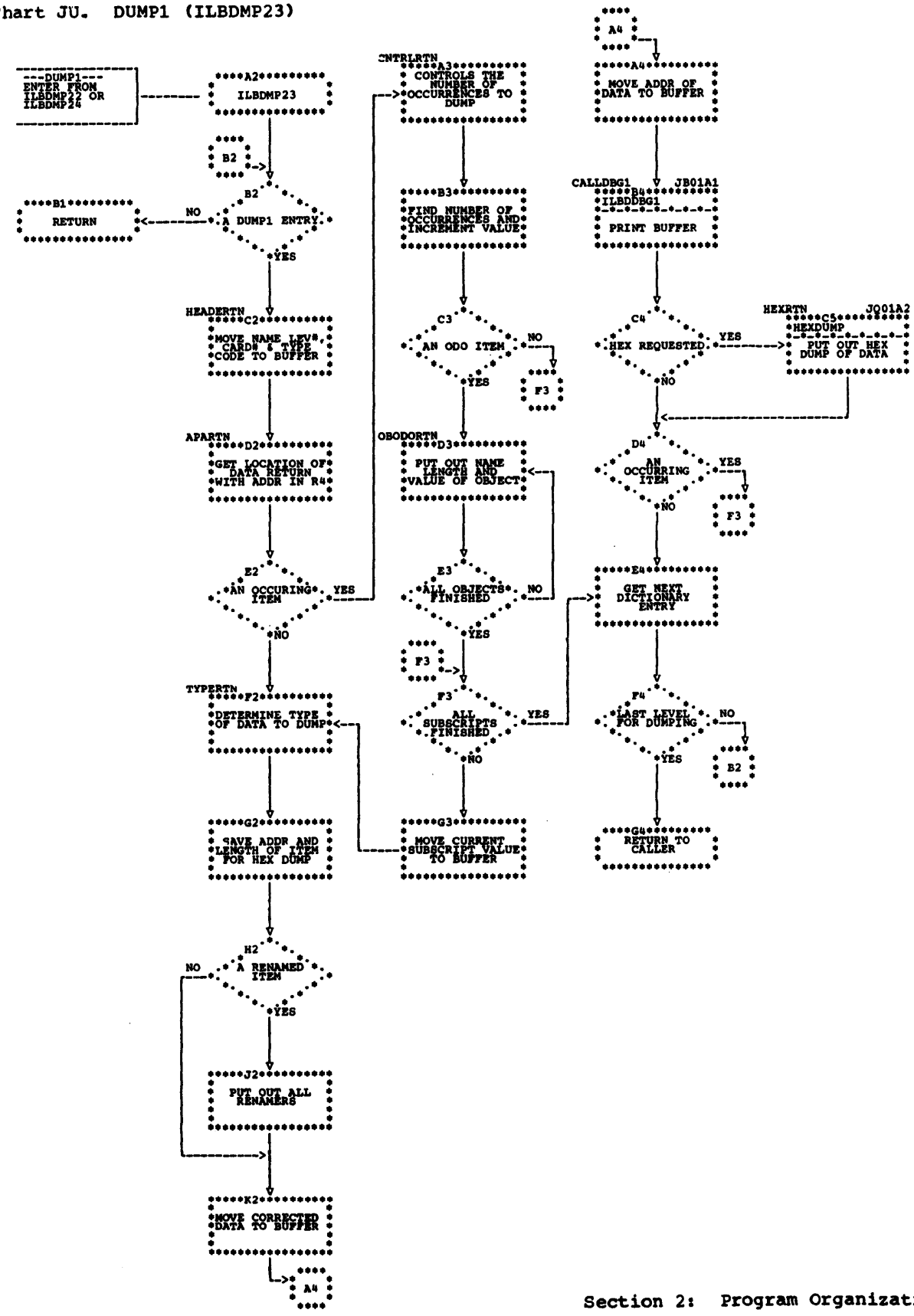
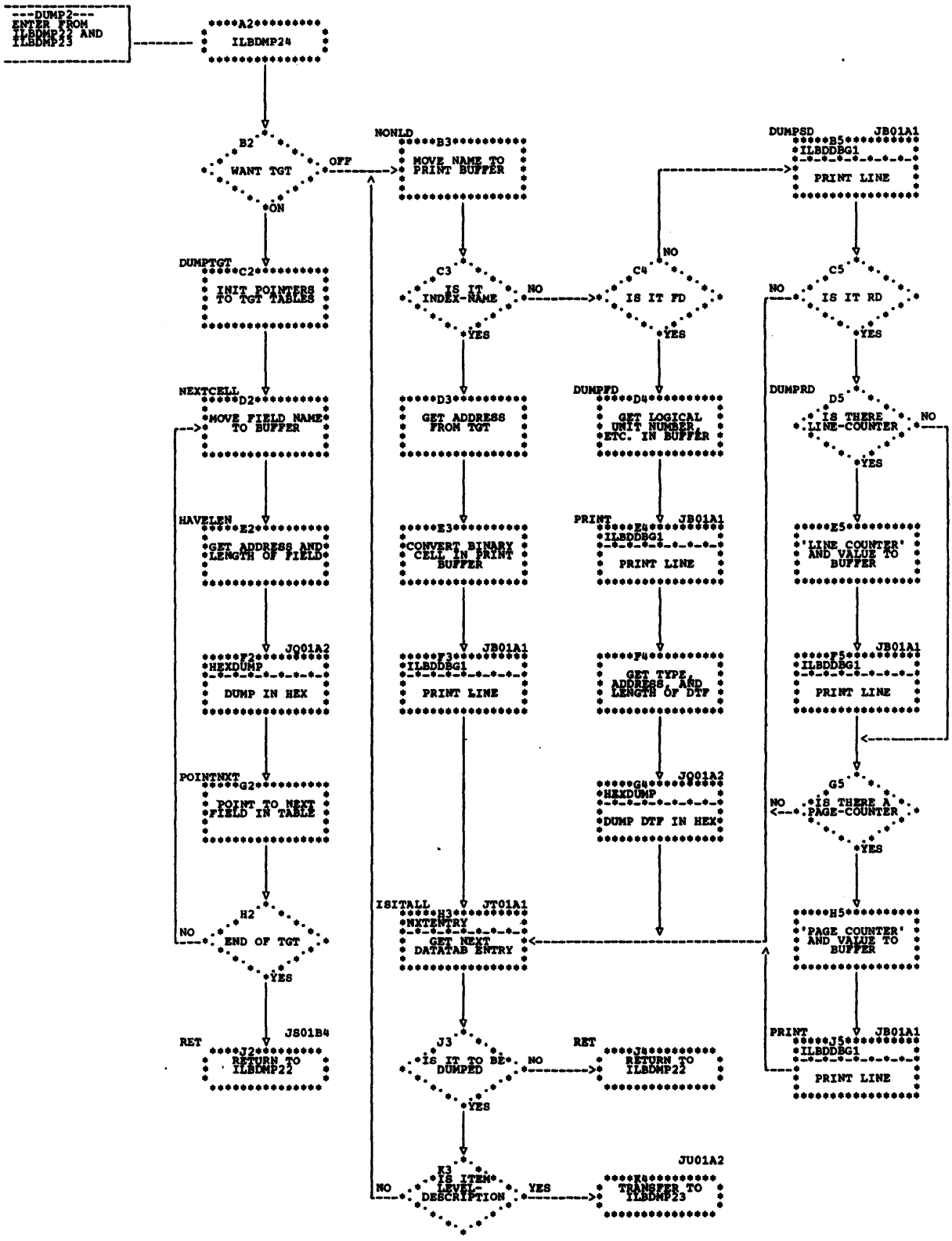


Chart JV. DUMP2 (ILBDMP24)

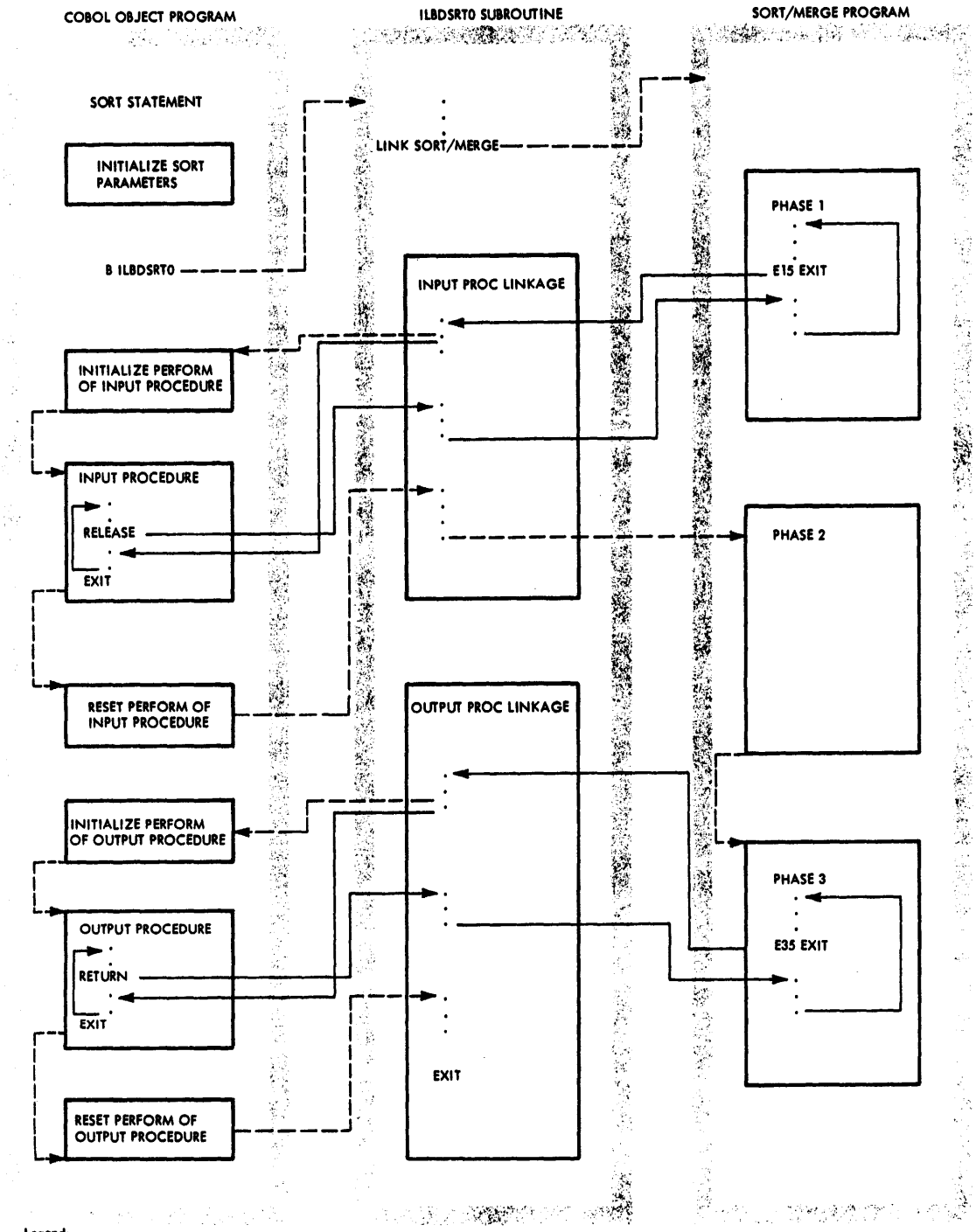


SECTION 2: PROGRAM ORGANIZATION

This section is divided into two parts: "Diagrams" and "Flowcharts". The diagrams describe the flow of control, loading and calling dependencies, and virtual storage layouts in instances where several programs are present together in virtual storage.

Flowcharts are provided for most of the data management subroutines, all of the subroutines for object-time debugging operations, and for other complex subroutines.

DIAGRAMS



Legend:
Broken line arrows indicate logic paths executed only once; solid line arrows represent logic paths in loops.

Diagram 1. ILBDSRT0 Logic Flow For SORT

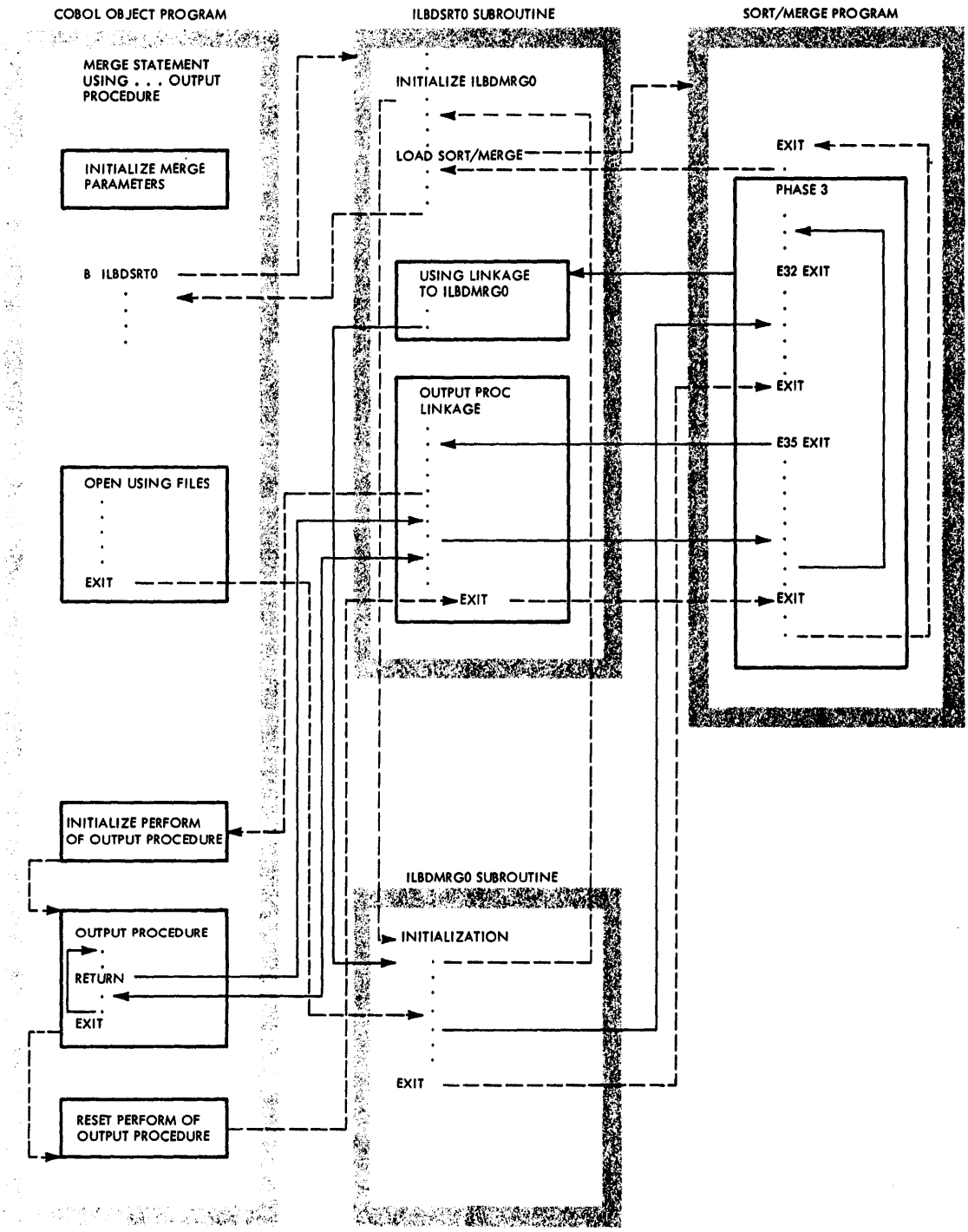


Diagram 2. ILBDSRT0 and IILBDMRG0 Logic Flow For MERGE

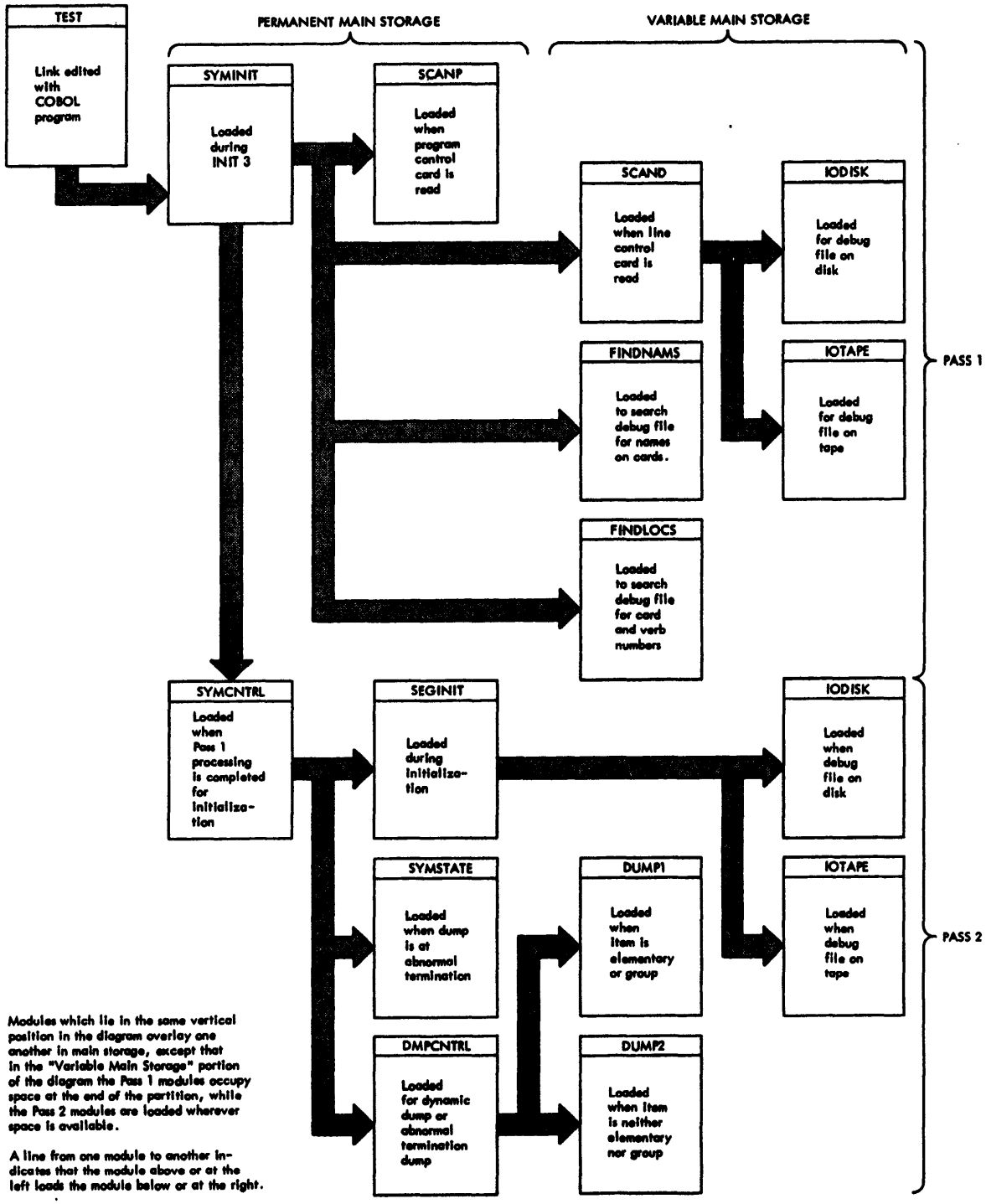


Diagram 3. SYMDMP Subroutines: Loading Dependencies

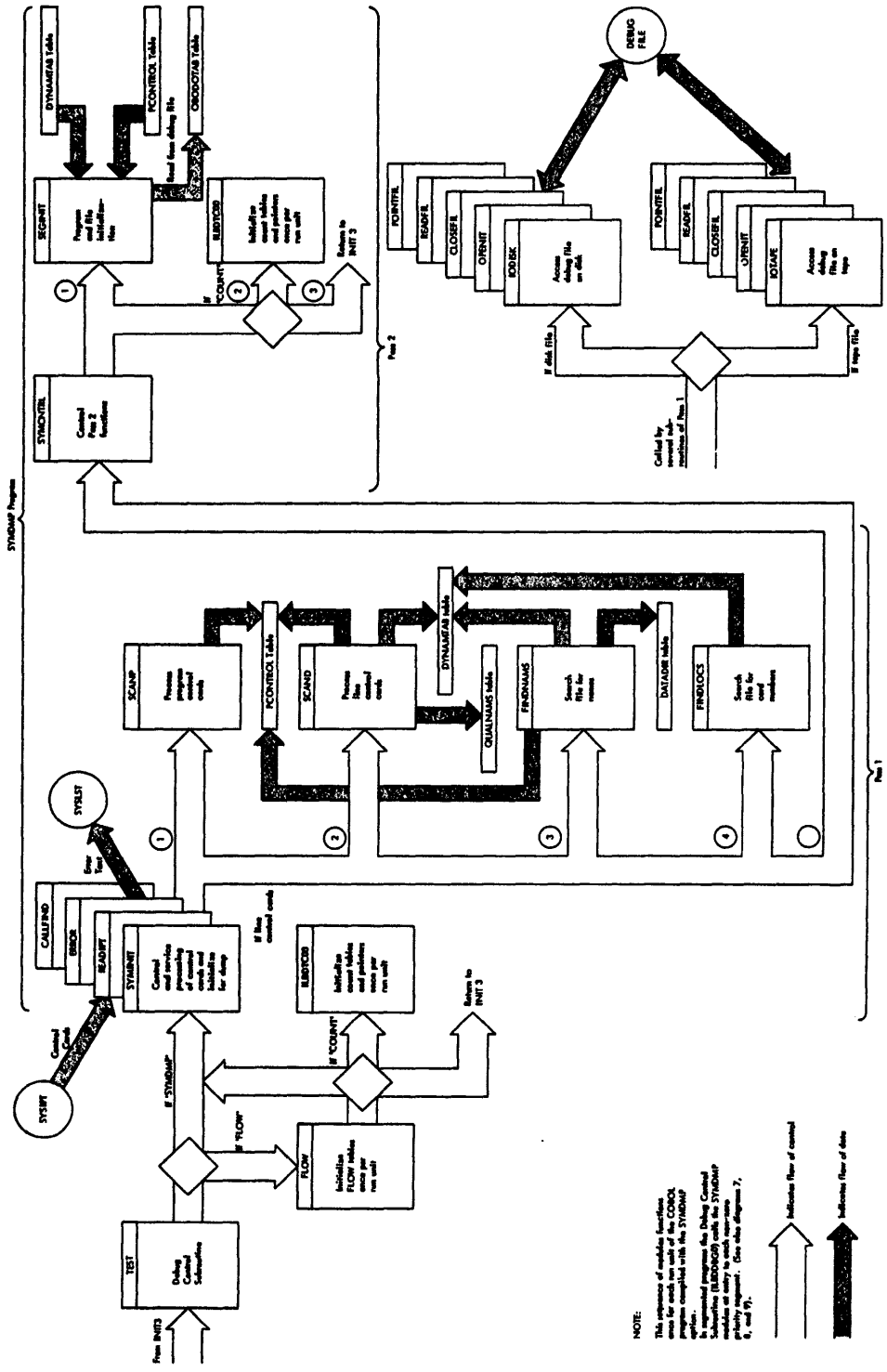


Diagram 4. Debug and Execution Statistics Subroutines: Flow of Control at Initialization

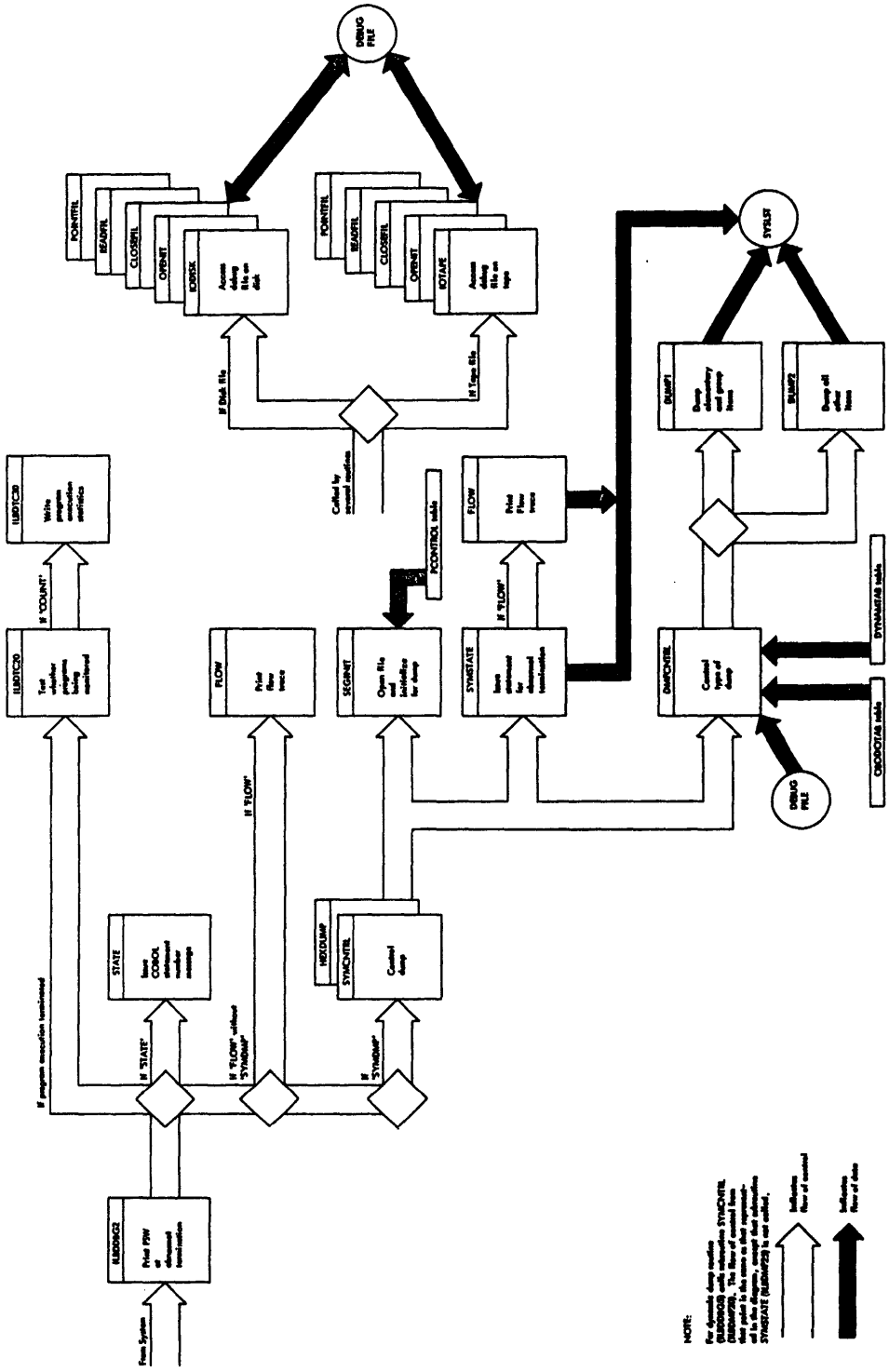


Diagram 5. Debug and Execution Statistics Subroutines: Flow of Control at Abnormal Termination

Routine: ILBDDBG0 -- Level 1			
ROUTINE	PURPOSE	CALLED ROUTINES	CALLING CONDITION
ILBDDBG0	Service 3 options for handling debugging information and the COUNT option for providing object-time execution statistics.	SYMINIT (ILBDMP10)	Called when SYMDMP option switch is on in TGT.
		SYMCNTRL (ILBDMP20)	Called when SYMDMP option switch is on in TGT every time after the first.
		ILBDFLW0	Called if FLOW option is specified.
		ILBDSTN0	Called if STATE option is specified.
		ILBDTC00	Called if COUNT option is specified (see "Object-Time Execution Statistics Subroutines").
		ILBDTC20	Called in all cases.

Diagram 6. Debug and Execution Statistics Subroutines: Calling Dependencies (Part 1 of 4)

Routine: ILBDDBG0 -- Level 2			
ROUTINE	PURPOSE	CALLED ROUTINES	CALLING CONDITION
SYMINIT (ILBDMP10)	Control routine and common subroutines for processing control cards in SYMDMP option.	SCANP (ILBDMP11)	Called if program-control card is found.
		SCAND (ILBDMP12)	Called if line control card is found.
		FINDNAMS (ILBDMP13)	Called if valid line-control cards entered in DYNAMTAB.
		FINDLOCS (ILBDMP14)	Called if valid line-control cards entered in DYNAMTAB.
SYMCNTRL (ILBDMP20)	Control routine for SYMDMP output.	SEGINIT (ILBDMP21)	Called each time a program or segment is entered and at abnormal termination.
		SYMSTATE (ILBDMP25)	Called at abnormal termination to produce a statement number message.
		DMPCNTRL (ILBDMP22)	Called whenever a dump is to be produced.
ILBDFLW0	Produce flow trace if FLOW is specified.	Calls no further routines.	
ILBDSTN0	Write statement number if STATE is specified message at abnormal termination.	Calls no further routines.	
ILBDTC00	Initialize COUNT statistics if COUNT specified.	Calls no further routines.	
ILBDTC20	Produce COUNT statistics if COUNT specified.	ILBDTC30	Called if COUNT specified.

Diagram 6. Debug and Execution Statistics Subrotines : Calling Dependencies (Part 2 of 4)

Routine: ILBDDBG0 -- Level 3			
ROUTINE	PURPOSE	CALLED ROUTINES	CALLING CONDITION
SCAND (ILBDMP11)	Processes program control cards.	Calls no further routines.	
SCAND (ILBDMP12)	Processes line control cards.	IODISK (ILBDMP01)	Called when Debug File is on disk.
		IOTAPE (ILBDMP02)	Called when Debug File is on tape.
FINDNAMS (ILBDMP13)	Searches Debug File for identifiers requested on line control cards; enters locators for them in DYNAMTAB.	IODISK (ILBDMP01)	Debug File on disk.
		IOTAPE (ILBDMP02)	Debug File on tape.
FINDLOCS (ILBDMP14)	Searches Debug File for card number information; enters it in DYNAMTAB.	IODISK (ILBDMP01)	Debug File on disk.
		IOTAPE (ILBDMP02)	Debug File on tape.
SEGINIT (ILBDMP21)	Initializes program segment for dynamic dumping by modifying specified instructions; allocates space; relocates table addresses; opens debug file.	IODISK (ILBDMP01)	Debug File on disk.
		IOTAPE (ILBDMP02)	Debug File on tape.
SYMSTATE (ILBDMP25)	Issues the abnormal termination statement number message.	IODISK (ILBDMP01)	Debug File on disk.
		IOTAPE (ILBDMP02)	Debug File on tape.
DMPCNTRL (ILBDMP22)	Contains main loop controlling dump.	DUMP1 (ILBDMP23)	Called when group or elementary items are to be dumped.
		DUMP2 (ILBDMP24)	Called when item to be dumped is neither group nor elementary.
		IODISK (ILBDMP01)	Debug File on disk.
		IOTAPE (ILBDMP02)	Debug File on tape.
ILBDTC30	Print the COUNT statistics.	Calls no further routines.	

Diagram 6. Debug and Execution Statistics Subroutines: Calling Dependencies (Part 3 of 4)

Routine: ILBDDBG0 -- Level 4			
DUMP1 (ILBDMP23)	Dumps elementary and group level items	IODISK (ILBDMP01) IOTAPE (ILBDMP02)	Debug File on disk. Debug File on tape.
DUMP2 ILBDMP24)	Dumps item which are neither elementary or group level items.	IODISK (ILBDMP01) IOTAPE (ILBDMP02)	Debug File on disk. Debug File on tape.

Routine: ILBDDBG0 -- Level 5			
IODISK (ILBDMP01)	Performs input/output operations for debug file on disk.	Calls ILBDMP04 before each open.	
IOTAPE (ILEDMP02)	Performs input/output operations for debug file on tape.	Calls no further routines.	
SRCHPUBS (ILBDMP04)	Performs initialization of SYS005 DTF for disk debug file.	Calls no further routines.	Called by IODISK for each open.

Diagram 6. Debug and Execution Statisitics Subroutines: Calling Dependencies (Part 4 of 4)

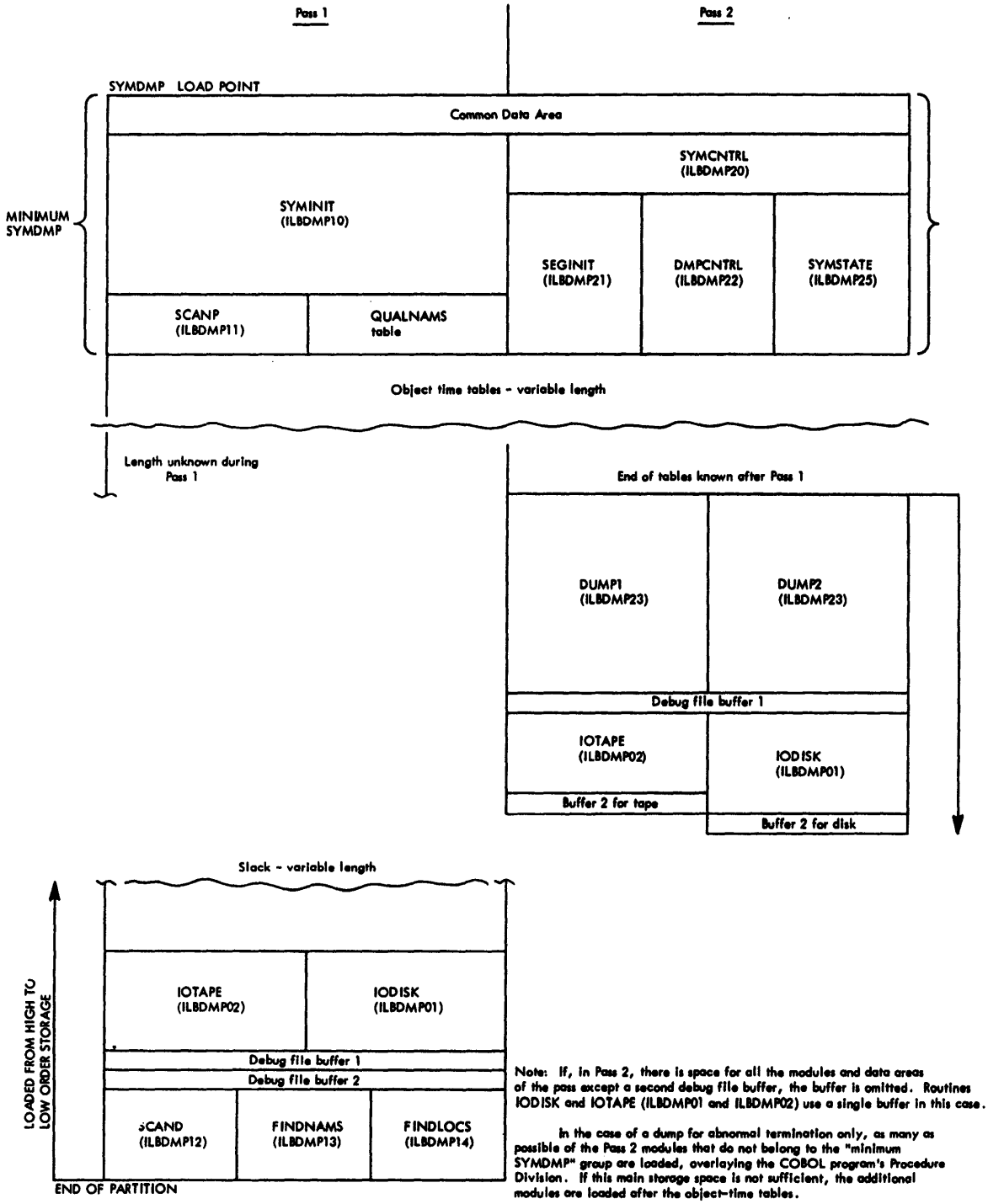
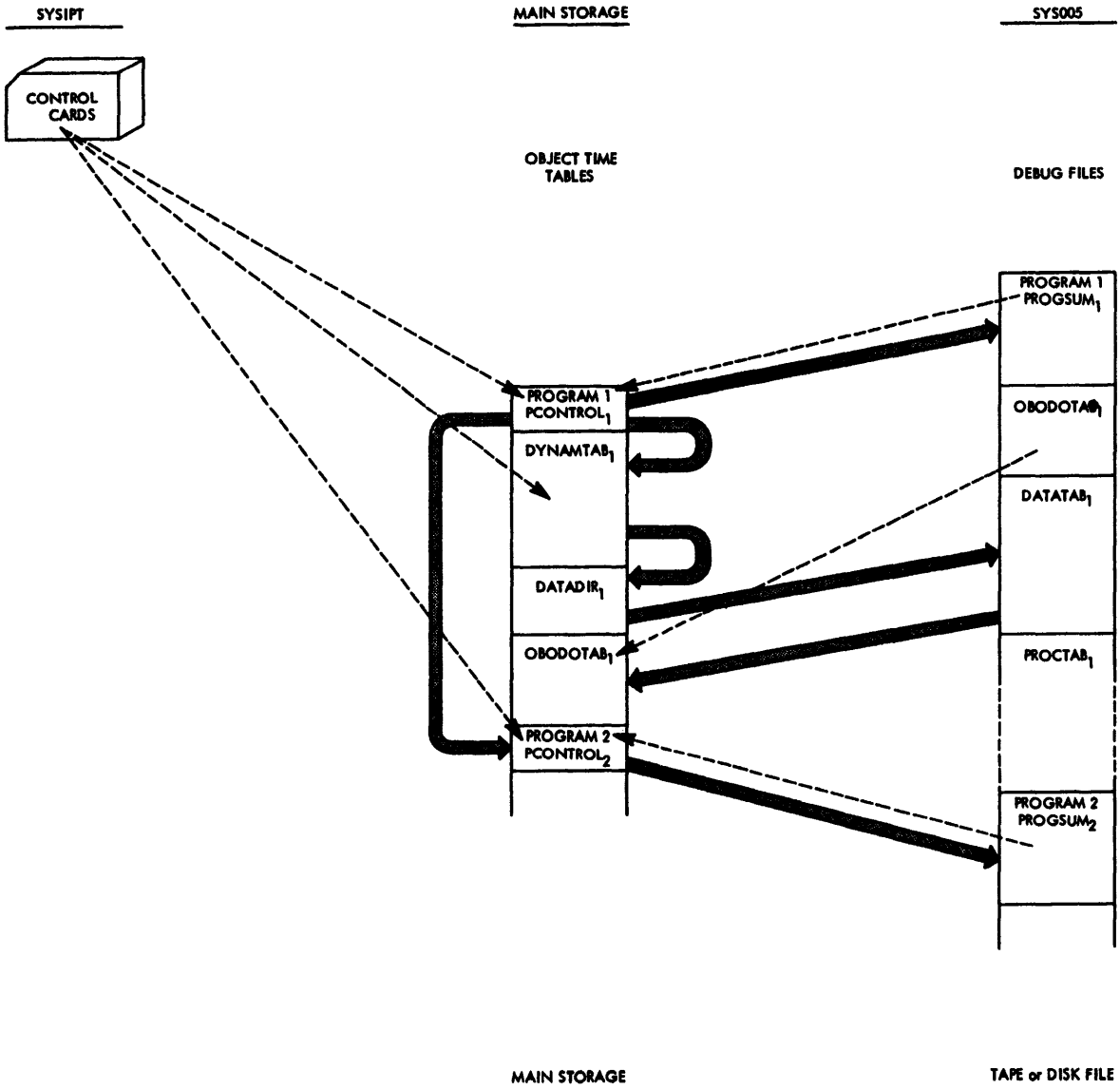


Diagram 7. Virtual Storage Layout of SYMDMP Modules



NOTE:

Solid arrows indicate the main pointers connecting the tables.
Broken arrows indicate the primary sources of information.
Broken lines indicate the boundary between files.

Diagram 8 SYMDMP Subroutines: Control Card Processing. Relation Between Object-Time Tables and Debug File in Processing Identifiers on Control Cards

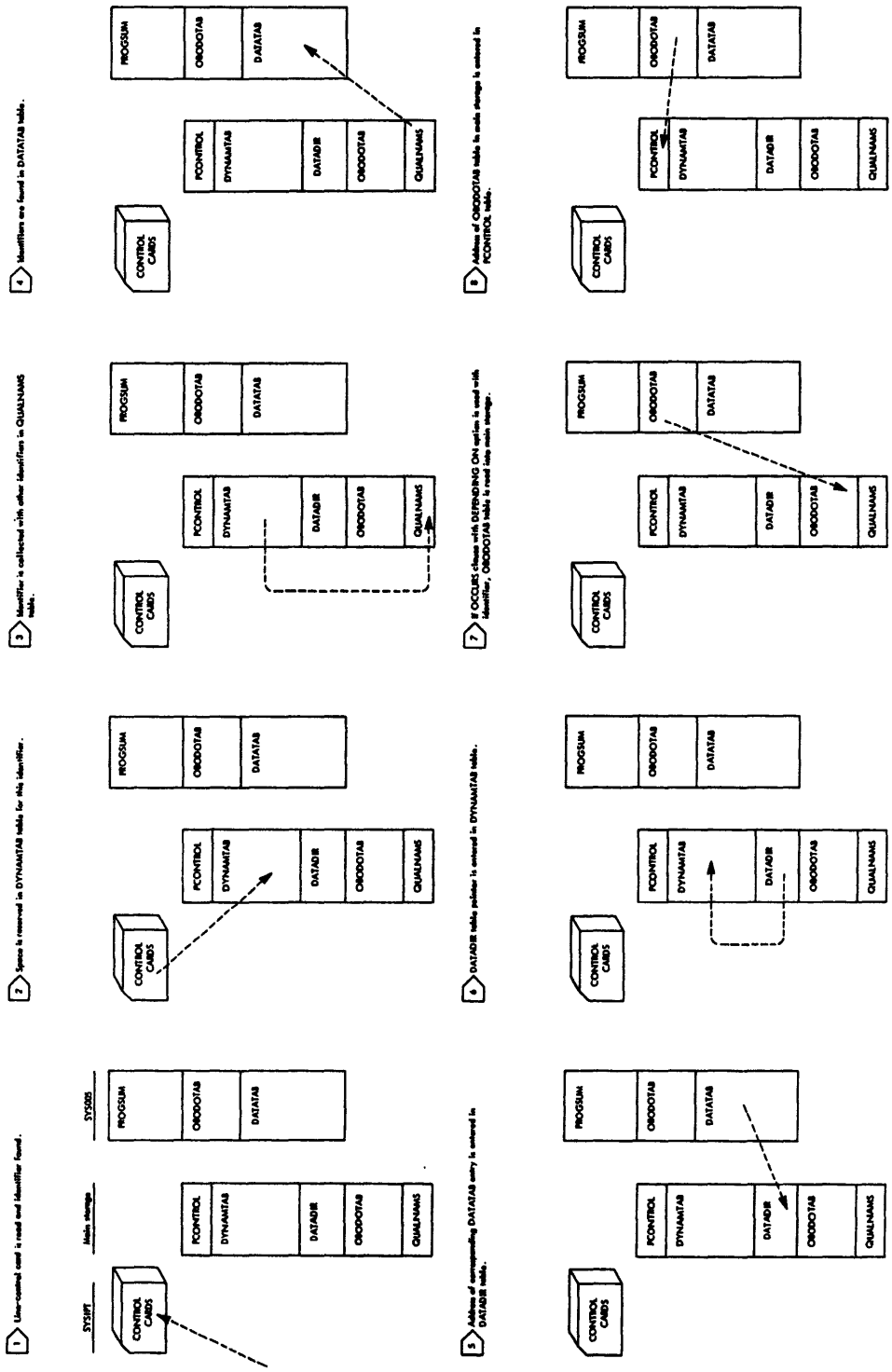


Diagram 9. SYMDMP Subroutines: Control Card Processing. Identifier Processing

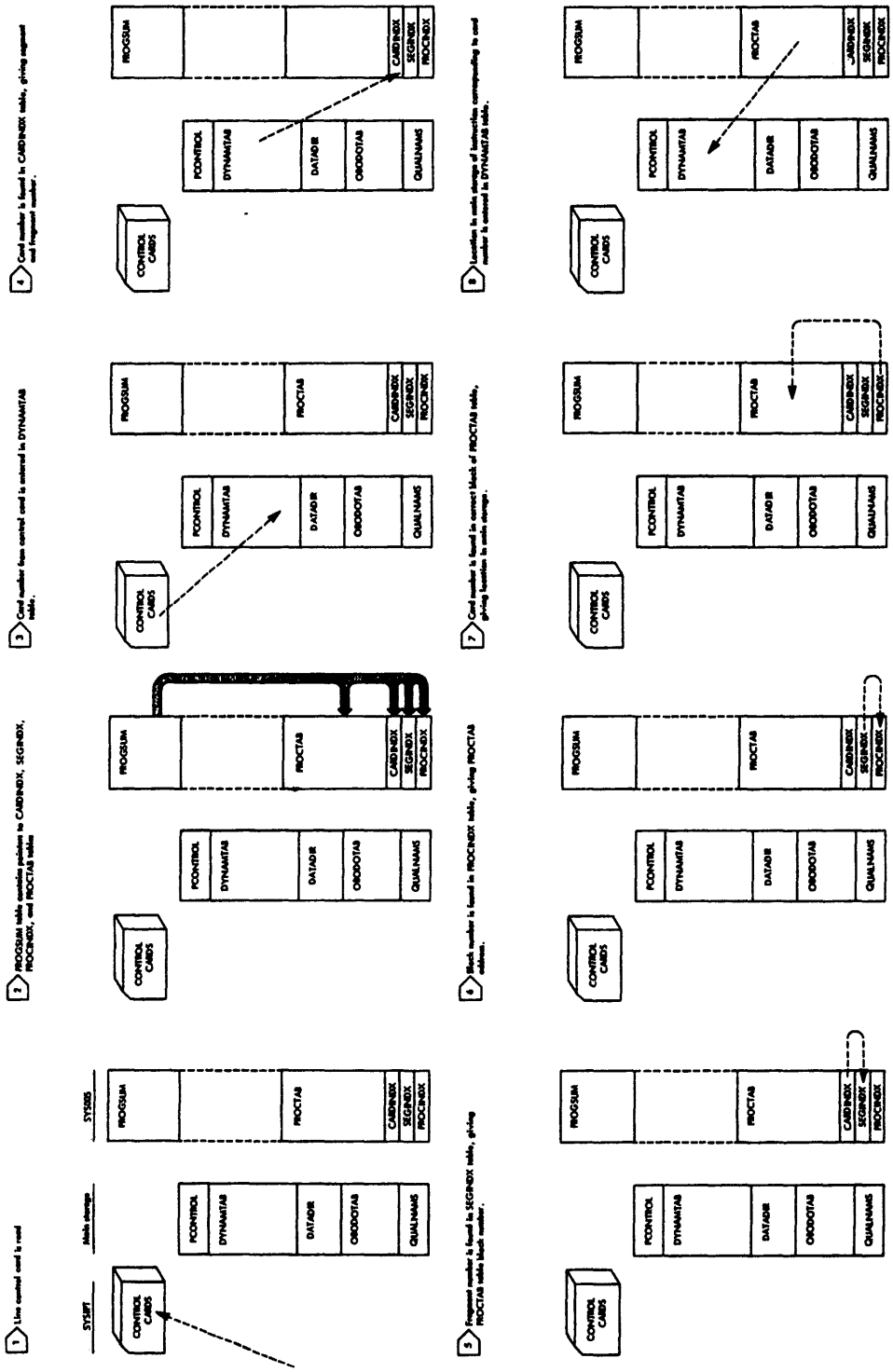
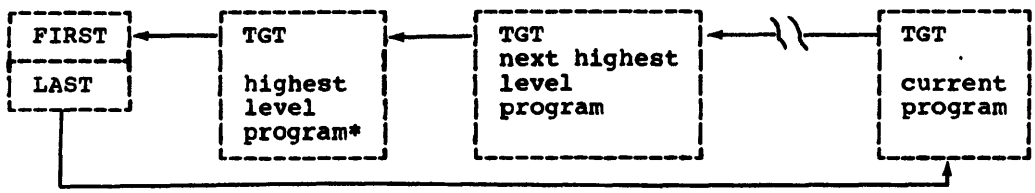


Diagram 10. SYMDMP Subroutines: Control Card Processing. Card Number Processing



When SYMDMP, STATE, or FLOW has been requested, the TGT Address Routine (ILBDDBG3) is called by the COBOL program at each return of control to the program after a branch outside of itself. The routine stores in the fullword LAST the address of the current TGT. At abnormal termination this data area is used by the STXIT routine (ILBDDBG2) to trace the calling programs of an interrupted program so that information may be provided for each of them. Tracing begins at the program whose TGT is stored in data area LAST; it ends at the program whose TGT is stored in data area FIRST.

*Compiled with the SYMDMP, STATE, or FLOW options.

Diagram 11. Doubleword Data Area Used by the TGT Address (ILBDDBG3) and STXIT (ILBDDBG2) Routines of the Debug Control Subroutine

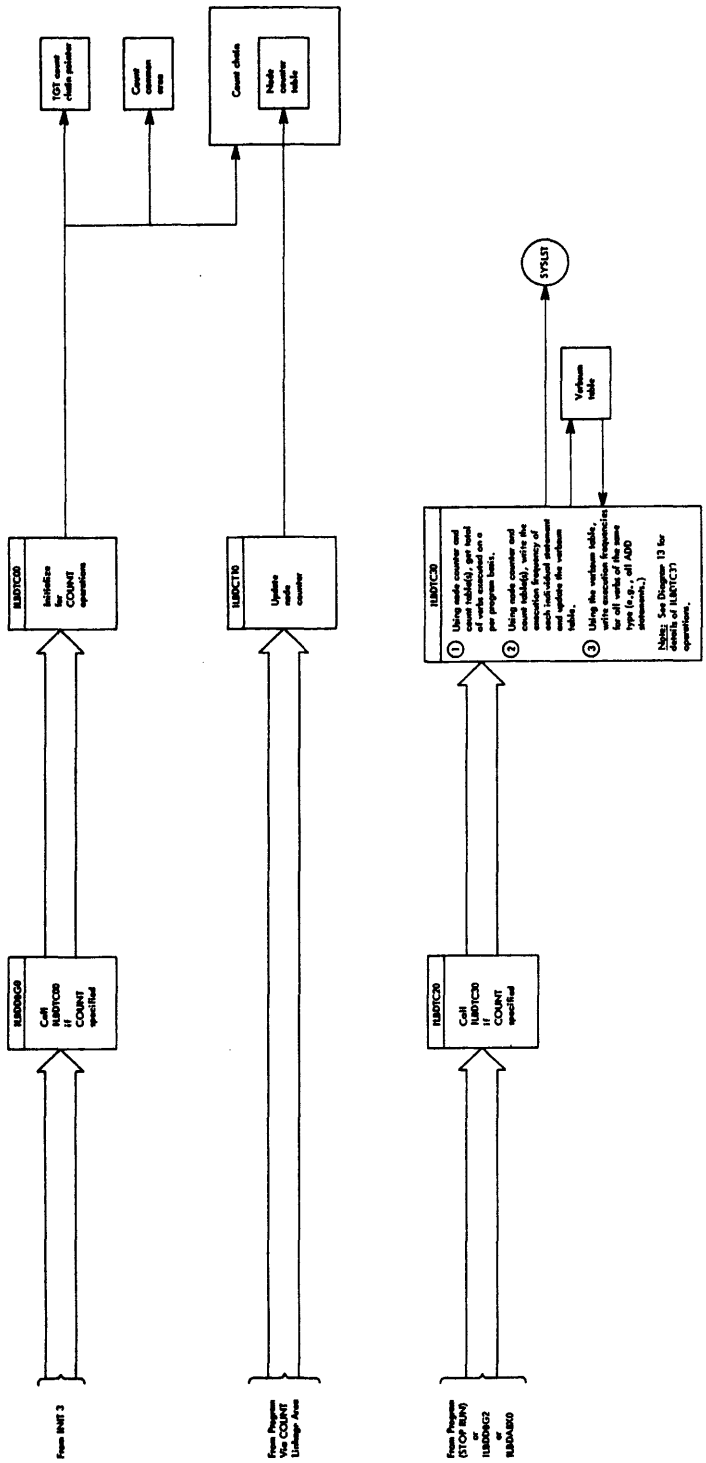


Diagram 12. Overall Processing for Producing Object-Time Execution Statistics

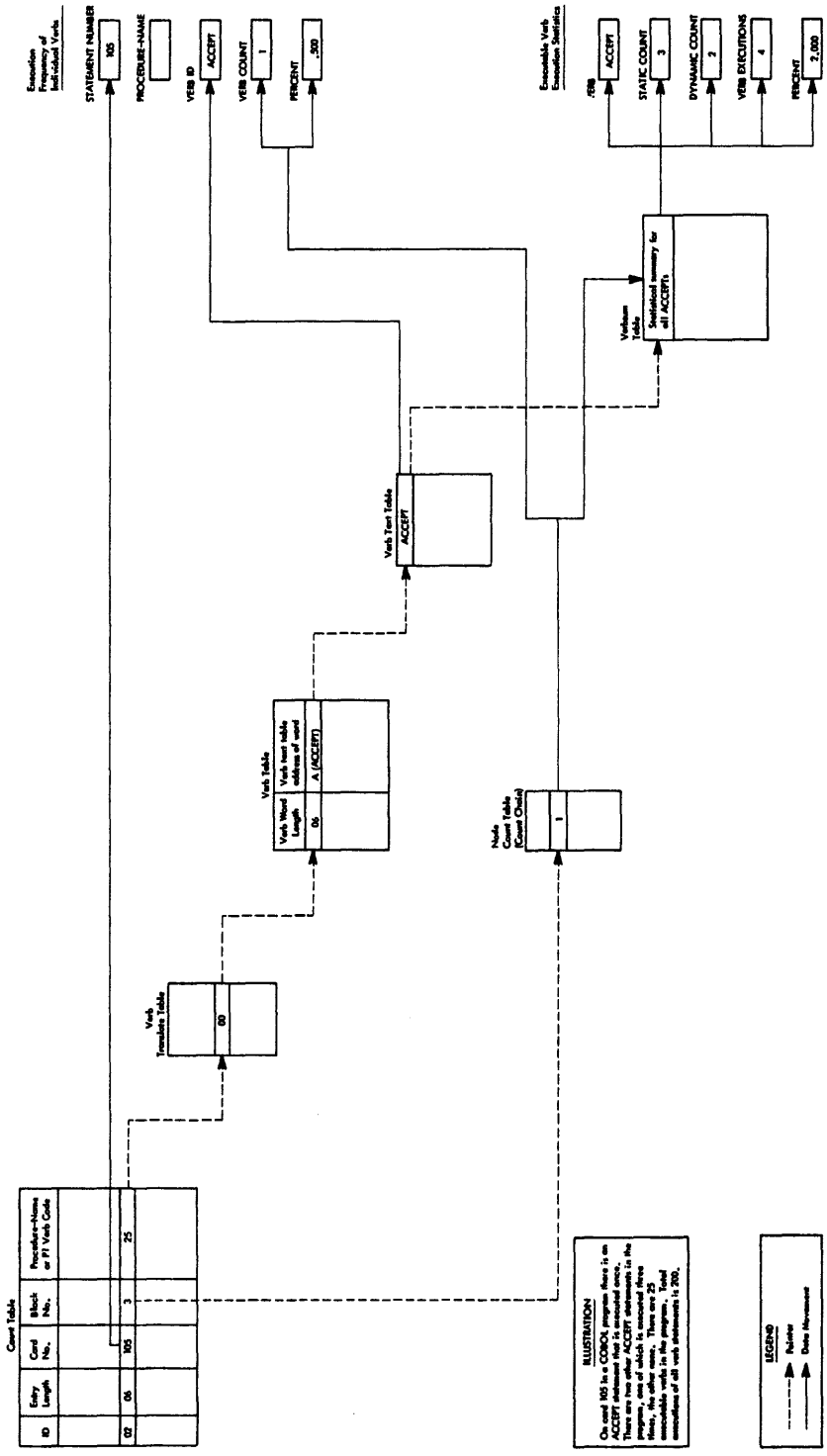


Diagram 13. How Tables Are Used to Produce Object-Time Execution Statistics

FLOWCHARTS

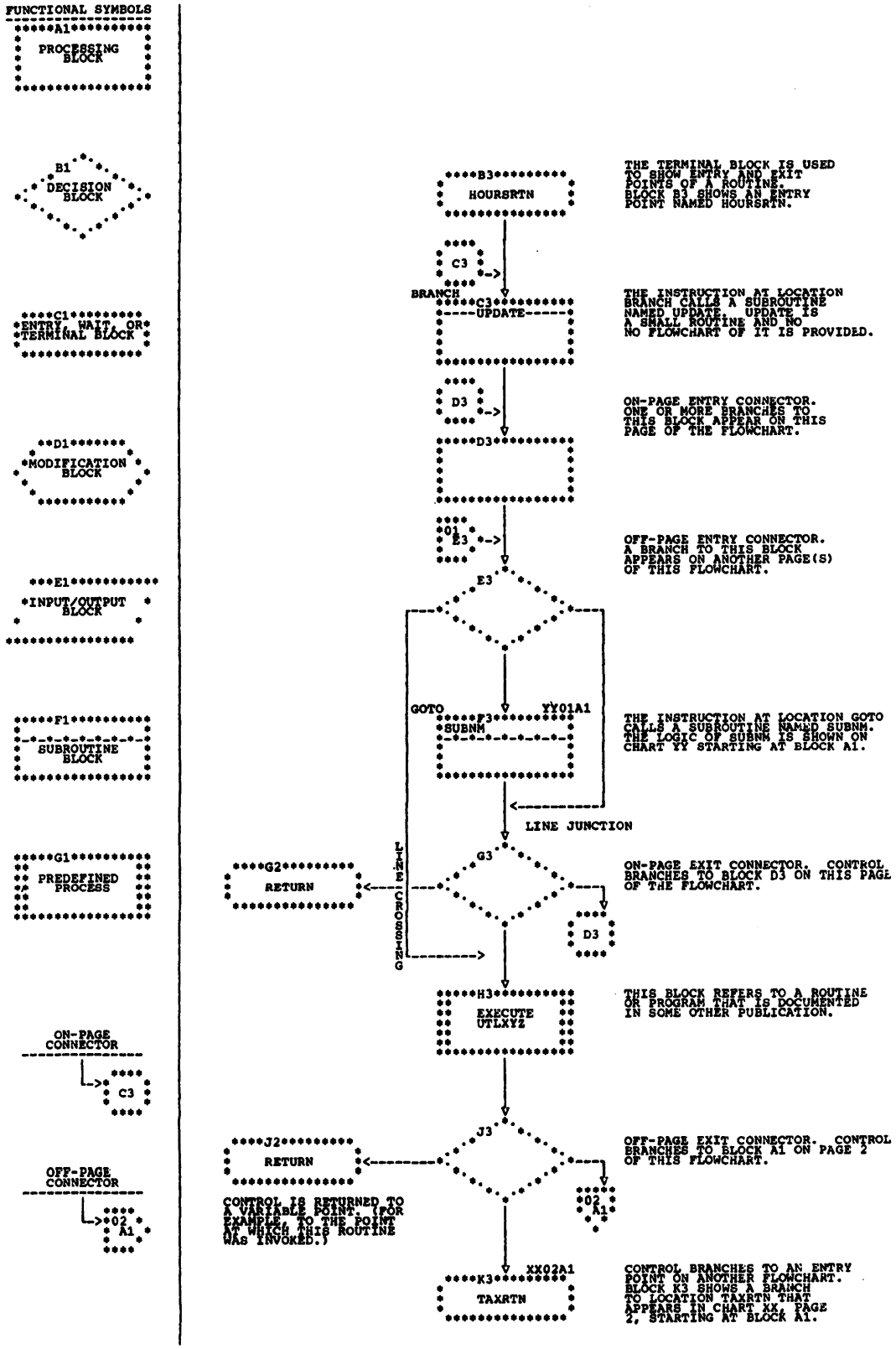

FUNCTIONAL SYMBOLS
A1
PROCESSING BLOCK
B1
DECISION BLOCK
C1
ENTRY, WAIT, OR TERMINAL BLOCK
D1
MODIFICATION BLOCK
E1
INPUT/OUTPUT BLOCK
F1
SUBROUTINE BLOCK
G1
PREDEFINED PROCESS
ON-PAGE CONNECTOR
C3
OFF-PAGE CONNECTOR
02 A1
B3
HOURSRTN
THE TERMINAL BLOCK IS USED TO SHOW ENTRY AND EXIT POINTS OF A ROUTINE. BLOCK B3 SHOWS AN ENTRY POINT NAMED HOURSRTN.
C3
BRANCH
C3
UPDATE
THE INSTRUCTION AT LOCATION BRANCH CALLS A SUBROUTINE NAMED UPDATE. UPDATE IS A SMALL ROUTINE AND NO NO FLOWCHART OF IT IS PROVIDED.
D3
D3
ON-PAGE ENTRY CONNECTOR. ONE OR MORE BRANCHES TO THIS BLOCK APPEAR ON THIS PAGE OF THE FLOWCHART.
01 E3
OFF-PAGE ENTRY CONNECTOR. A BRANCH TO THIS BLOCK APPEARS ON ANOTHER PAGE(S) OF THIS FLOWCHART.
E3
GOTO
F3
SUBNM
YY01A1
THE INSTRUCTION AT LOCATION GOTO CALLS A SUBROUTINE NAMED SUBNM. THE LOGIC OF SUBNM IS SHOWN ON CHART YY STARTING AT BLOCK A1.
LINE JUNCTION
G3
G2
RETURN
ON-PAGE EXIT CONNECTOR. CONTROL BRANCHES TO BLOCK D3 ON THIS PAGE OF THE FLOWCHART.
D3
H3
EXECUTE UTLXYZ
THIS BLOCK REFERS TO A ROUTINE OR PROGRAM THAT IS DOCUMENTED IN SOME OTHER PUBLICATION.
J3
J2
RETURN
OFF-PAGE EXIT CONNECTOR. CONTROL BRANCHES TO BLOCK A1 ON PAGE 2 OF THIS FLOWCHART.
02 A1
CONTROL IS RETURNED TO A VARIABLE POINT. (FOR EXAMPLE, TO THE POINT AT WHICH THIS ROUTINE WAS INVOKED.)
K3
XX02A1
TAXRTN
CONTROL BRANCHES TO AN ENTRY POINT ON ANOTHER FLOWCHART. BLOCK K3 SHOWS A BRANCH TO LOCATION TAXRTN THAT APPEARS IN CHART XX, PAGE 2, STARTING AT BLOCK A1.

Chart AA. Decimal to Binary (ILBDCVB0) and Binary to Decimal (ILBDCVB1) (Part 1 of 3)

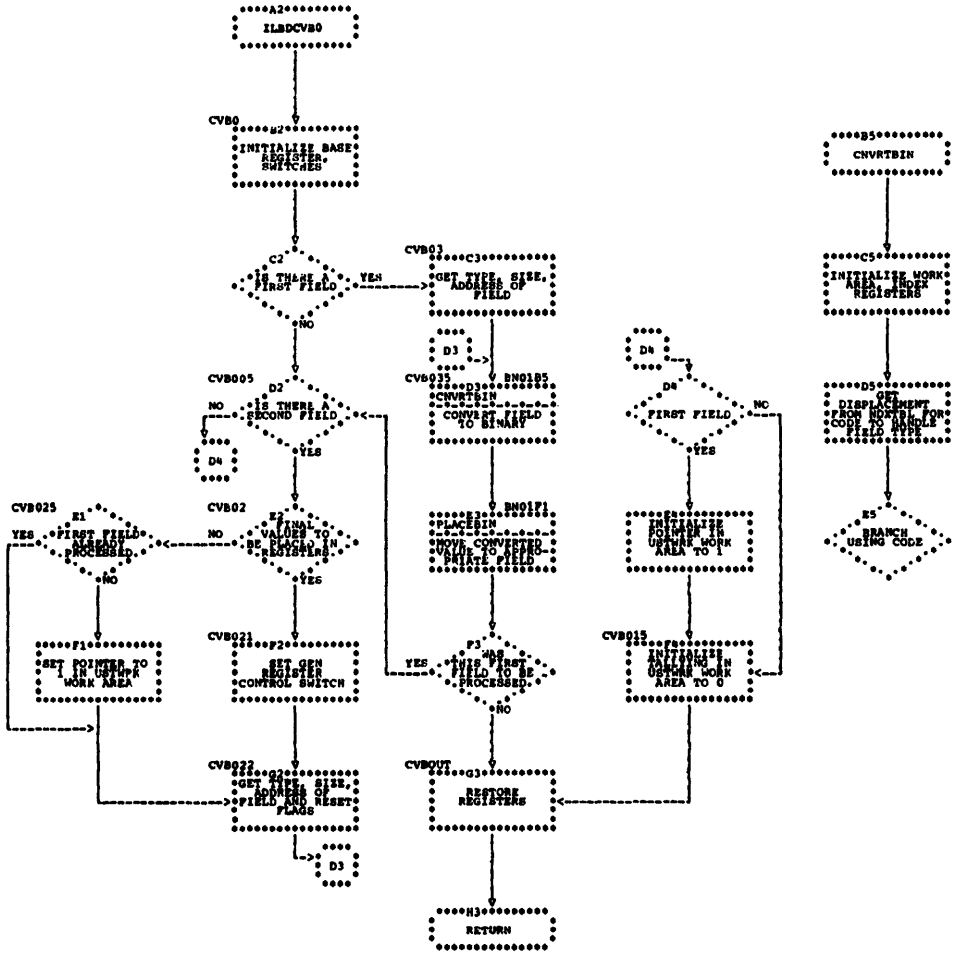


Chart AA. Decimal to Binary (ILBDCVB0) and Binary to Decimal (ILBDCVB1) (Part 2 of 3)

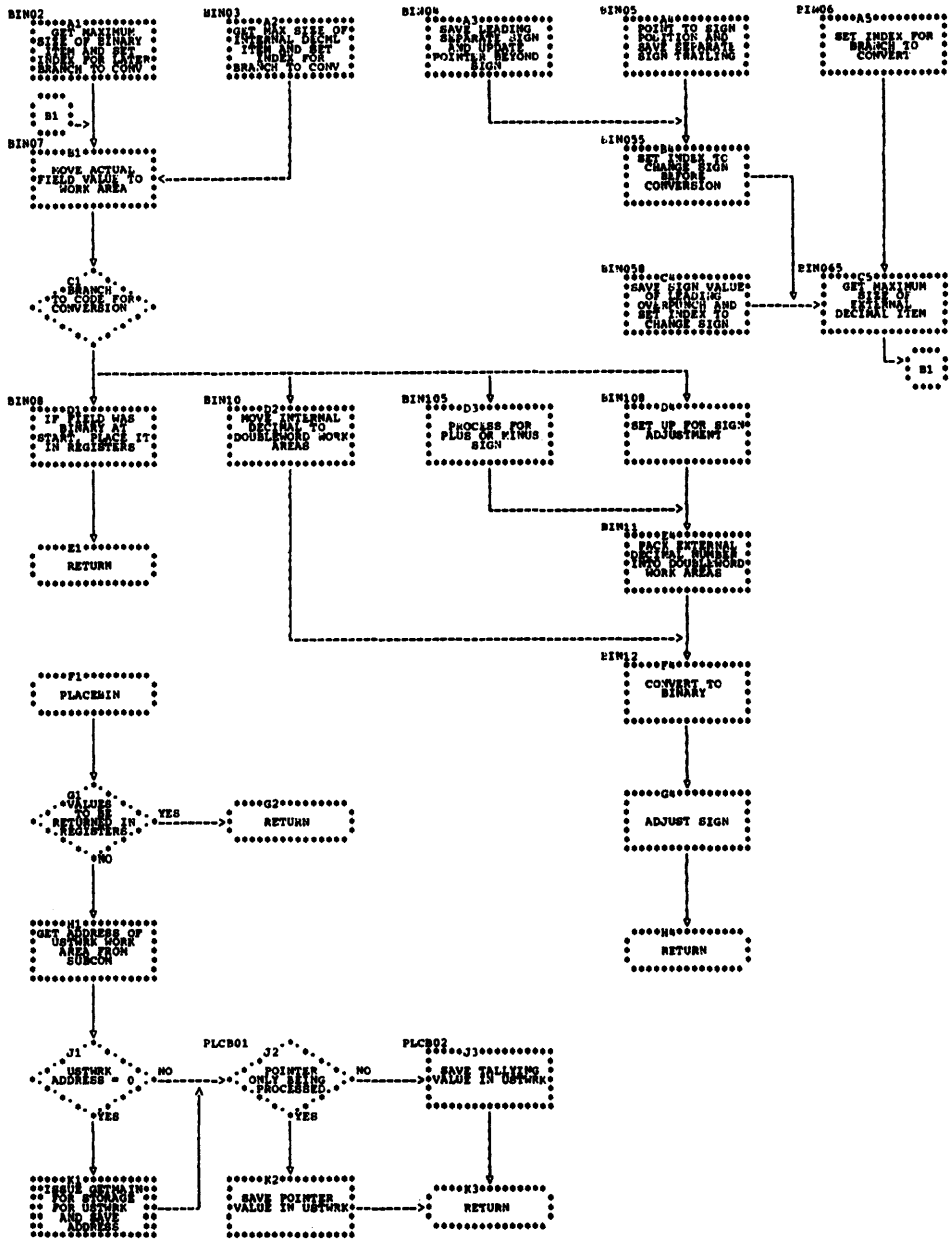


Chart AA. Decimal to Binary (ILBDCVB0) and Binary to Decimal (ILBDCVB1) (Part 3 of 3)

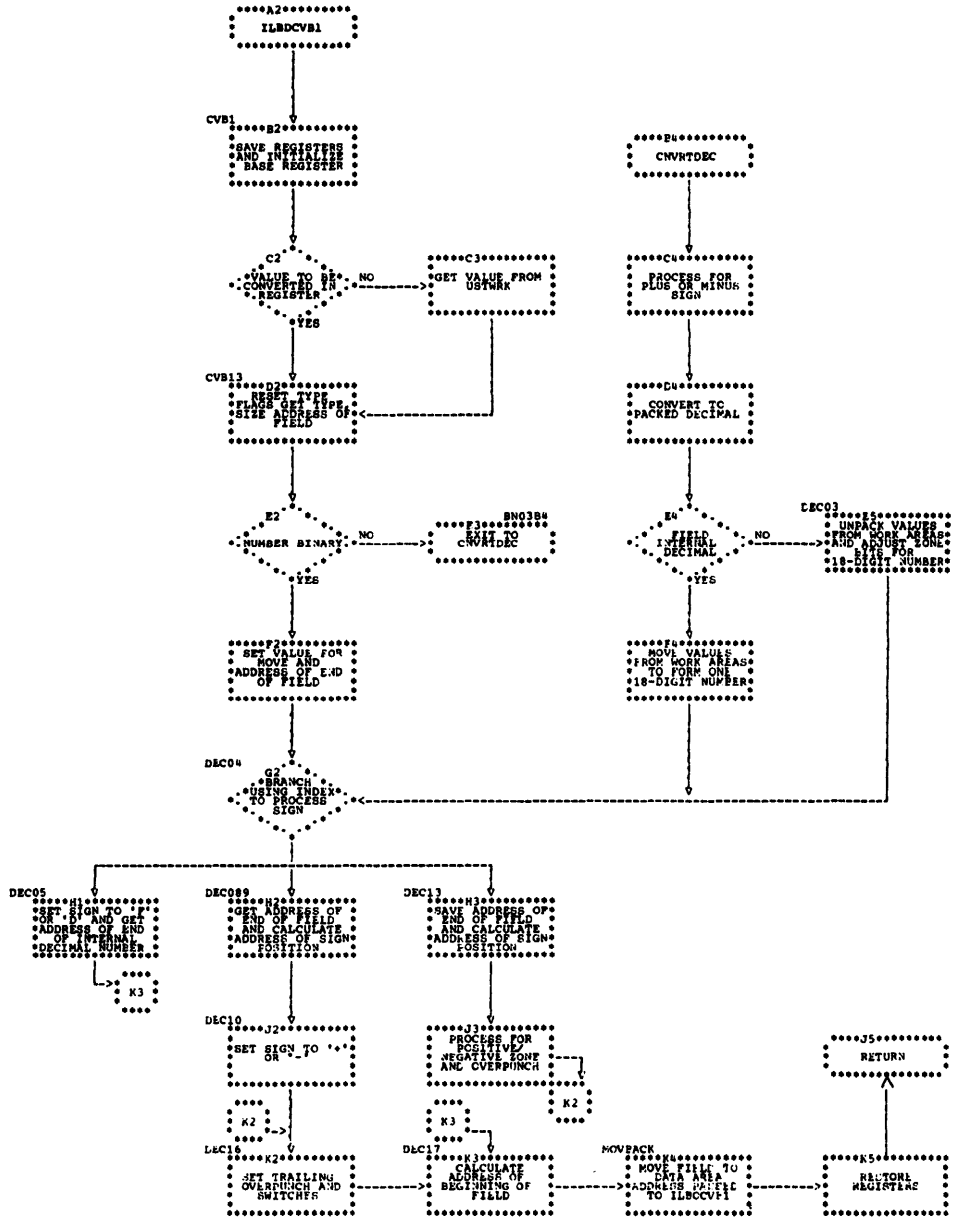


Chart CA. Sort/Merge (ILBDSRT0, ILBDMRG0) (Part 1 of 5): Main Routine

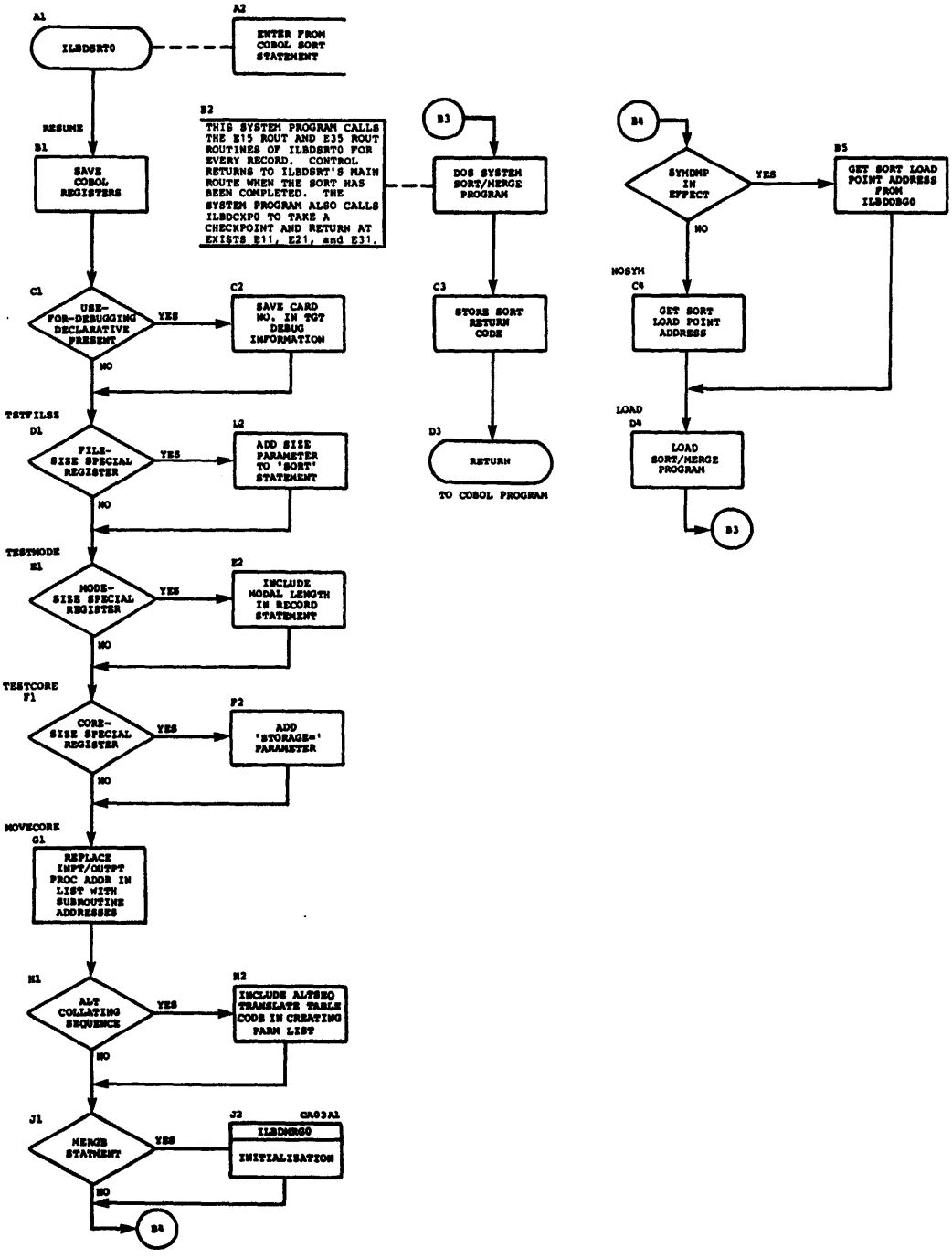


Chart CA. Sort/Merge (ILBDSRT0, ILBDMRG0) (Part 2 of 5): E15ROUT Routine

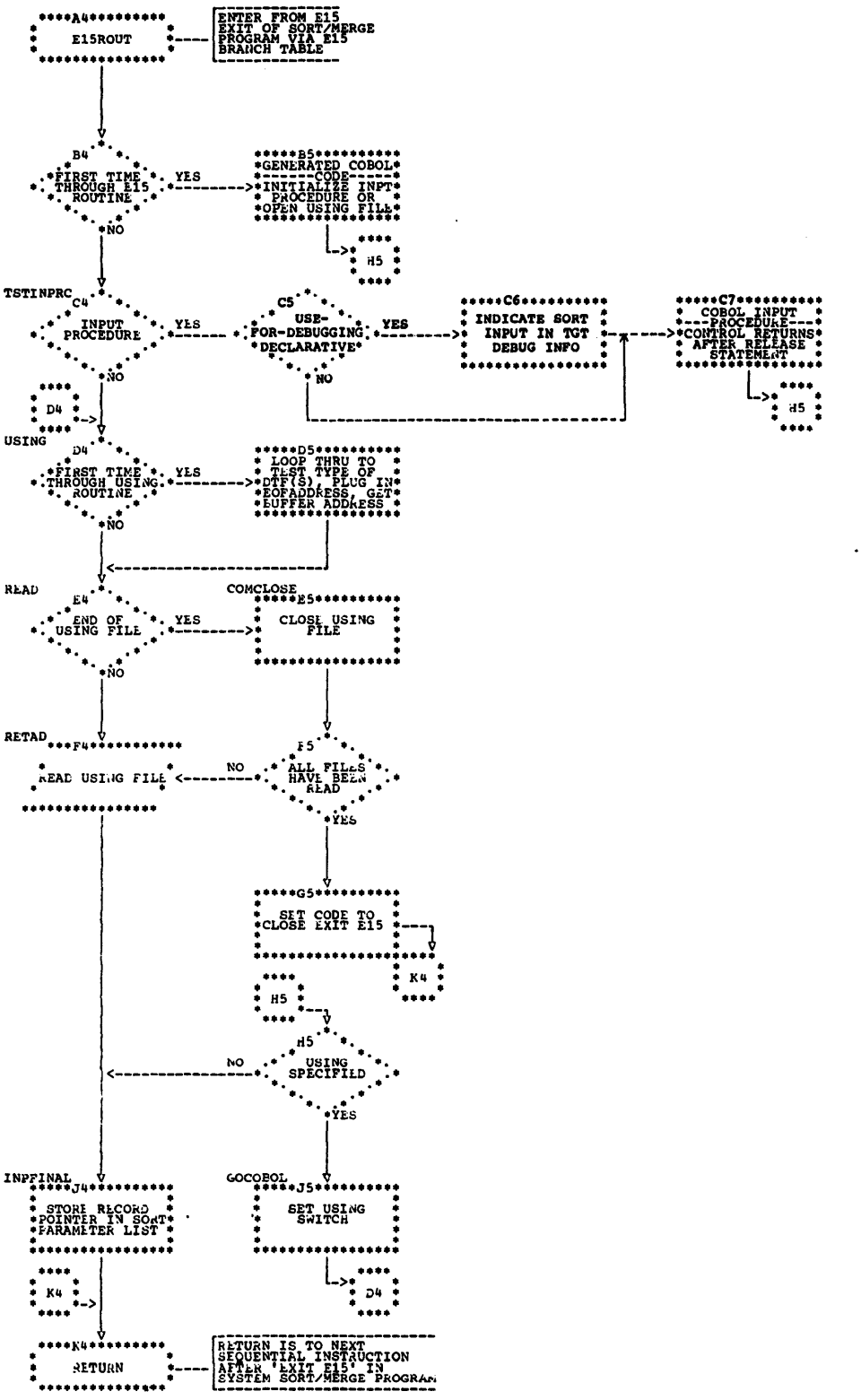


Chart CA. Sort/Merge (ILBDSRT0, ILBDMRG0) (Part 3 of 5): E35ROUT Routine

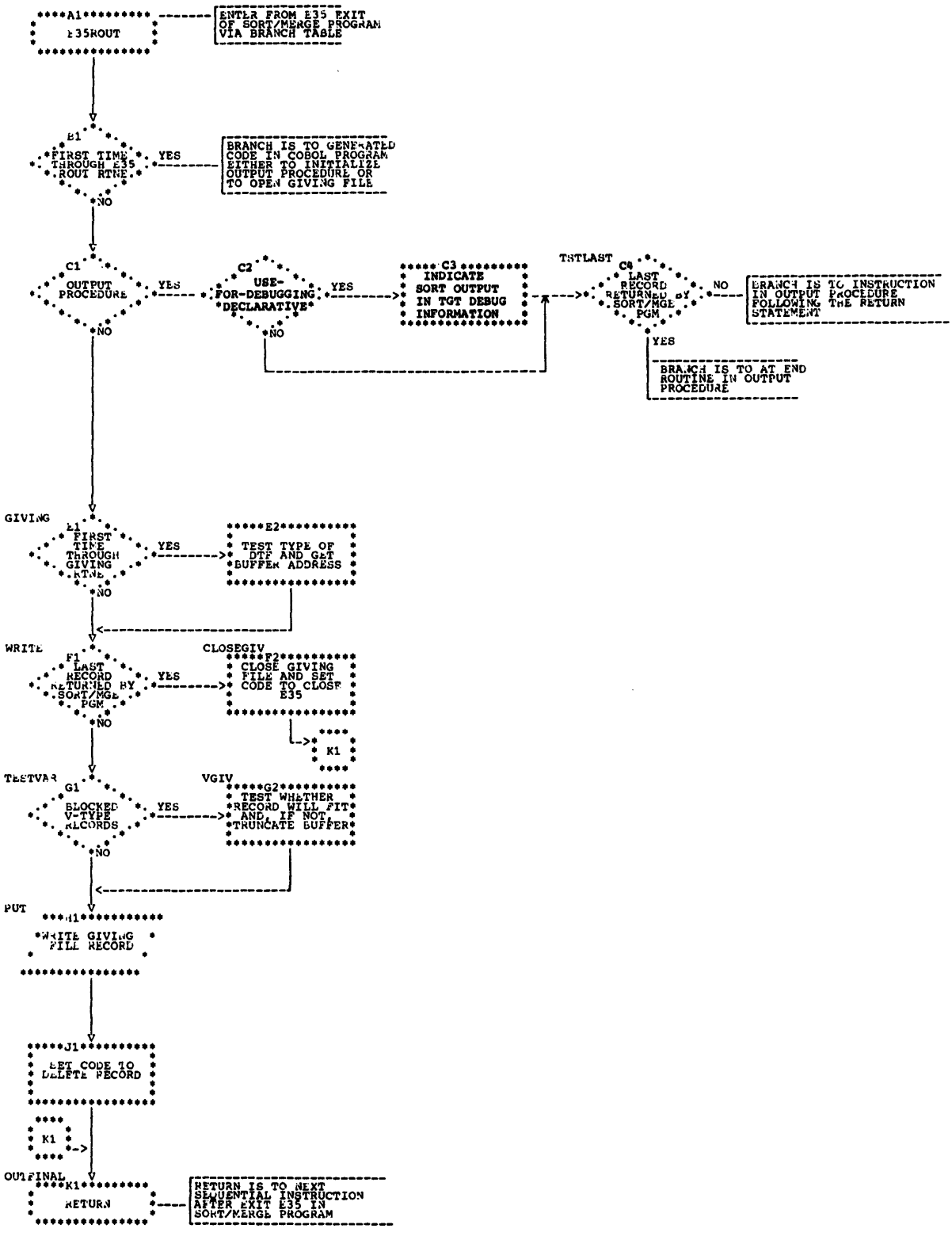


Chart CA. Sort/Merge (ILBDSRT0, ILBDMRG0) (Part 4 of 5): CHKPOINT Routine

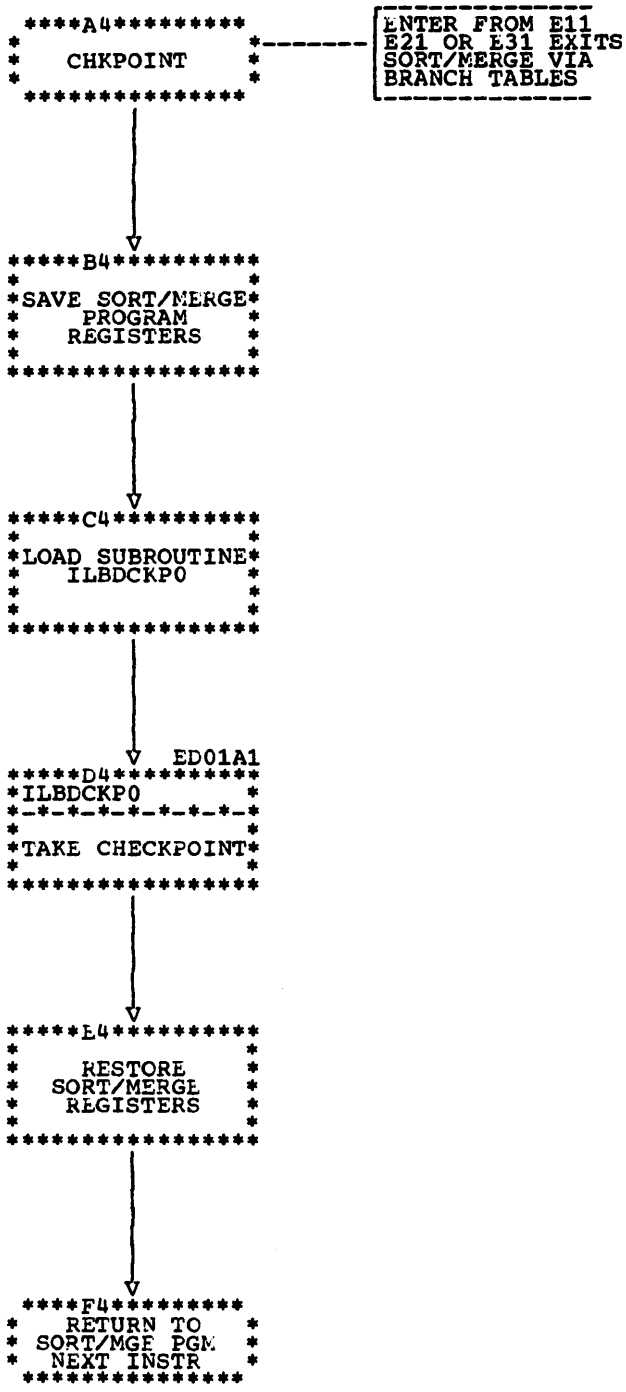


Chart CA. Sort/Merge (ILBDSRT0, ILBDMRG0) (Part 5 of 5): E32 Routine

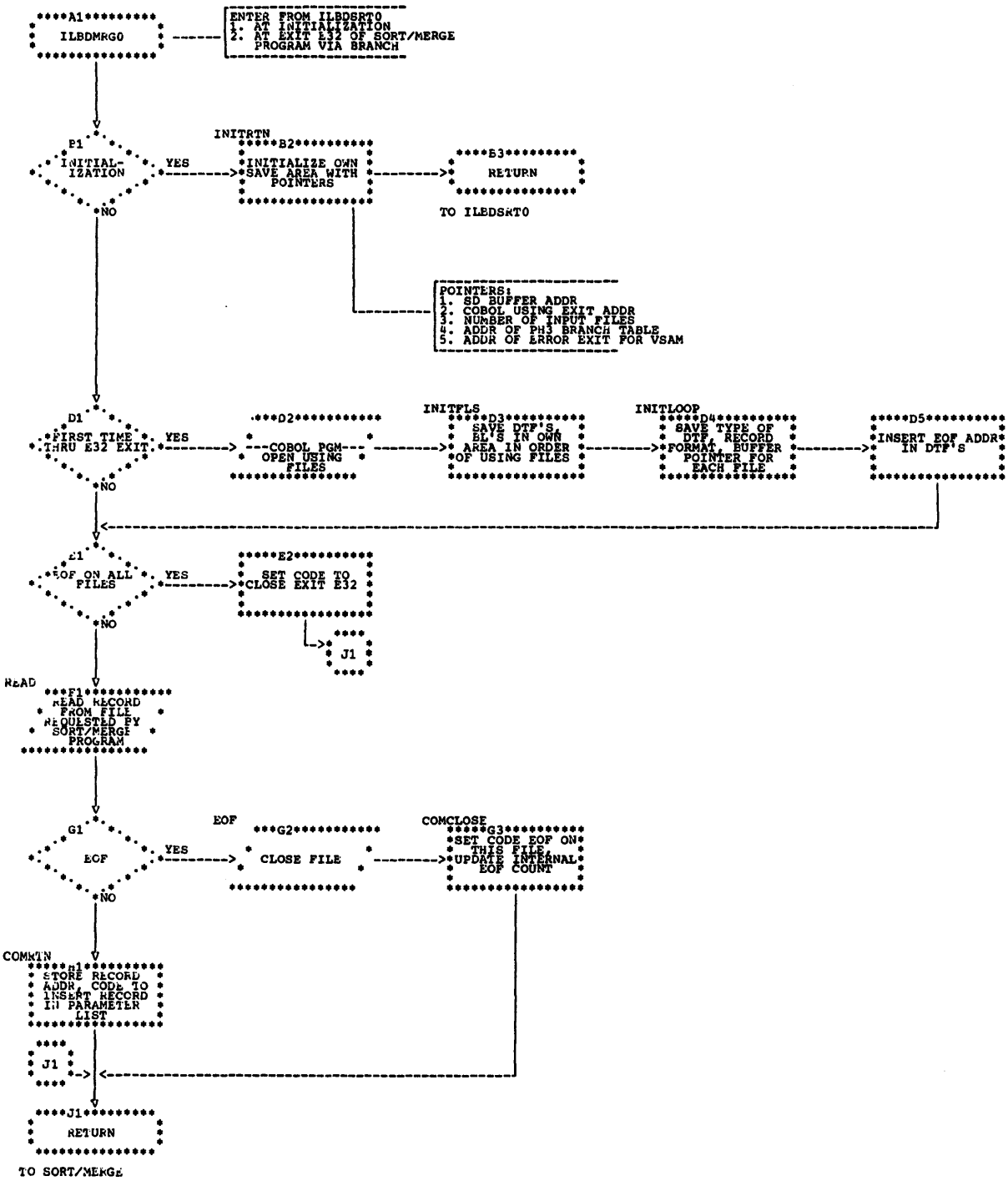


Chart CB. Moving Characters (ILBDMOV0)

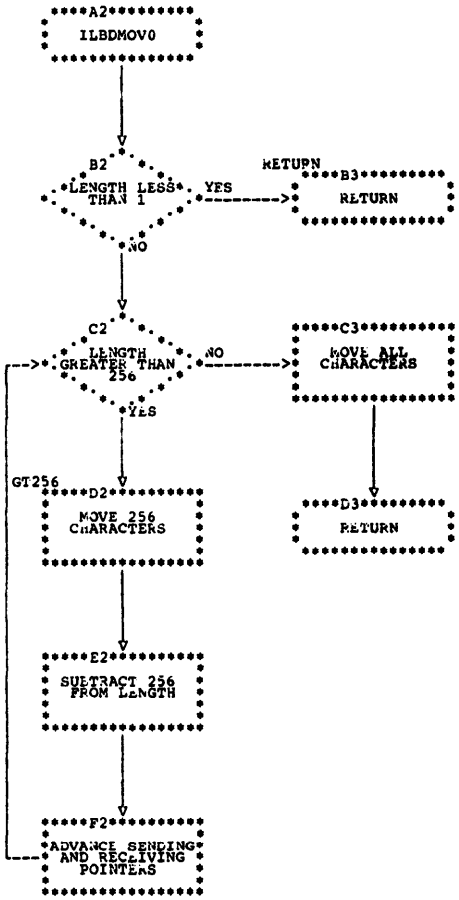


Chart CBA. STRING (ILBDSTG0)(Part 1 of 2)

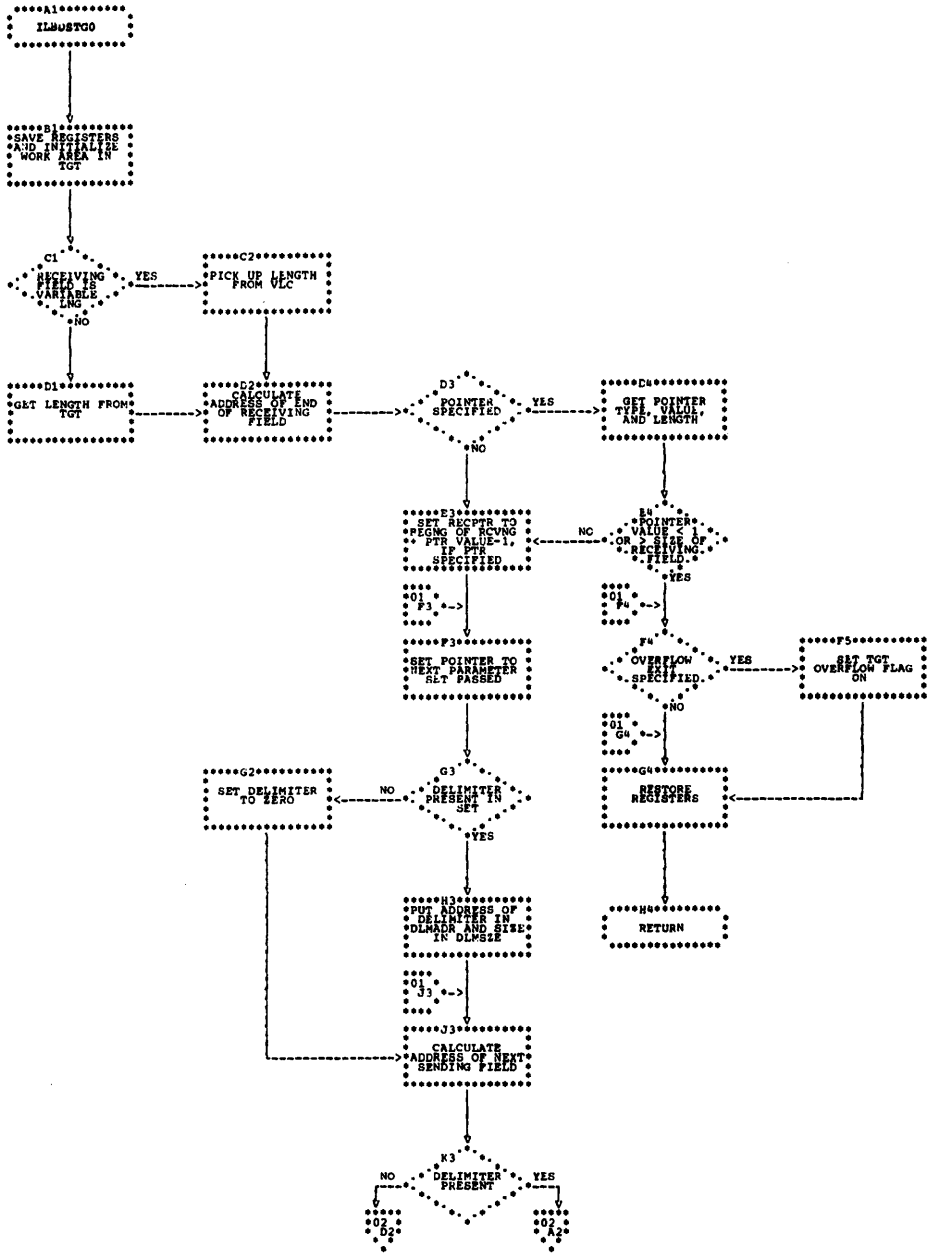


Chart CBA. STRING (ILBDSTG0) (Part 2 of 2)

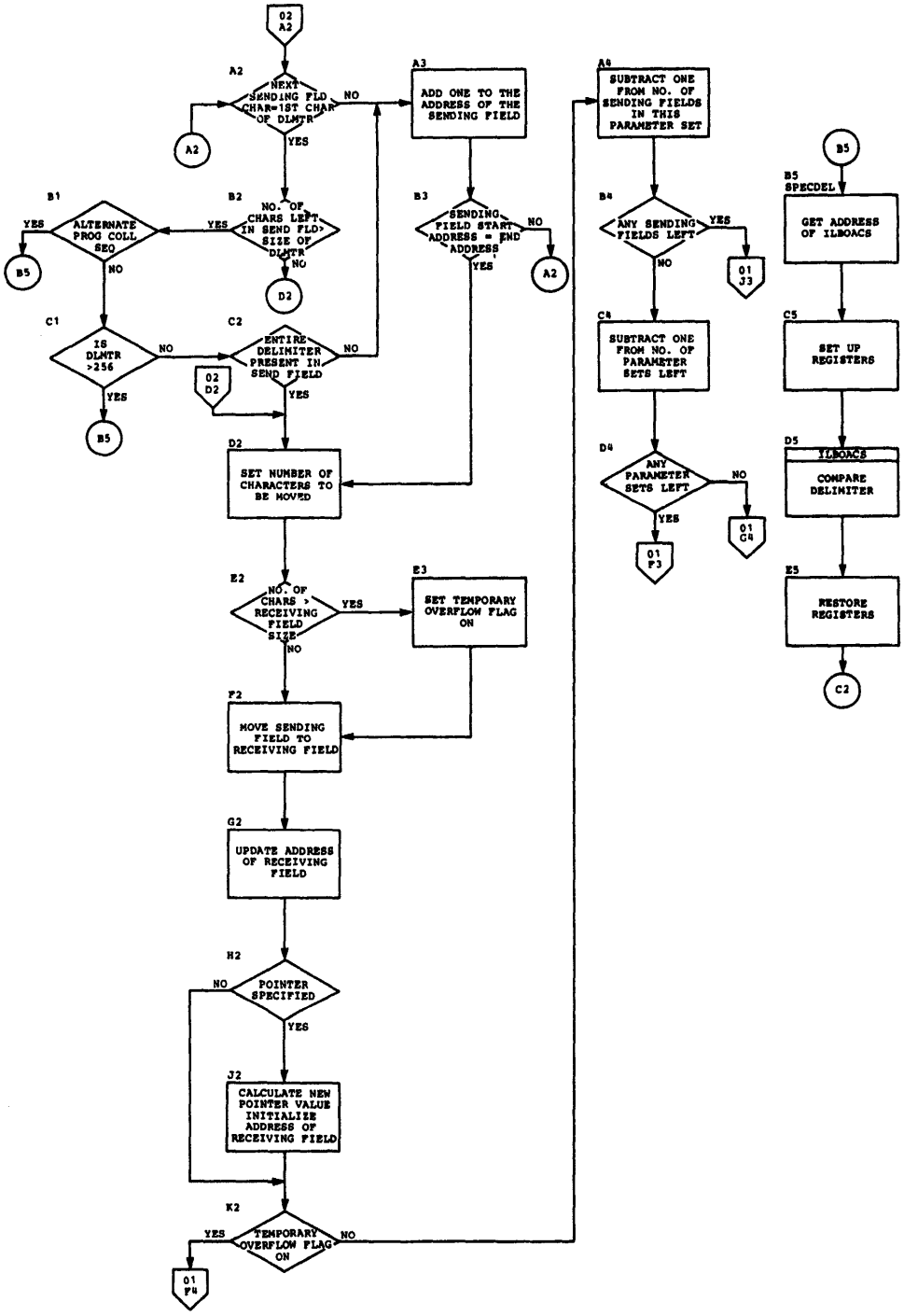


Chart CBB. UNSTRING (ILBDUST0) (Part 1 of 4)

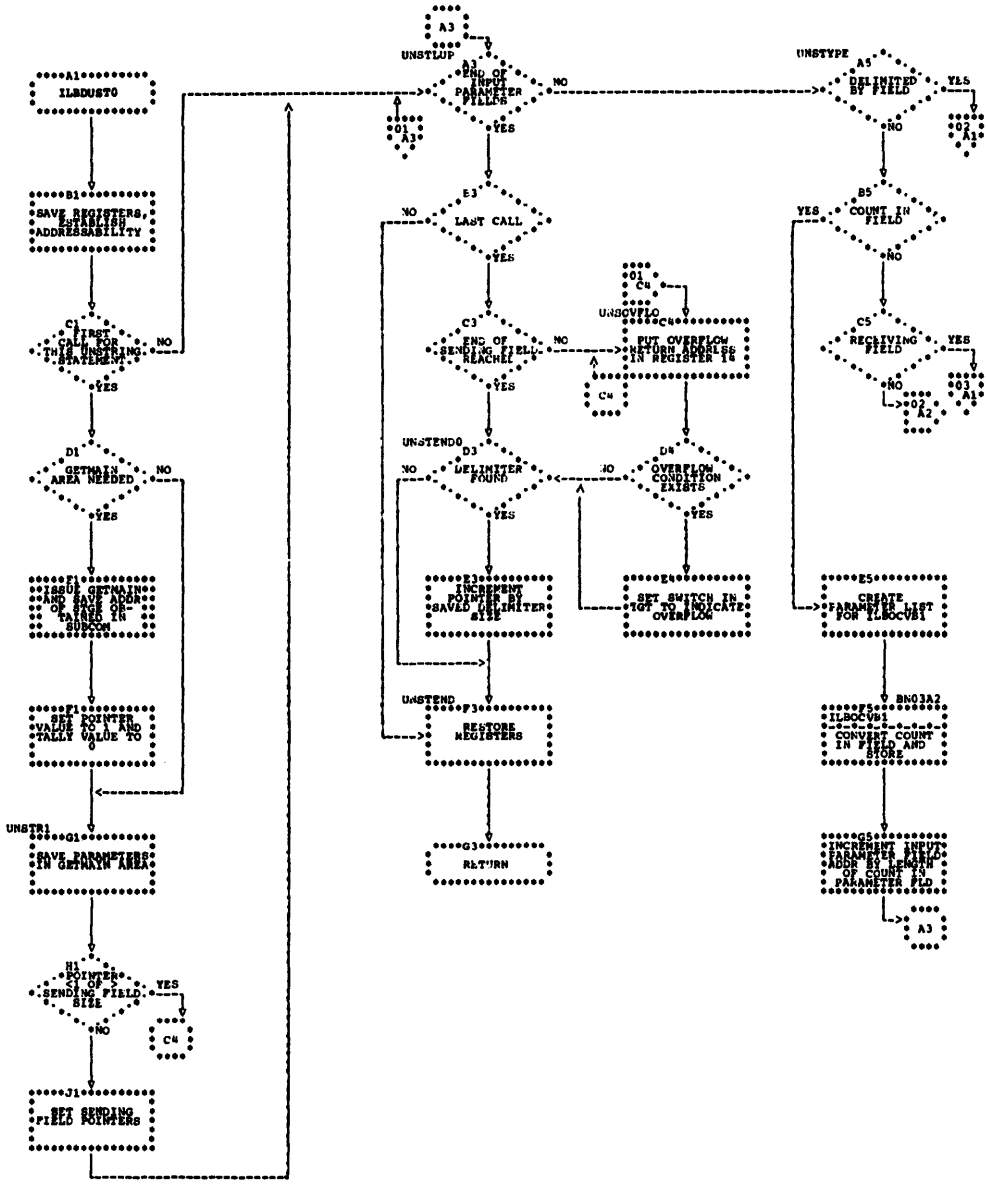


Chart CBB. UNSTRING (ILBDUST0) (Part 2 of 4)

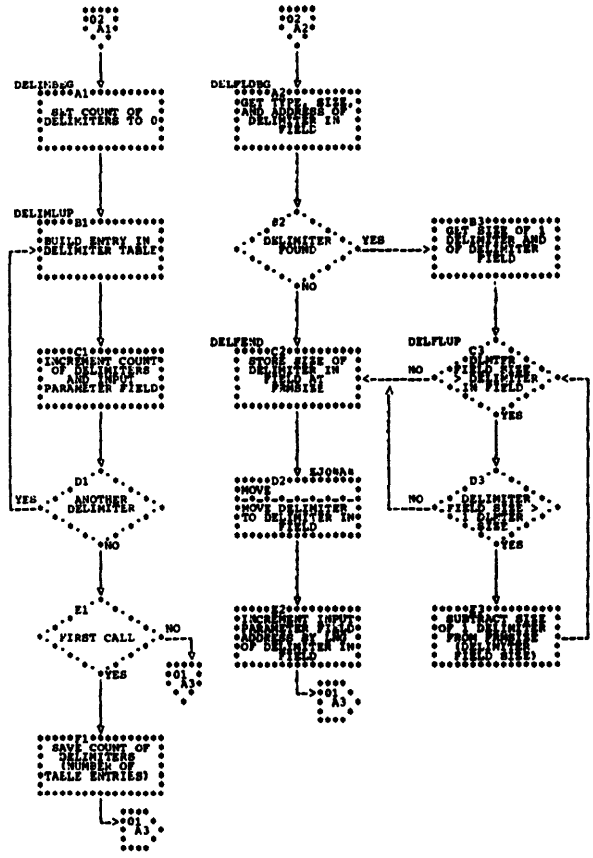


Chart CBB. UNSTRING (ILBDUST0) (Part 3 of 4)

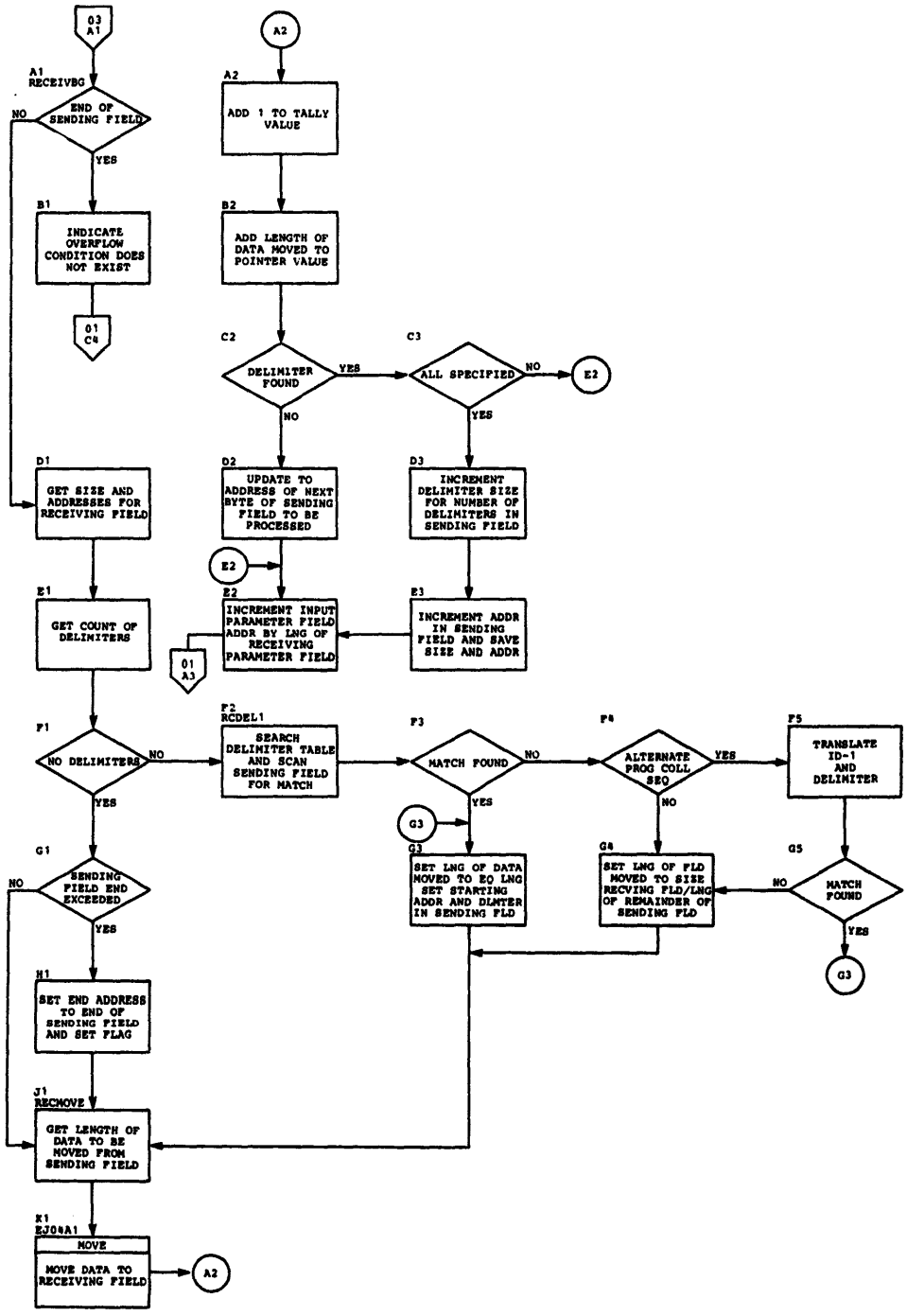


Chart CBB. UNSTRING (ILBDUST0) (Part 4 of 4)

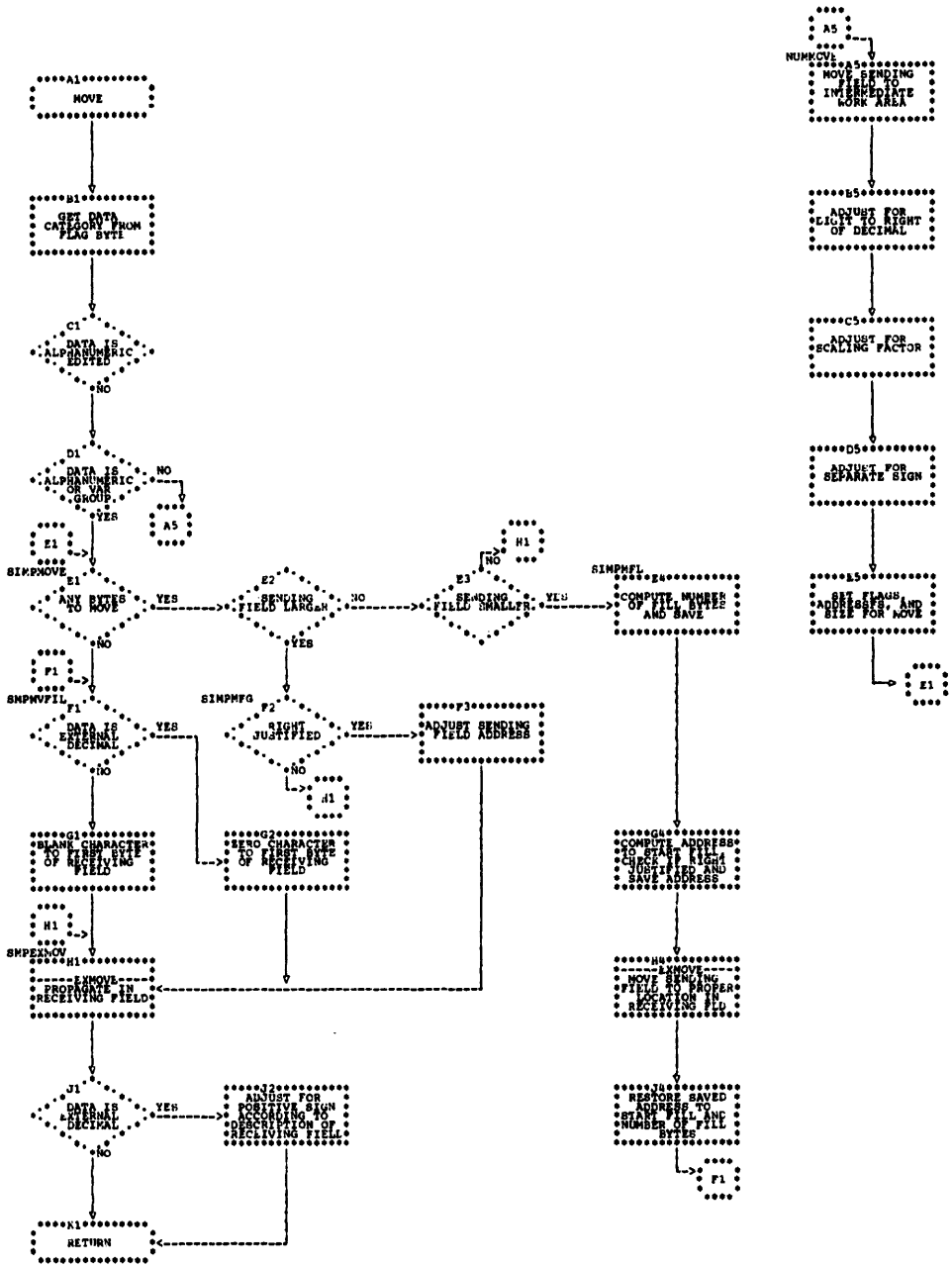


Chart CBC. INSPECT Subroutine (ILBDINS0) (Part 1 of 4)

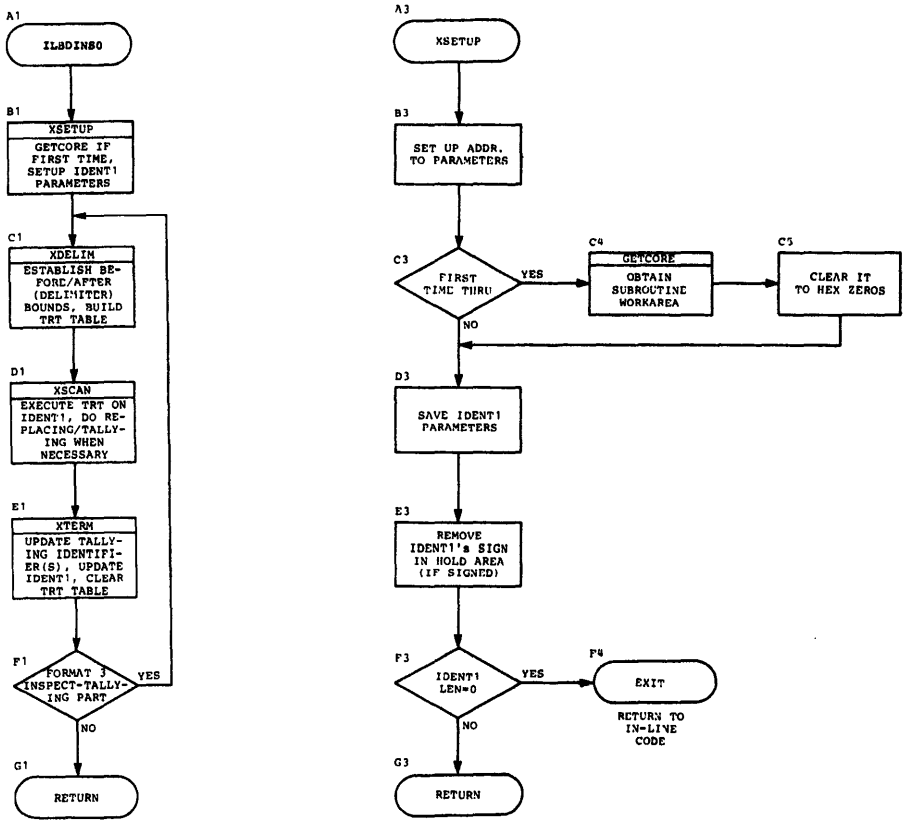


Chart CBC. INSPECT Subroutine (ILBDINS0) (Part 2 of 4)

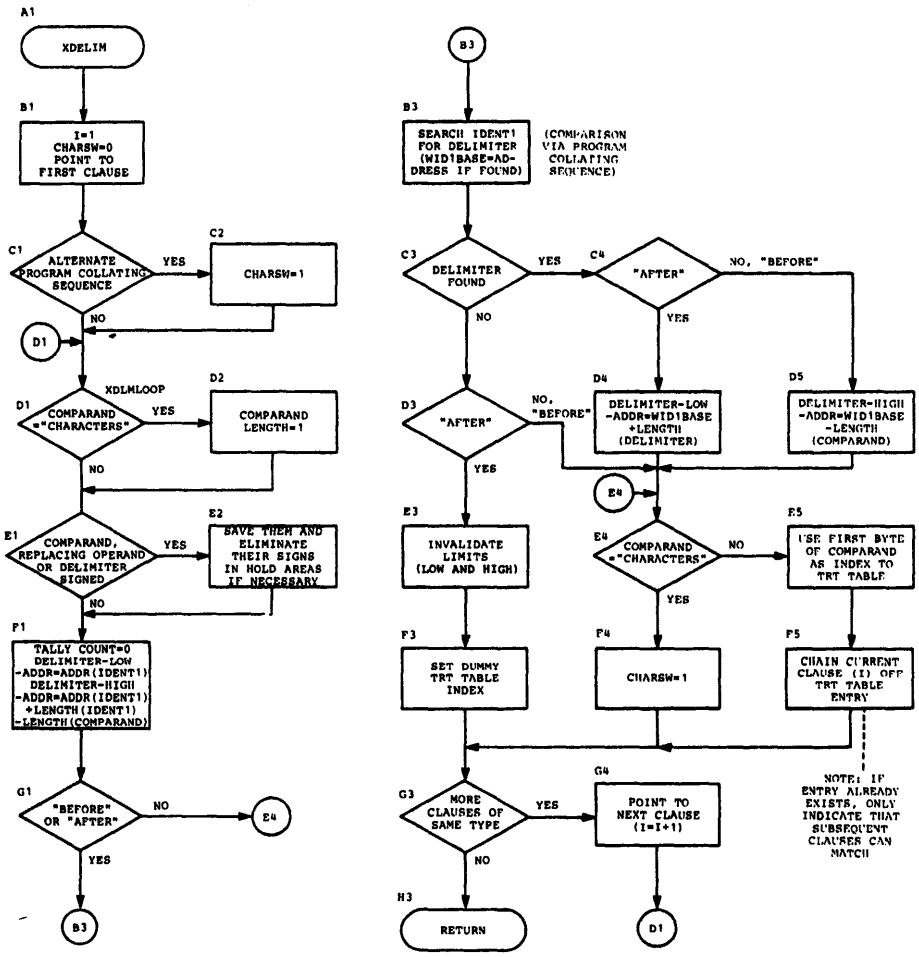


Chart CBC. INSPECT Subroutine (ILBDINS0) (Part 3 of 4)

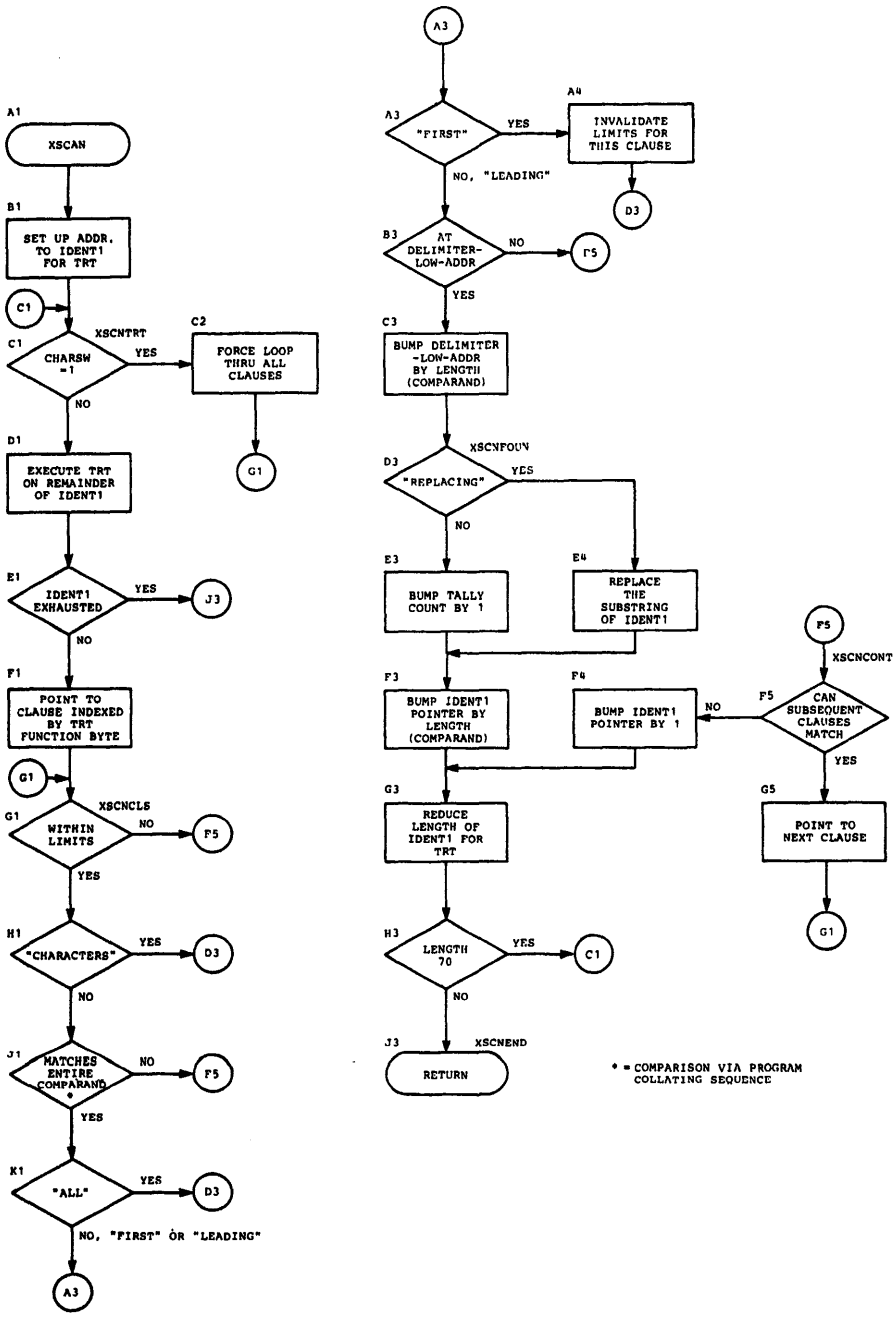


Chart CBC. INSPECT Subroutine (ILBDINS0) (Part 4 of 4)

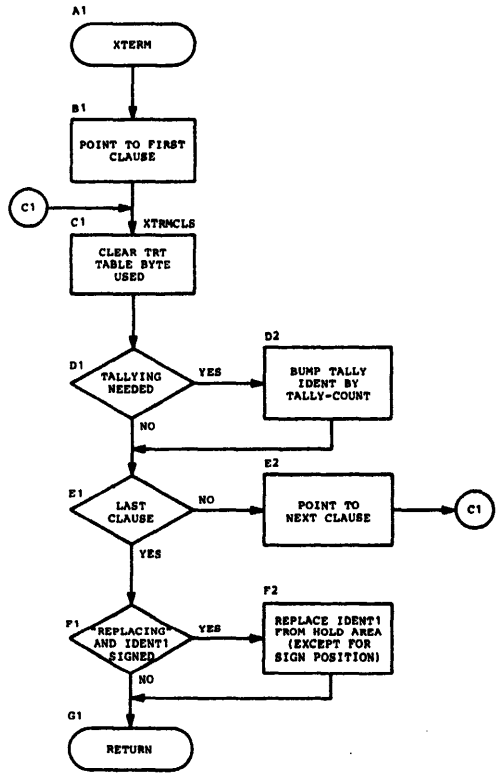


Chart CC. Segmentation (ILBDSEM0)

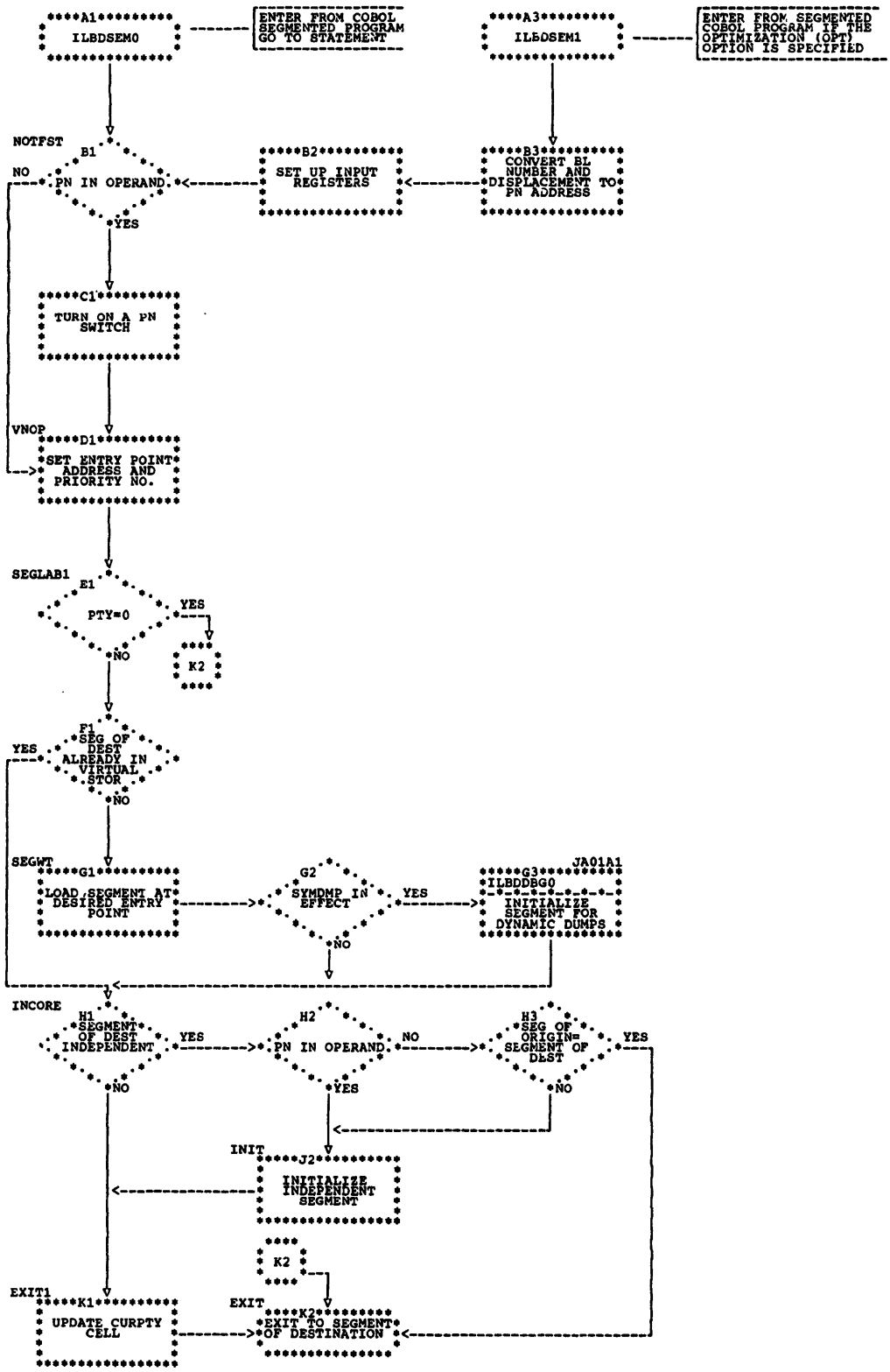


Chart CCA. GO TO DEPENDING ON (ILBDGDO0, ILBDGDO1, ILBDGDO2)

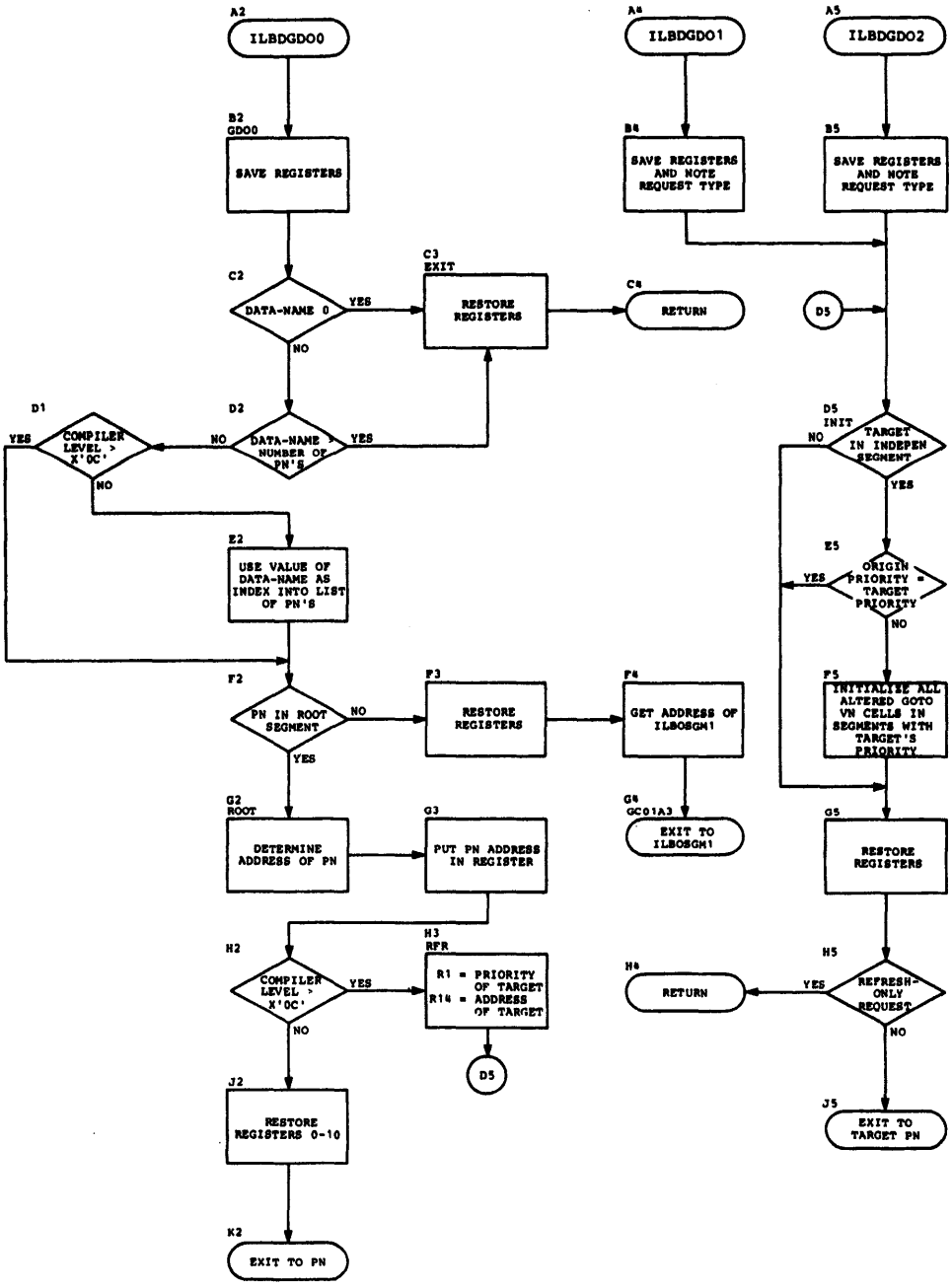


Chart CCB. DATE, DAY, and TIME (ILBDDTE0, ILBDDTE1, ILBDDTE2)

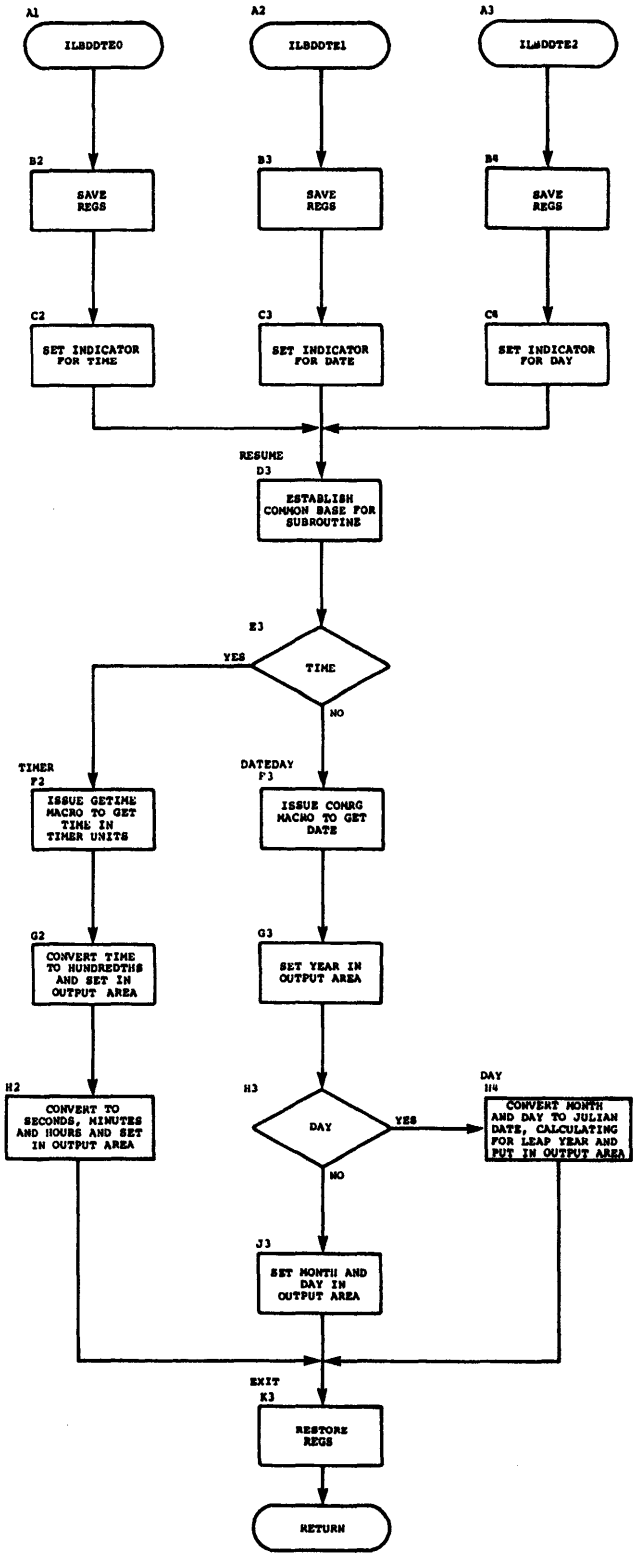


Chart CCC. GETCORE/FREECORE Subroutines (ILBDCMM0, ILBDCMM1) (Part 1 of 2)

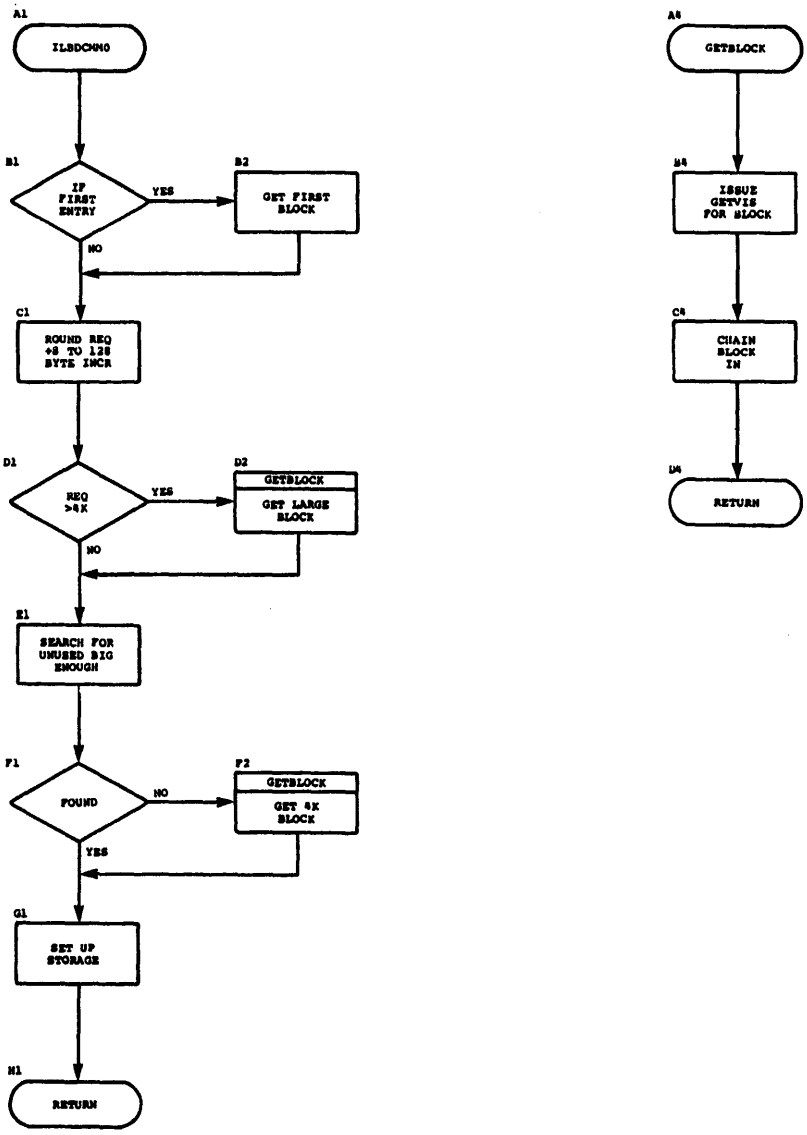


Chart CCC. GETCORE/FREECORE Subroutines (ILBDCMM0, ILBDCMM1) (Part 2 of 2)

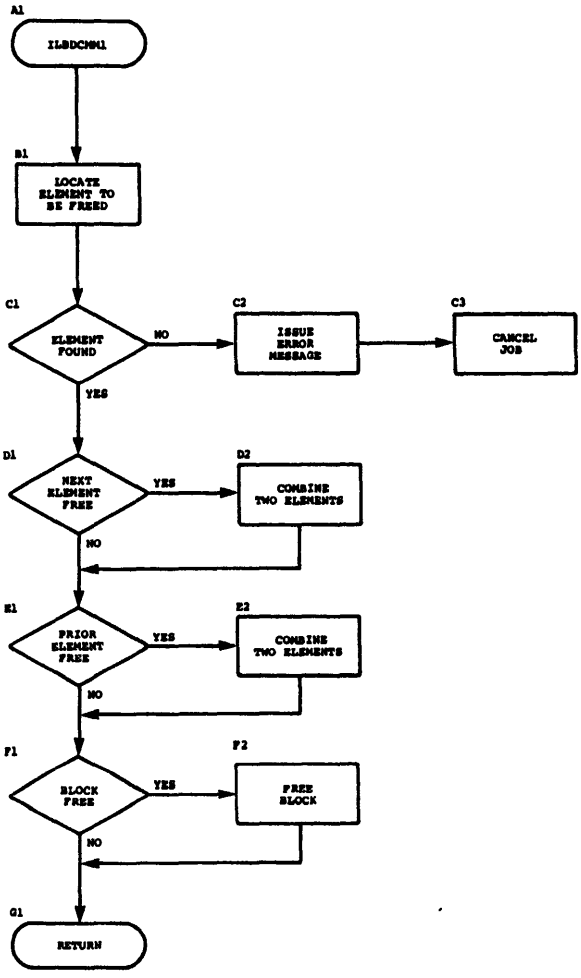


Chart CCD. Comparison with Alternate Collating Sequence (ILBDACS)

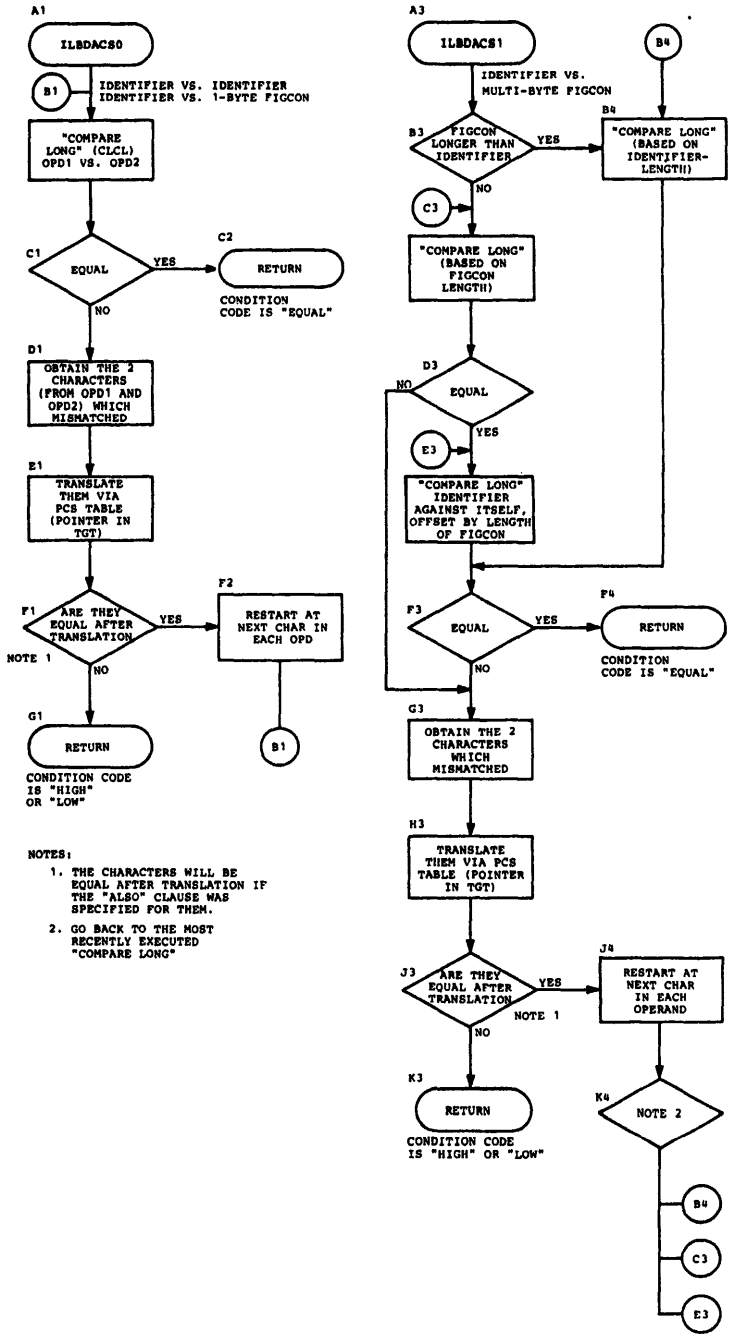


Chart EA. Display (ILBDDSP0) (Part 1 of 2)

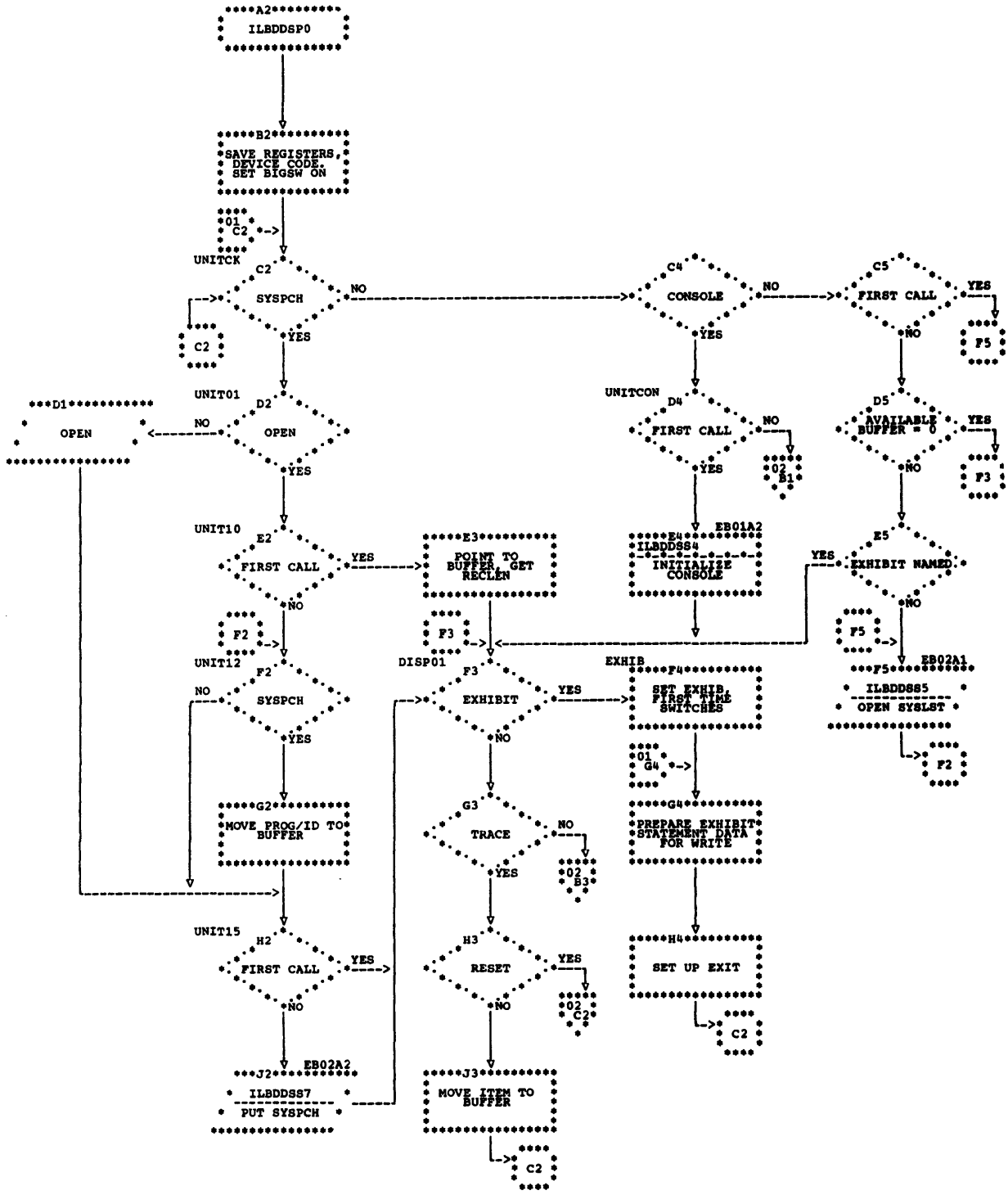


Chart EA. Display (ILBDDSP0) (Part 2 of 2)

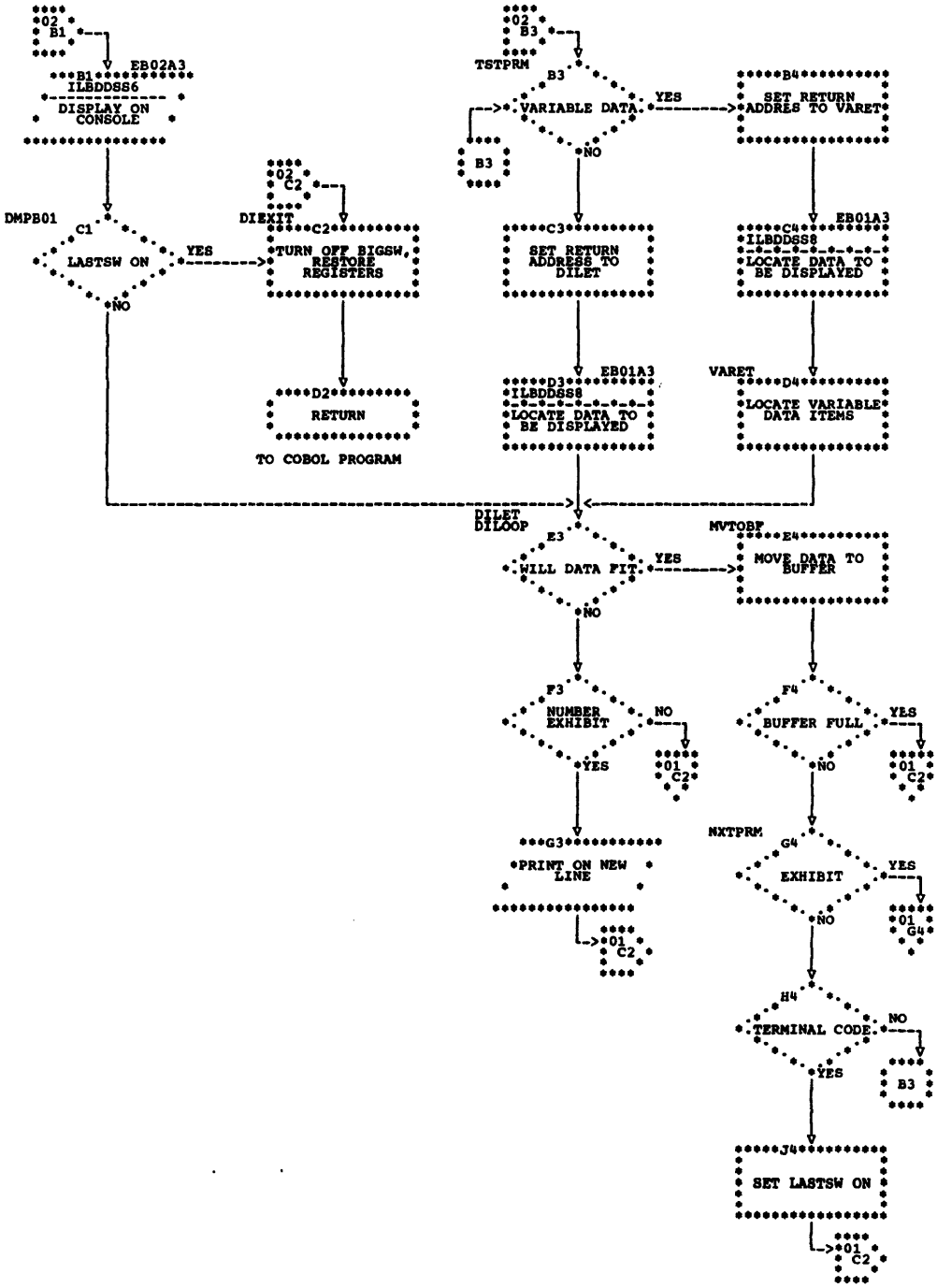


Chart EB. Optimizer DISPLAY (ILBDDSS0) (Part 1 of 2)

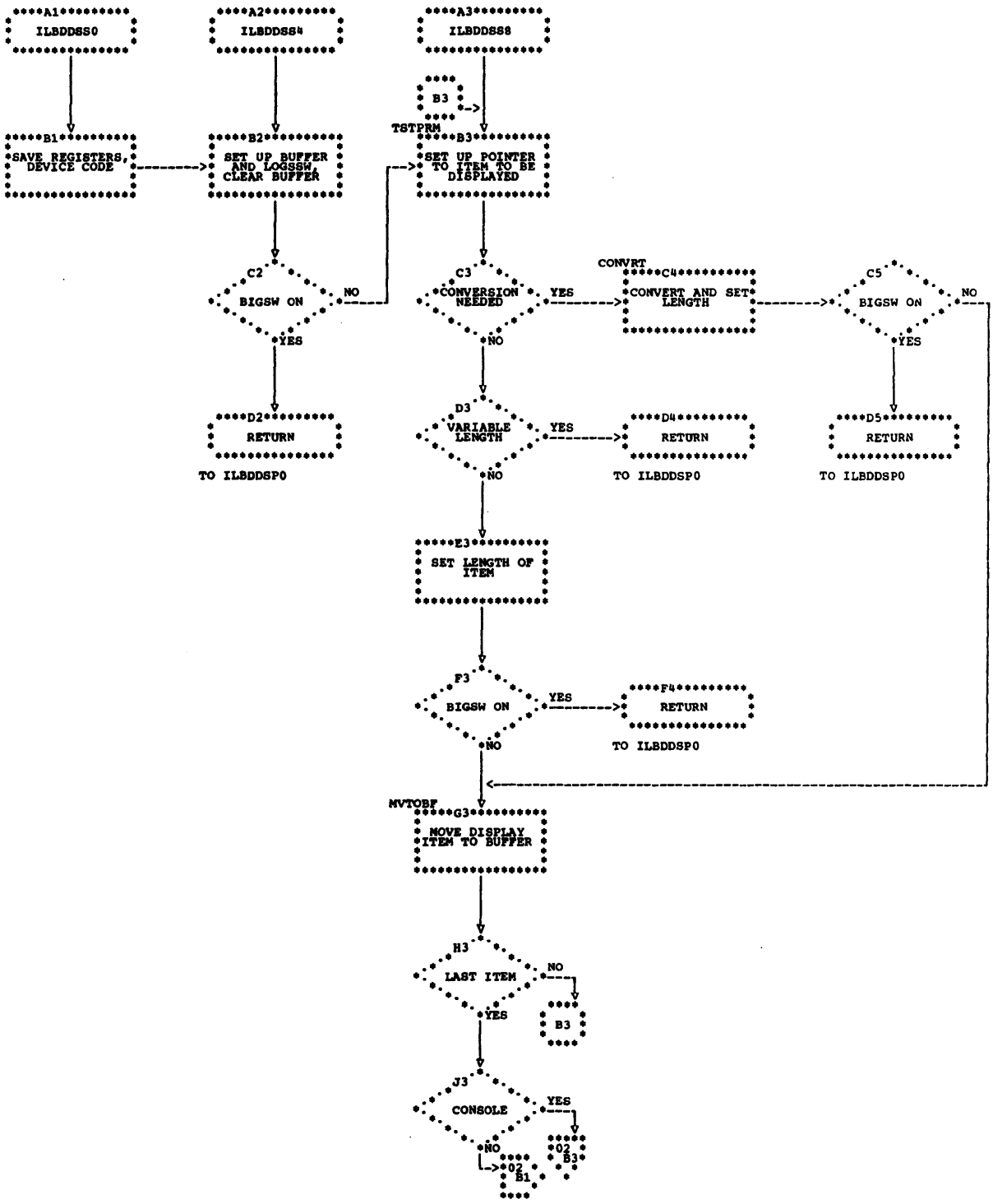


Chart EB. Optimizer DISPLAY (ILBDDSS0) (Part 2 of 2)

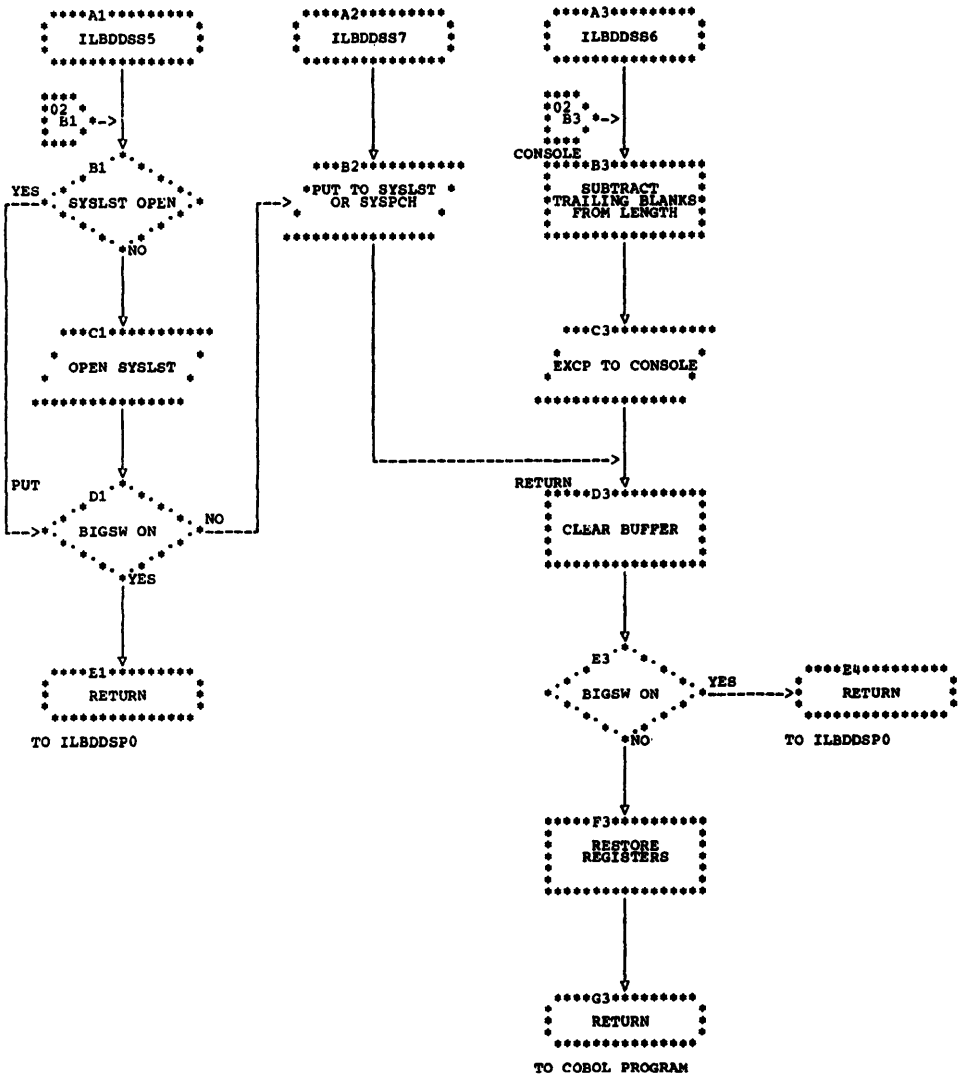


Chart EC. Accept (ILBDACP0)

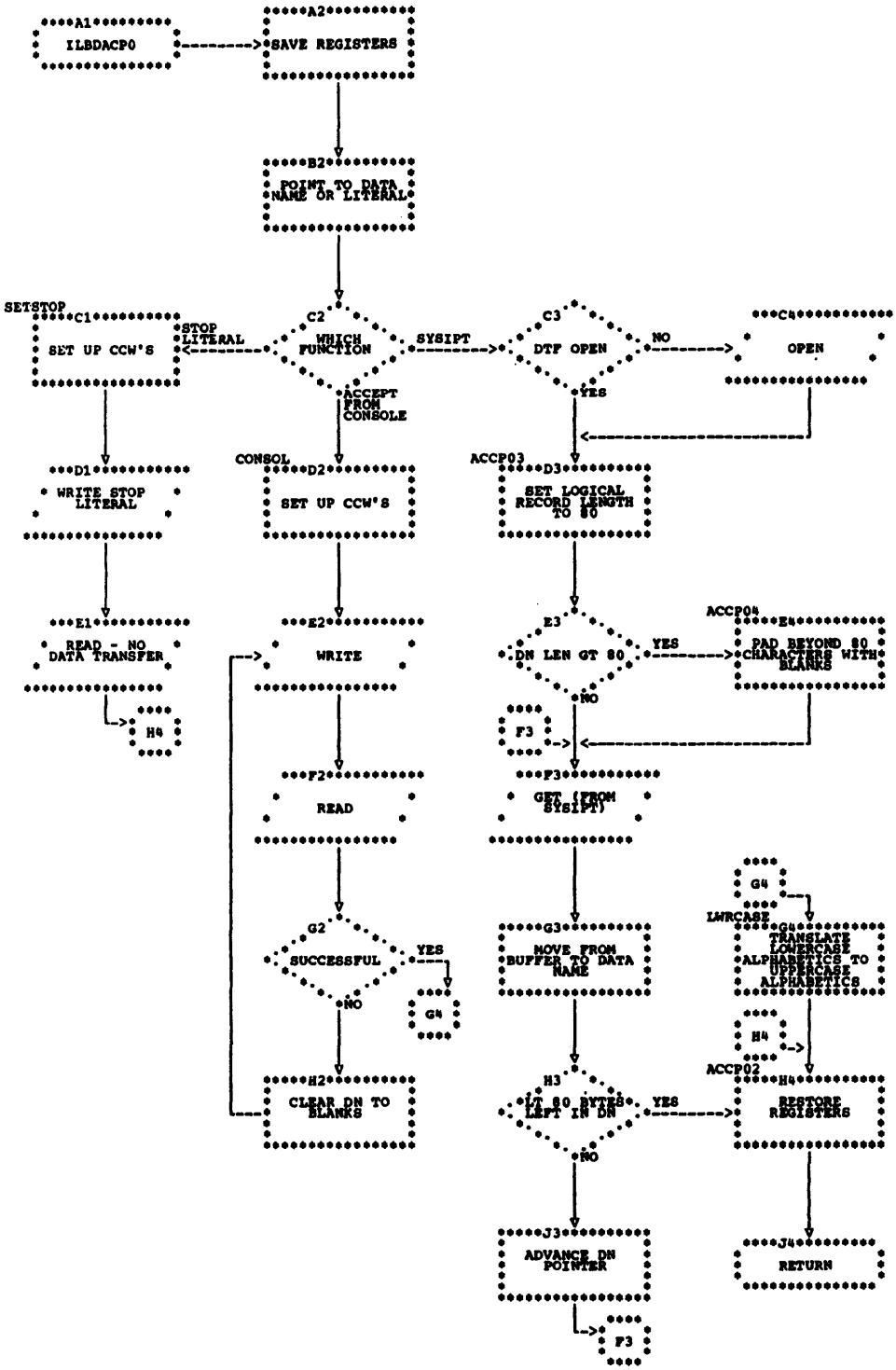


Chart ED. Checkpoint (ILBDCKP0)

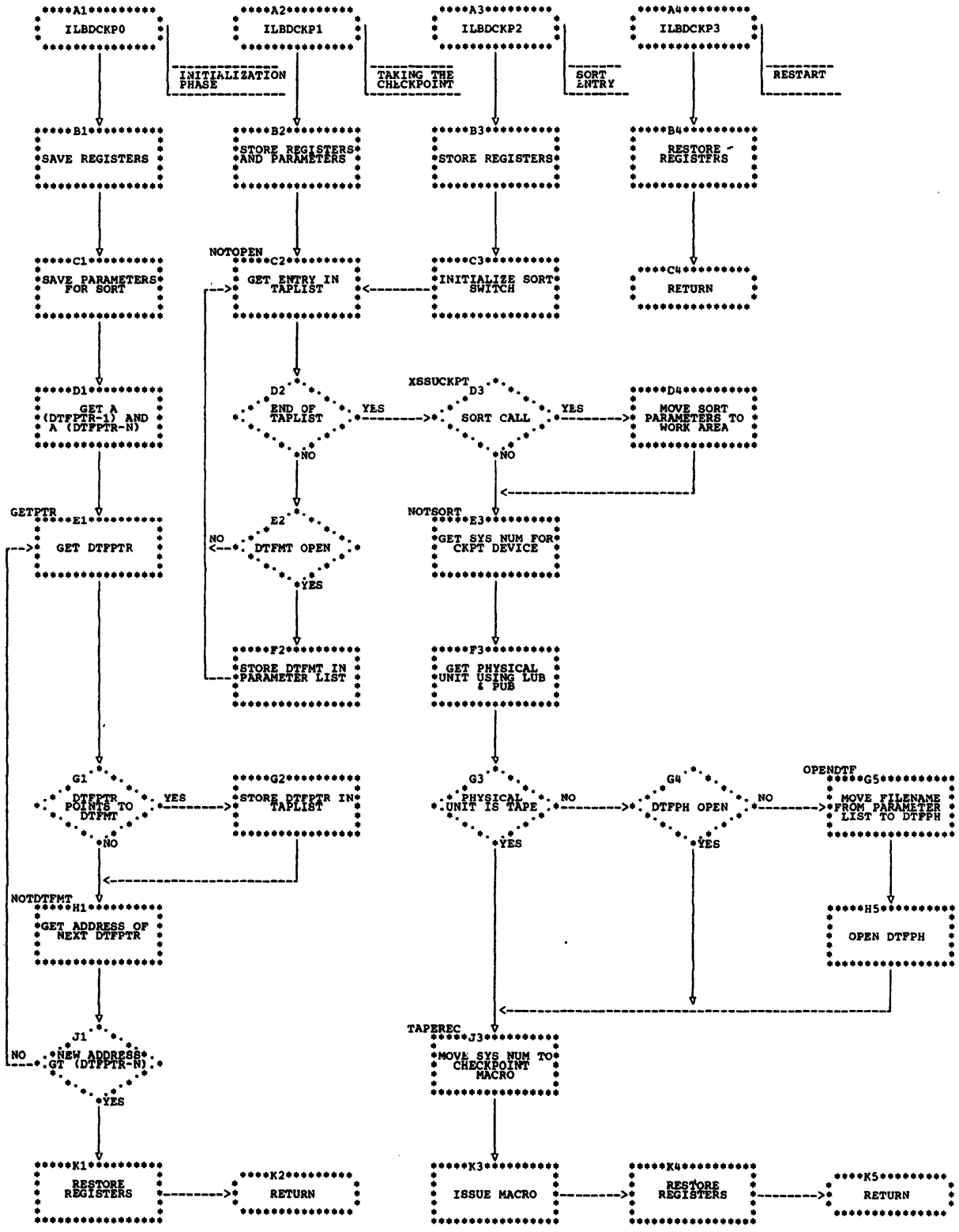


Chart EE. Open ACCEPT File (ILBDASY0)

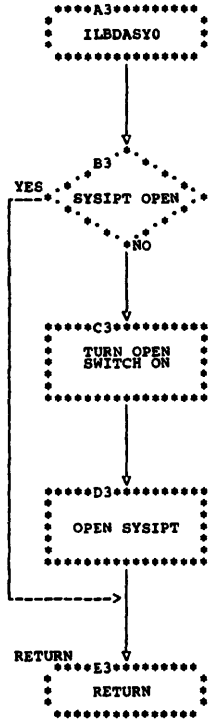


Chart EF. Open DISPLAY File (ILBDOSY0)

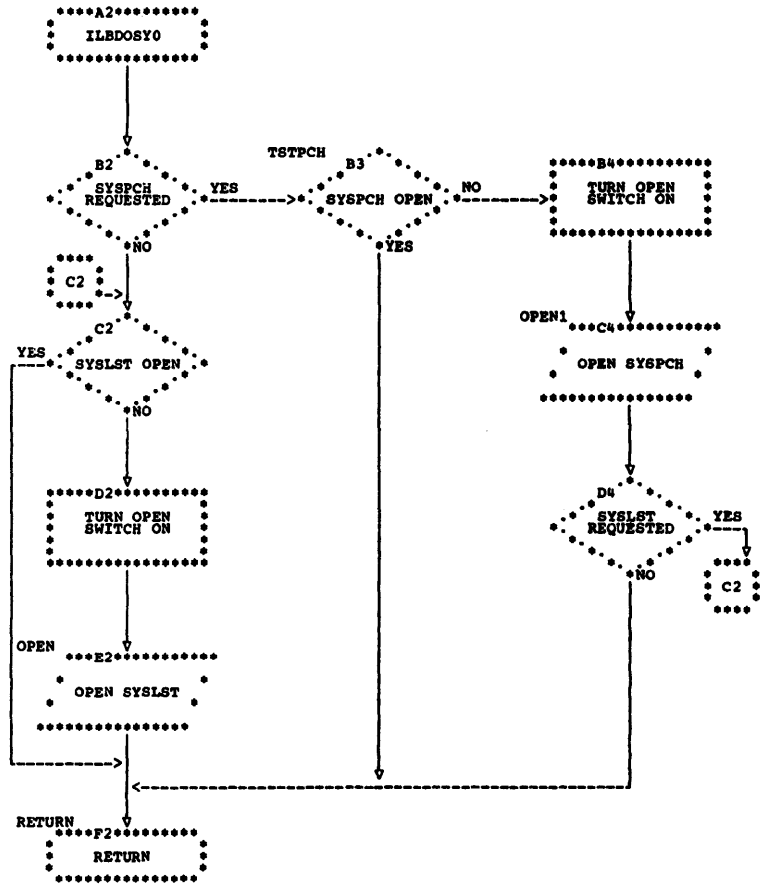


Chart EG. Close With Lock (ILBDCLK0)

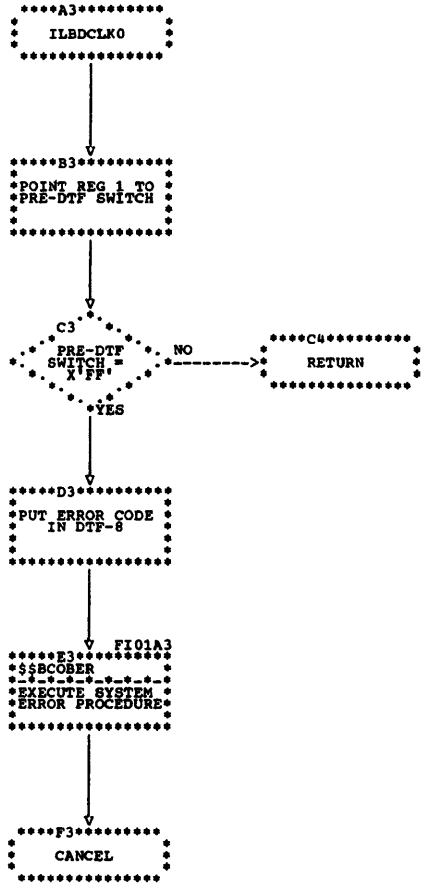


Chart EH. User Standard Labels (ILBDUSL0)

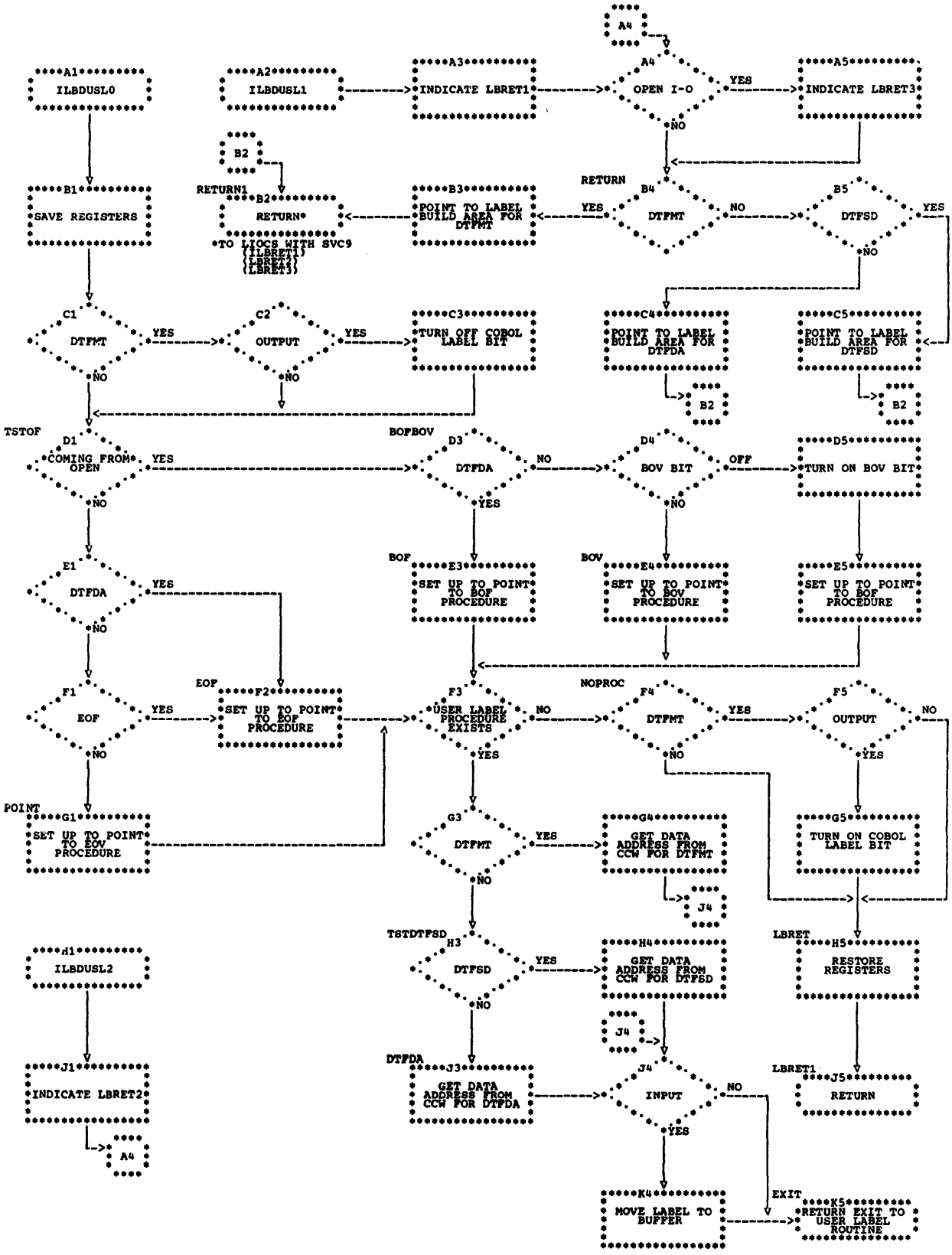


Chart EI. Nonstandard Labels (ILBDNSL0)

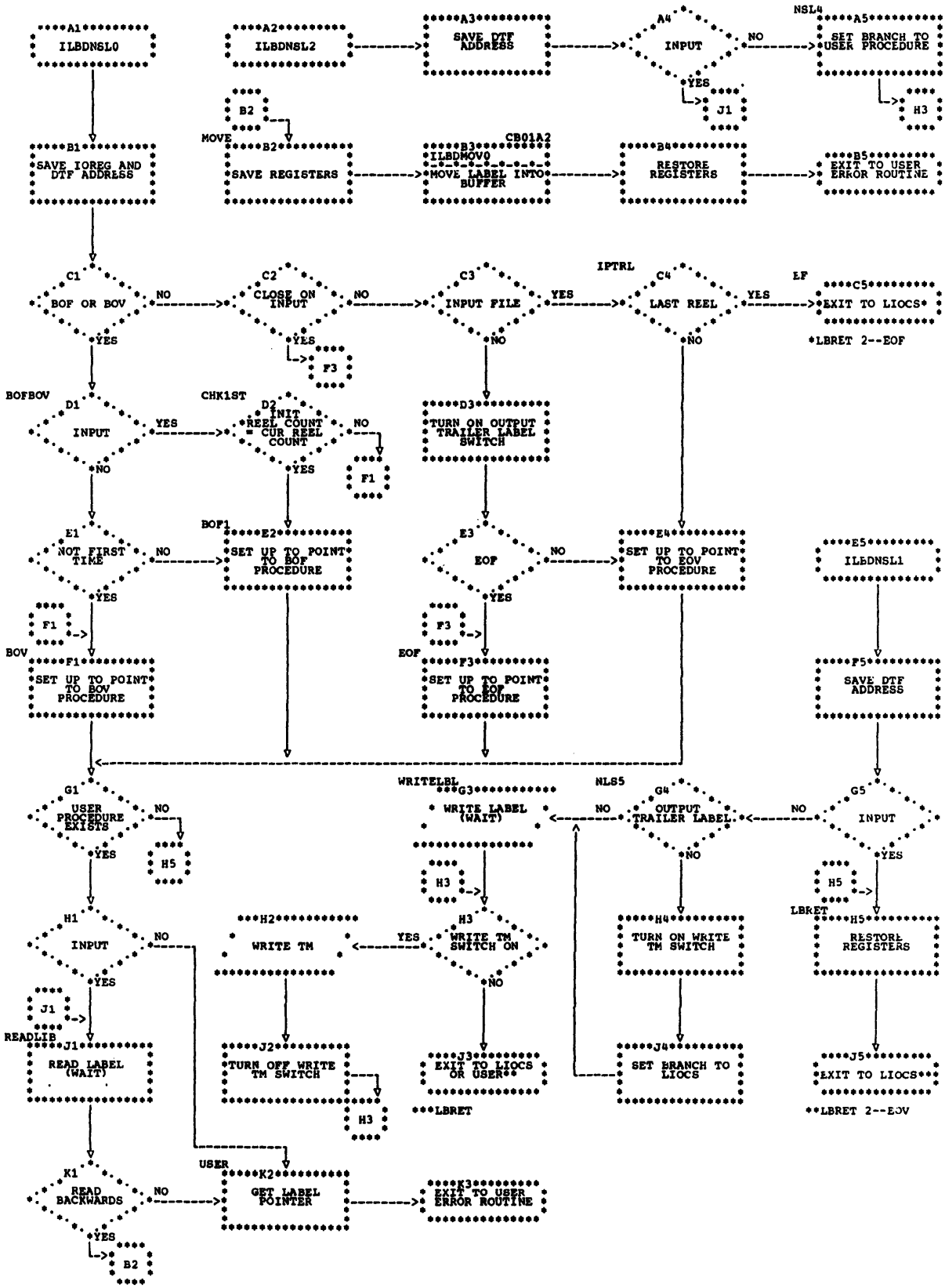


Chart EJ. Error Messages ($$BCOBER)

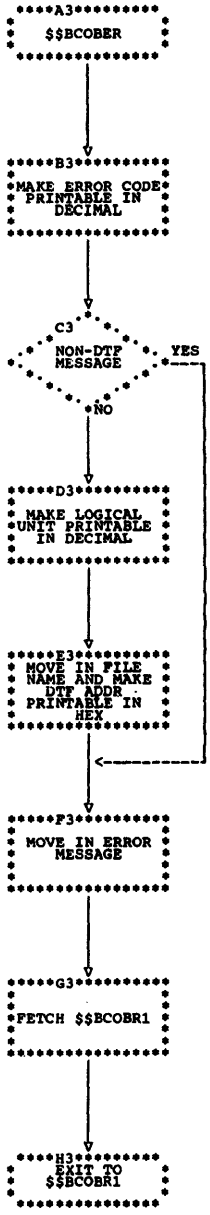


Chart EK. Error Messages Print ($$BCOBR1)

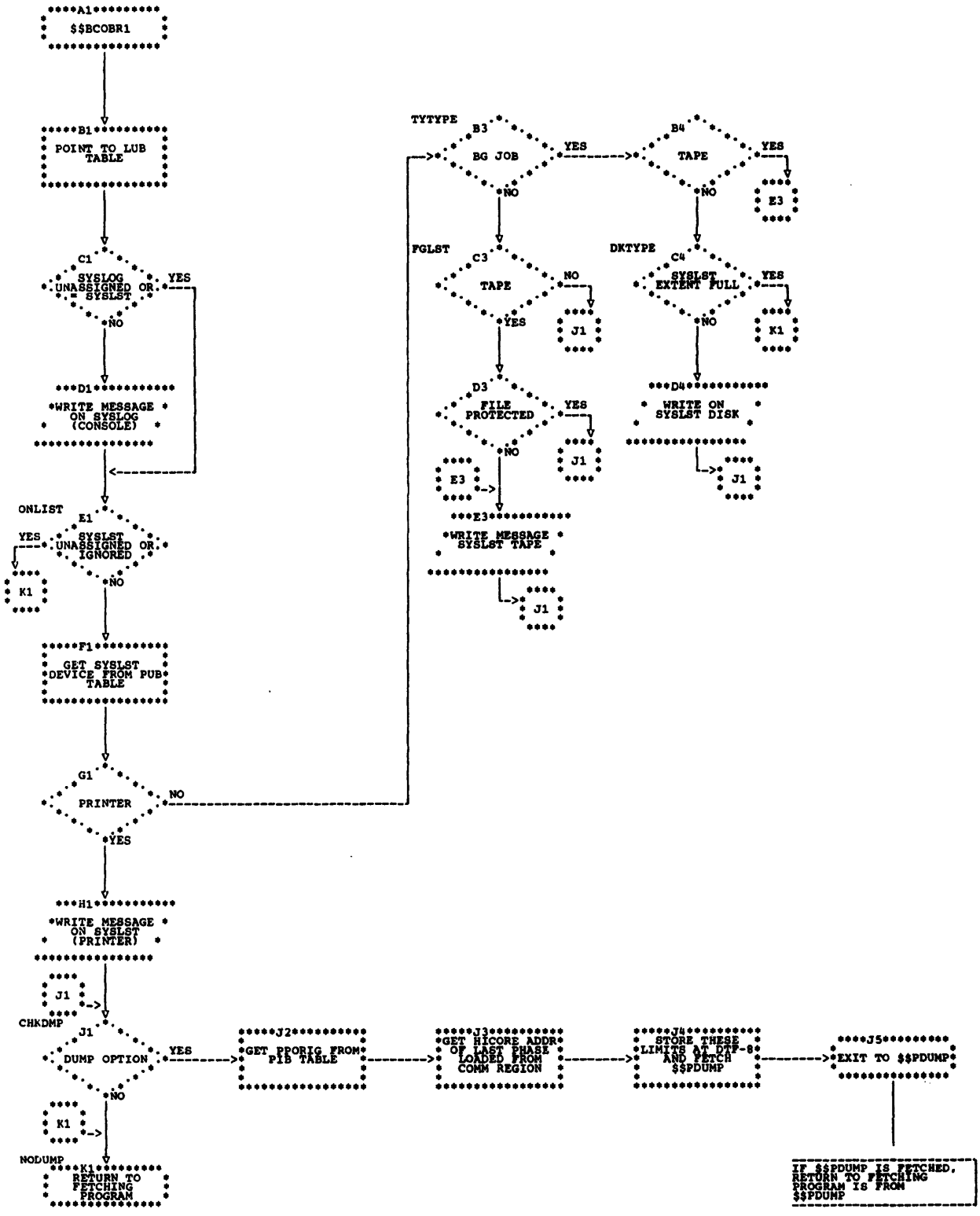


Chart EL. SYMDMP Error Messages ($$BCOBEM)

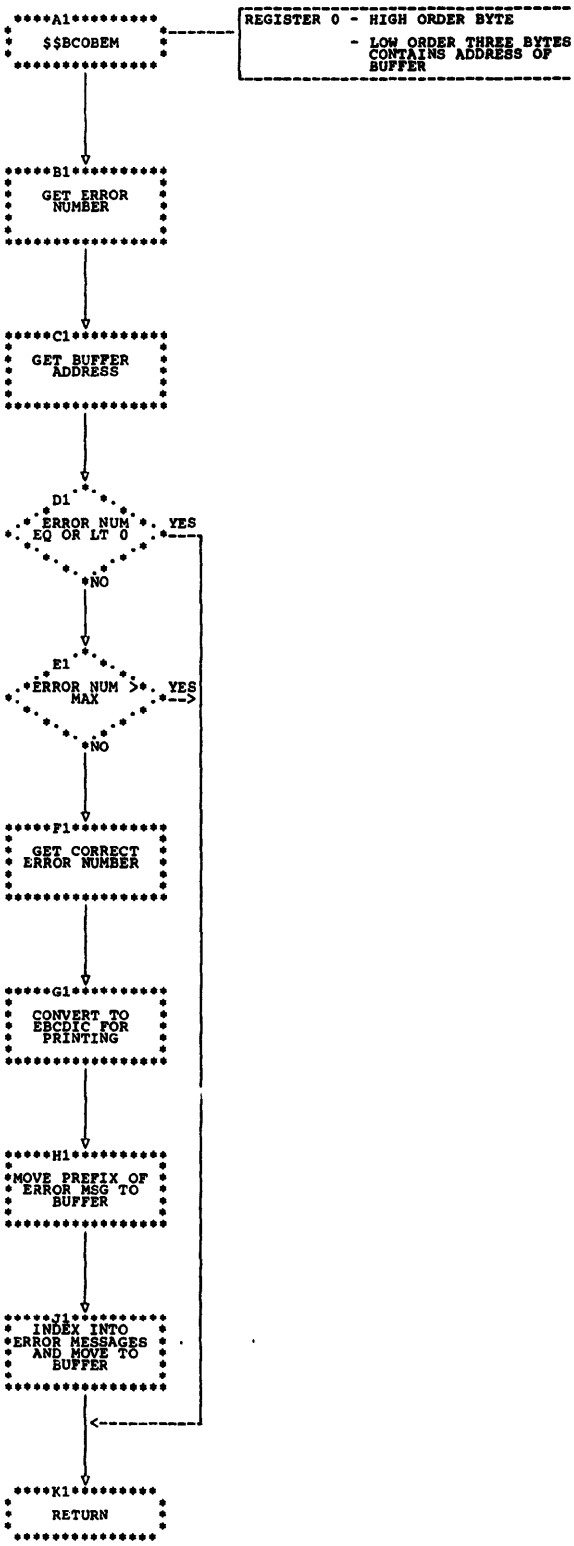


Chart EM. Optical Character Reader (OCR) Interface (ILBDOCR0)

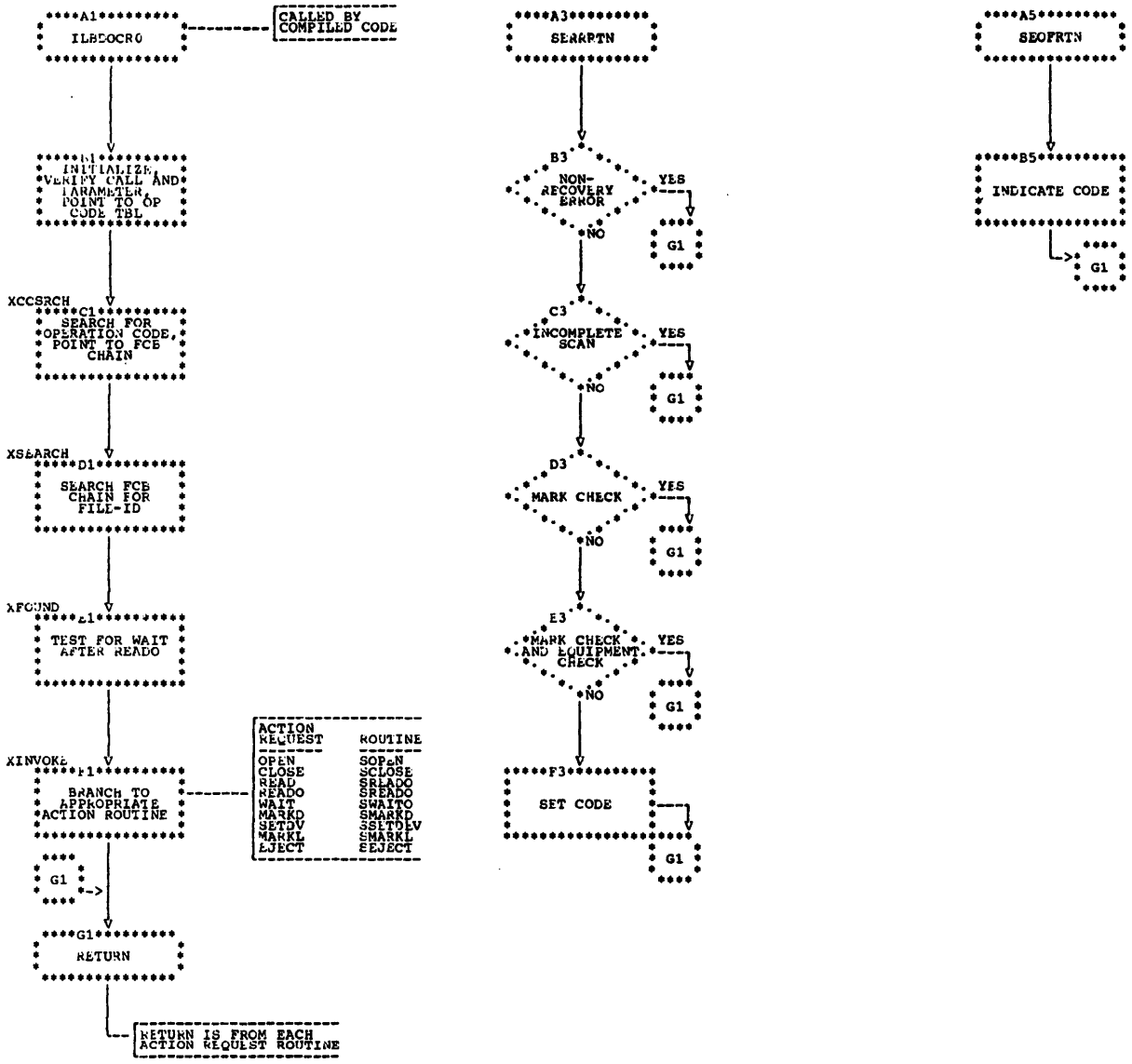


Chart F. SAM I/O (ILBDSIO0) (Part 1 of 10)

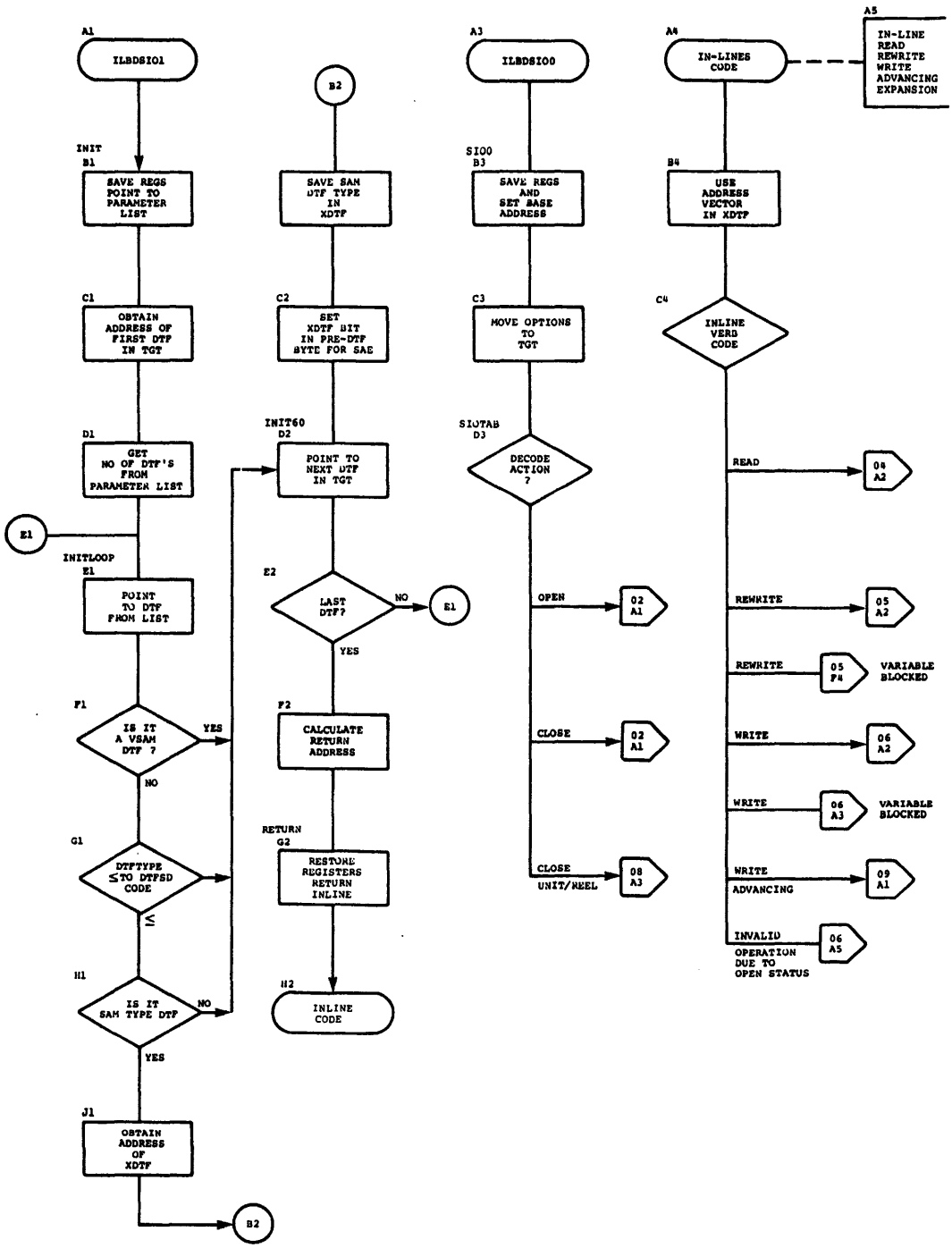


Chart F. SAM I/O (ILBDSIO0) (Part 2 of 10)

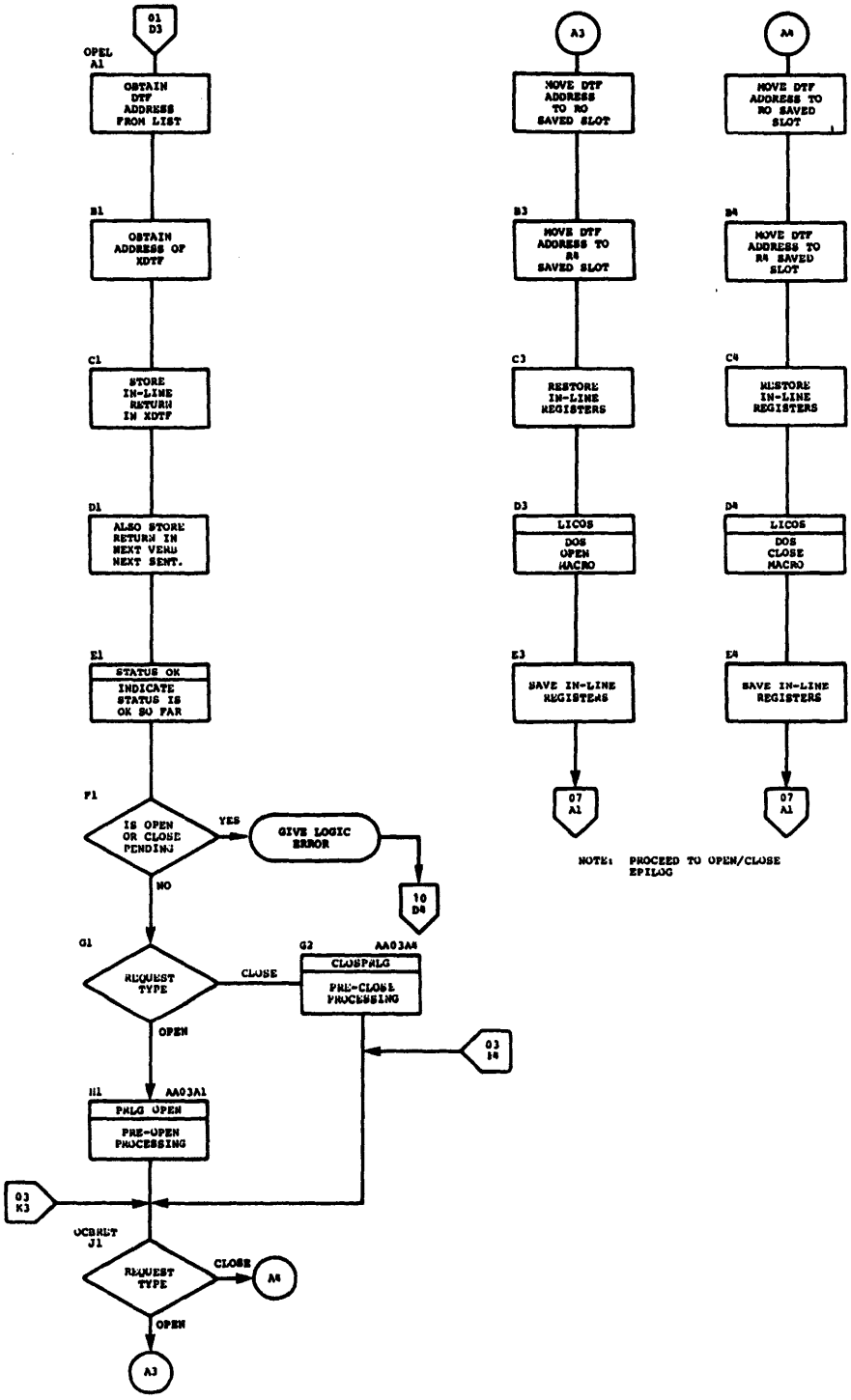


Chart F. SAM I/O (ILBDSIO0) (Part 3 of 10)

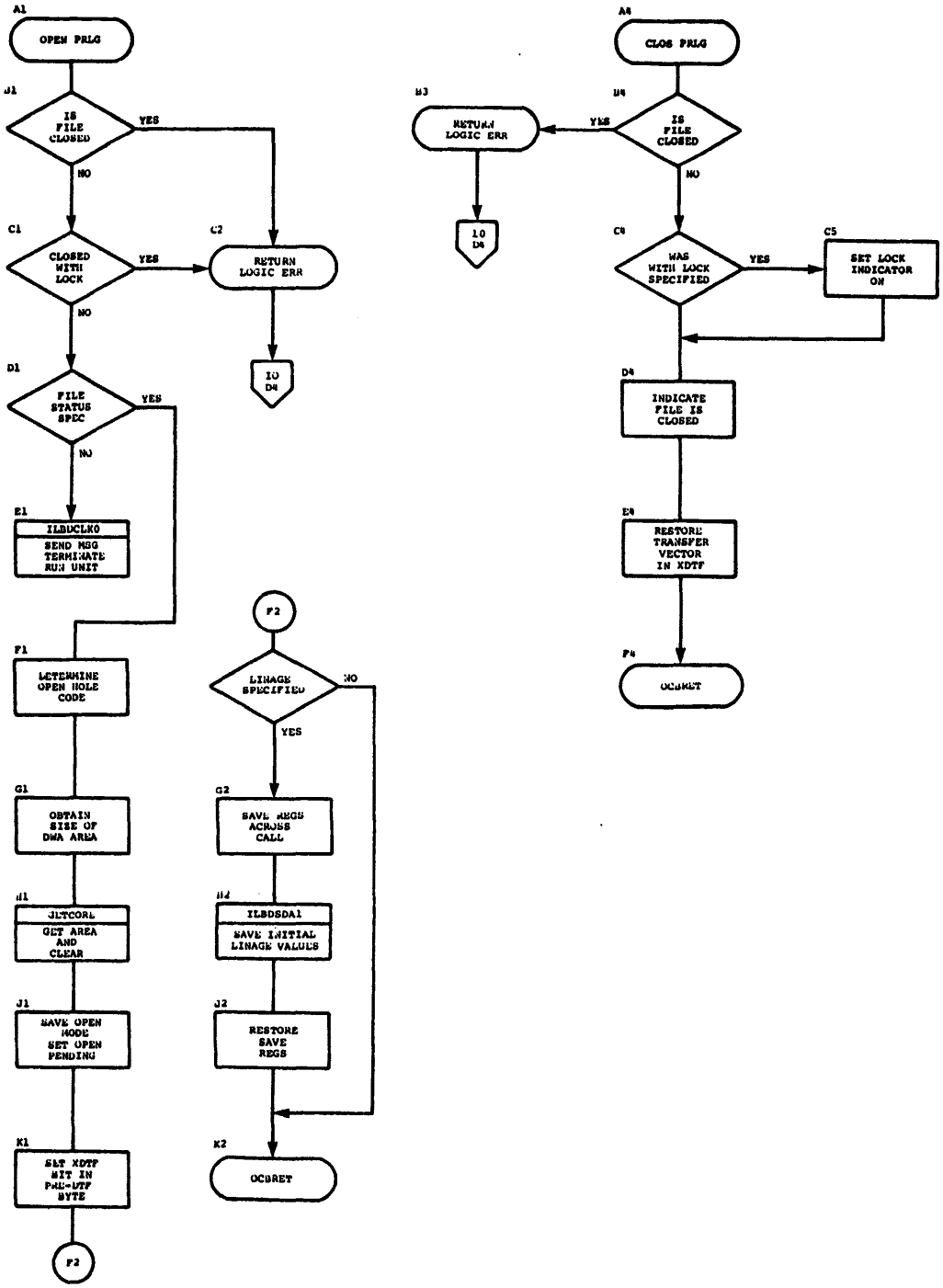


Chart F. SAM I/O (ILBDSIO0) (Part 4 of 10)

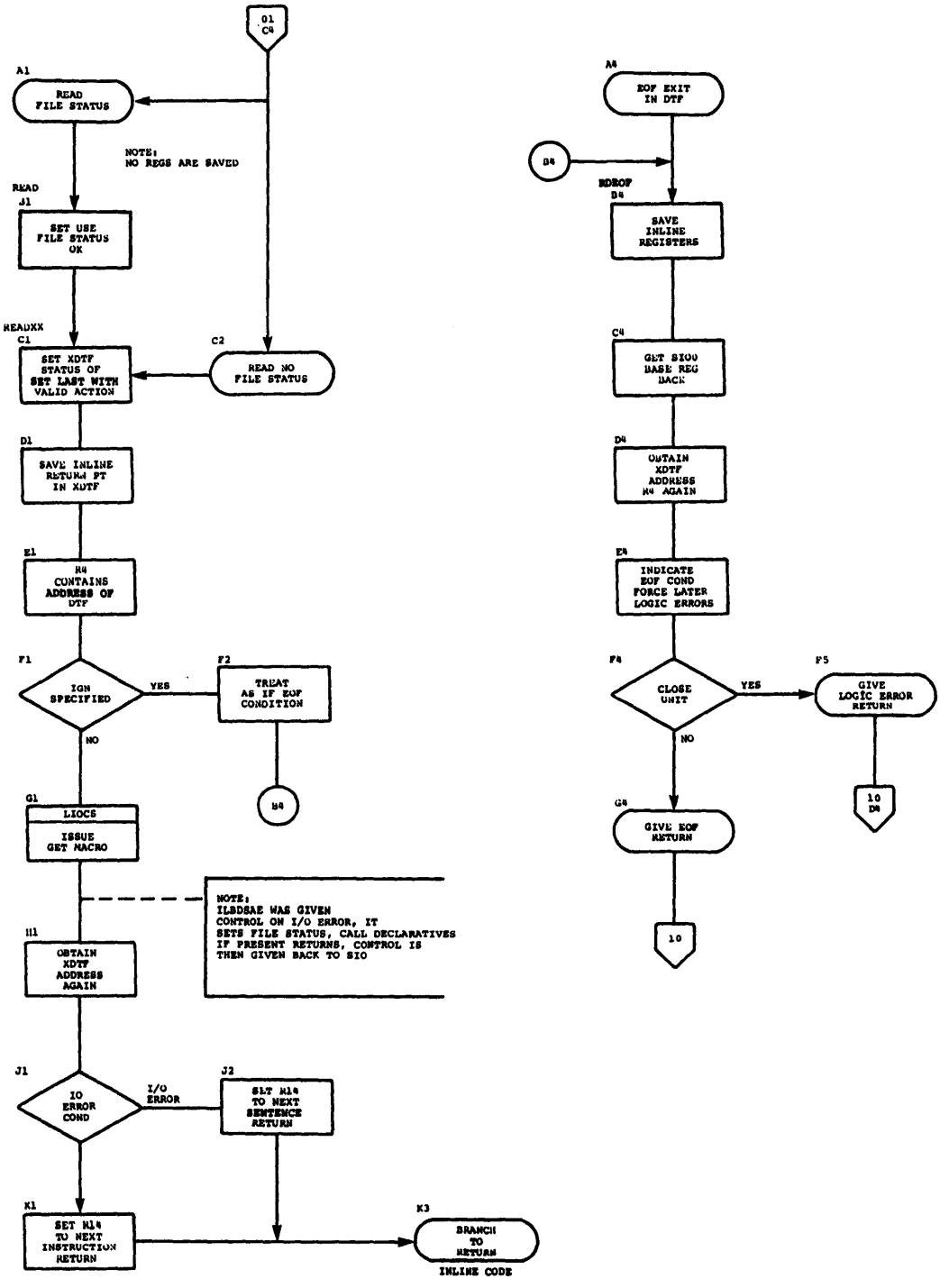


Chart F. SAM I/O (ILBDSIO0) (Part 5 of 10)

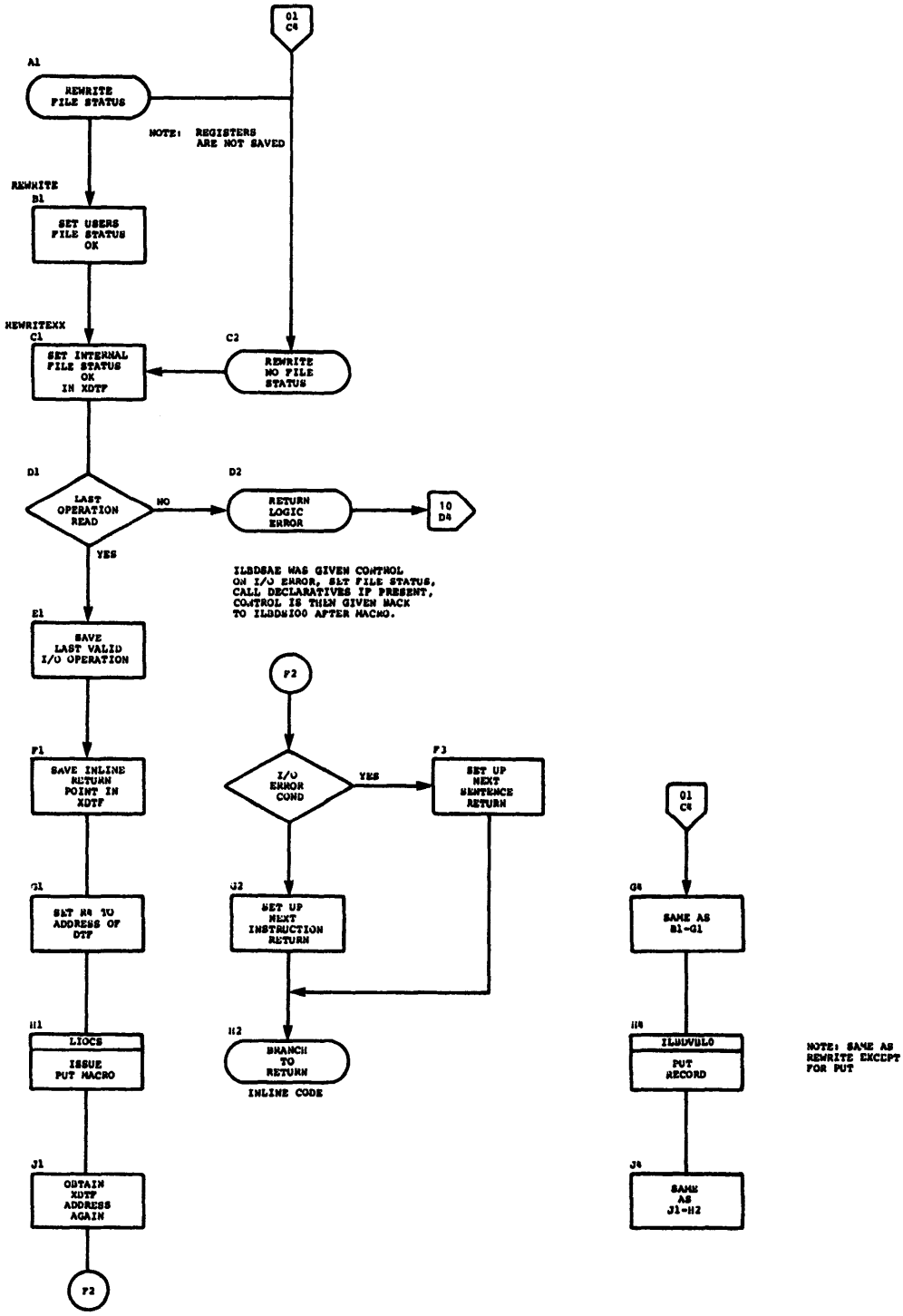


Chart F. SAM I/O (ILBDSIO0) (Part 6 of 10)

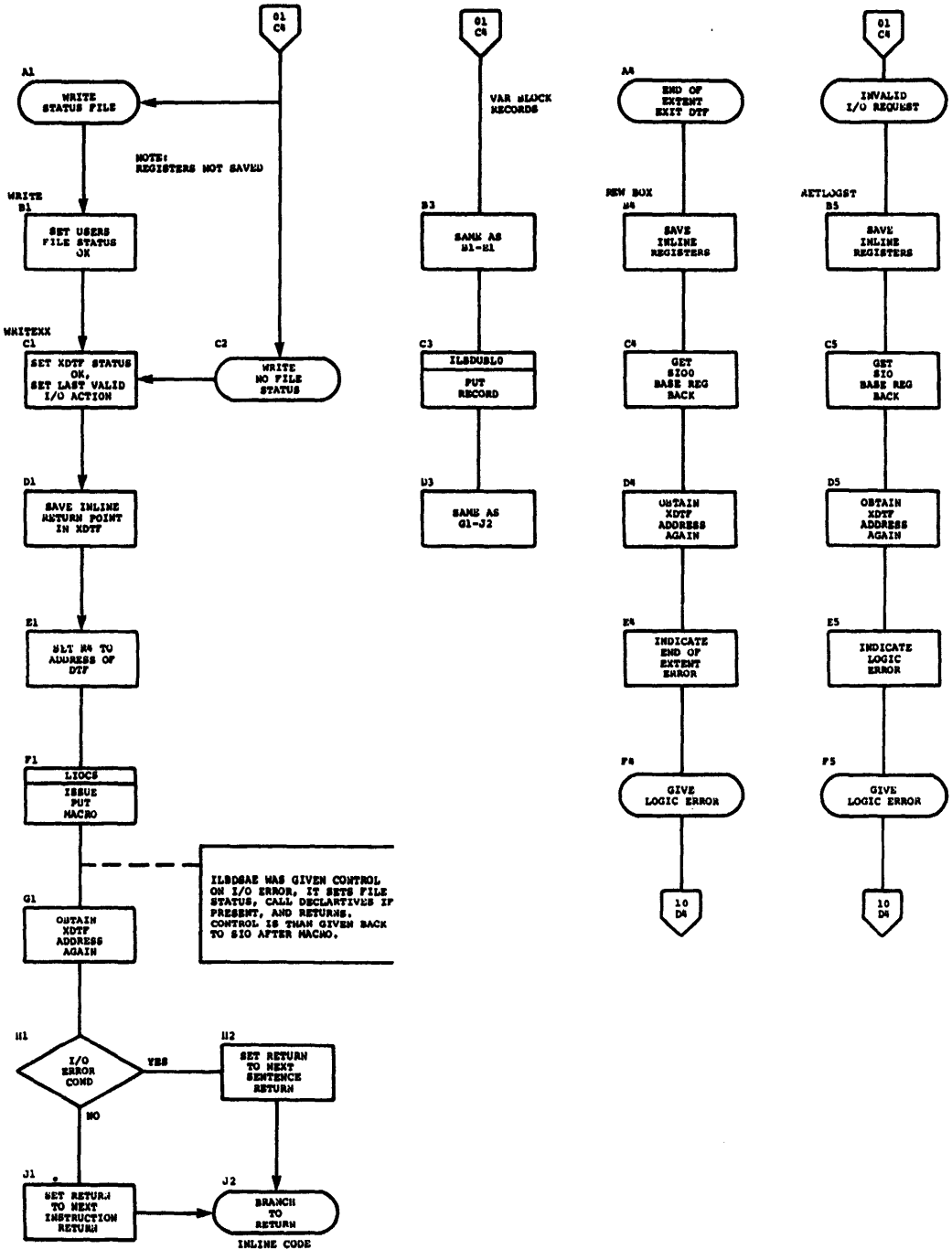


Chart F. SAM I/O (ILBDSIO0) (Part 7 of 10)

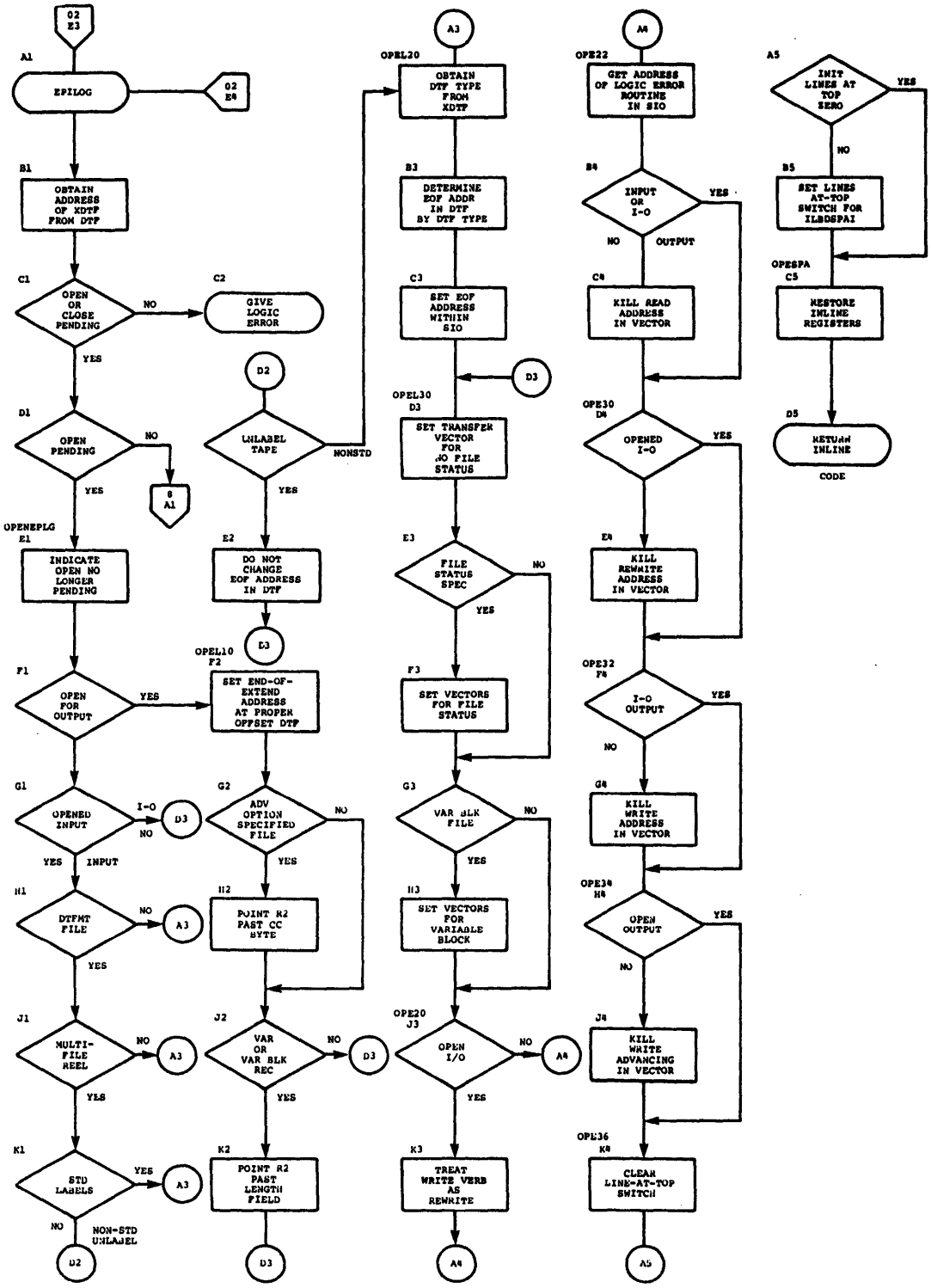


Chart F. SAM I/O (ILBDSIO0) (Part 8 of 10)

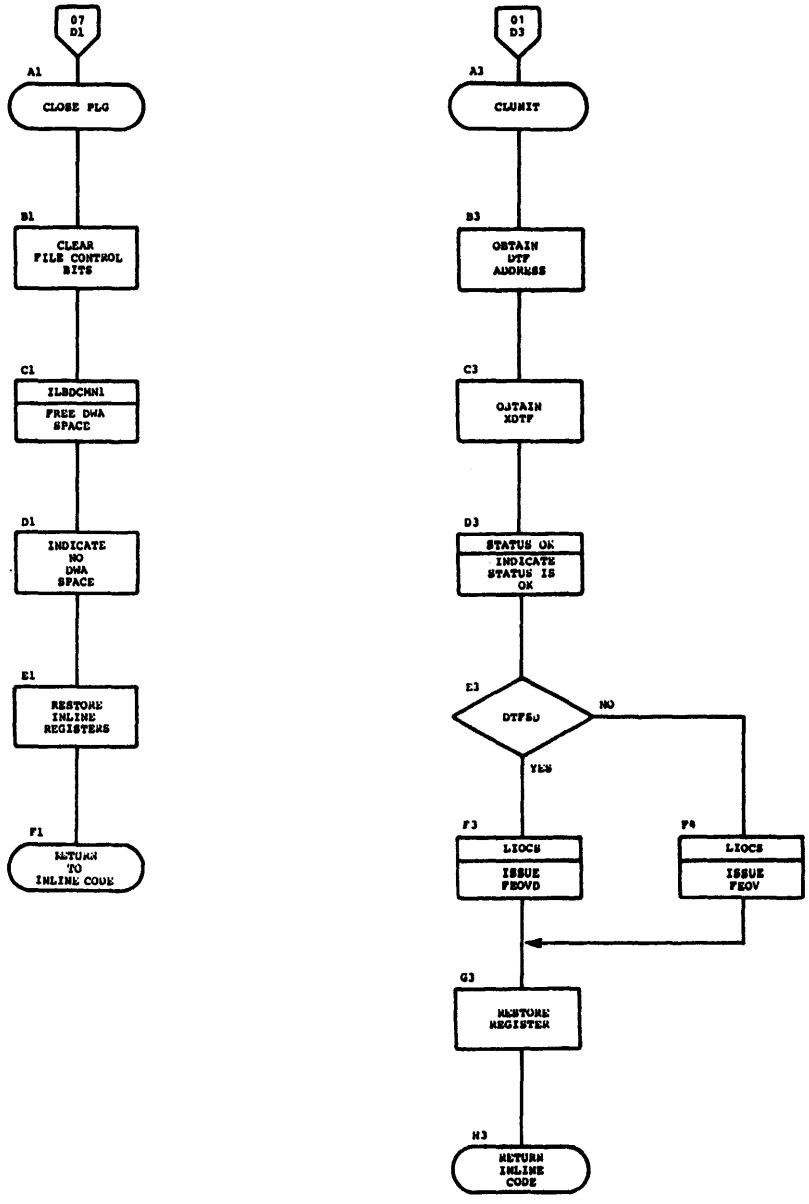


Chart F. SAM I/O (ILBDSIO0) (Part 9 of 10)

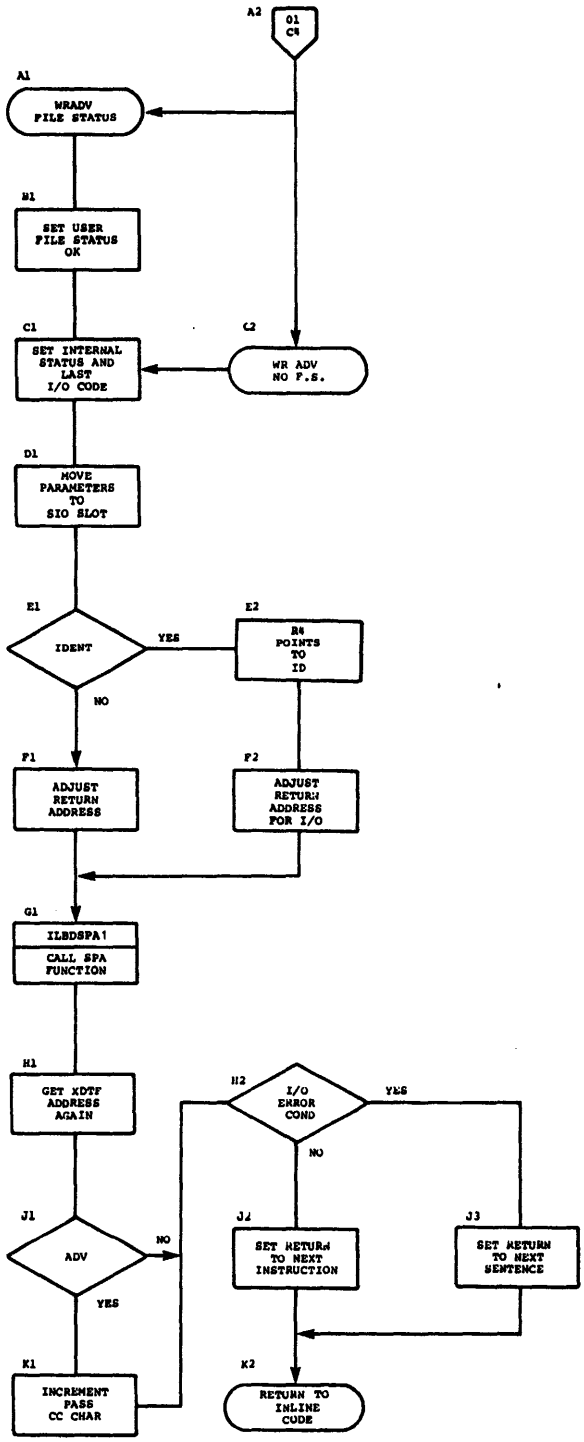


Chart F. SAM I/O (ILBDSIO0) (Part 10 of 10)

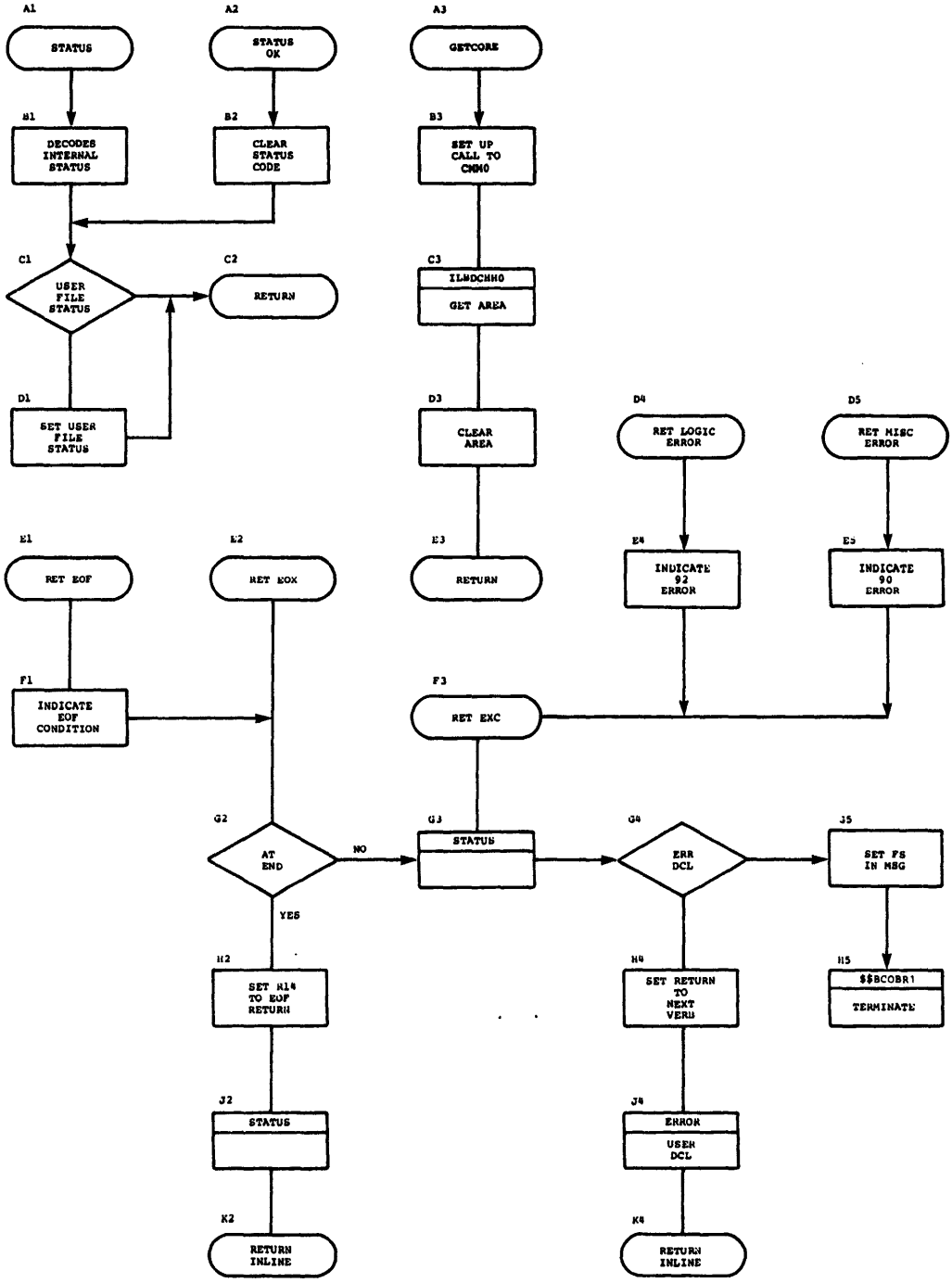


Chart FA. SA Printer Spacing (ILBDSPA0) (Part 1 of 7)

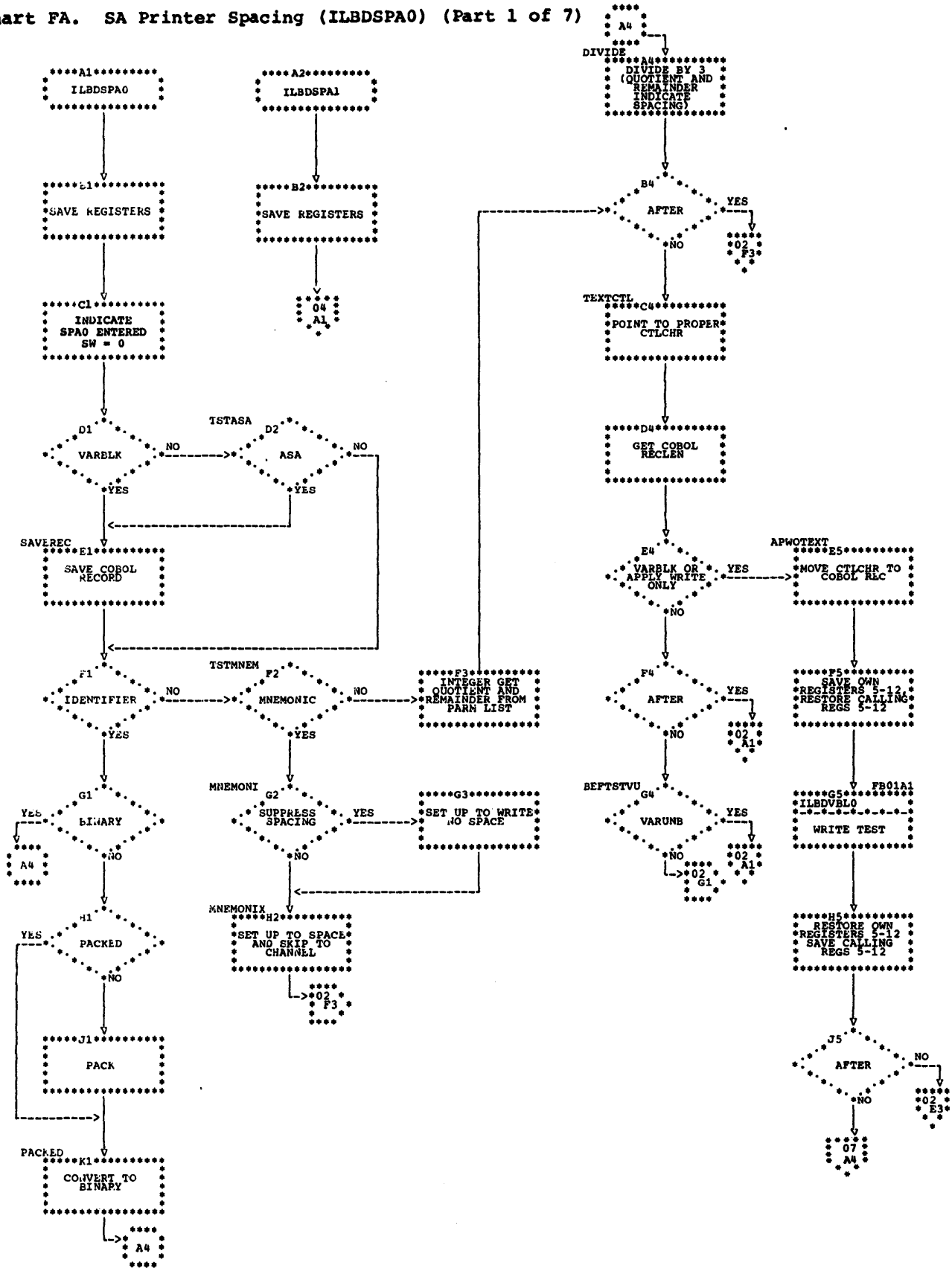


Chart FA. SA Printer Spacing (ILBDSPA0) (Part 2 of 7)

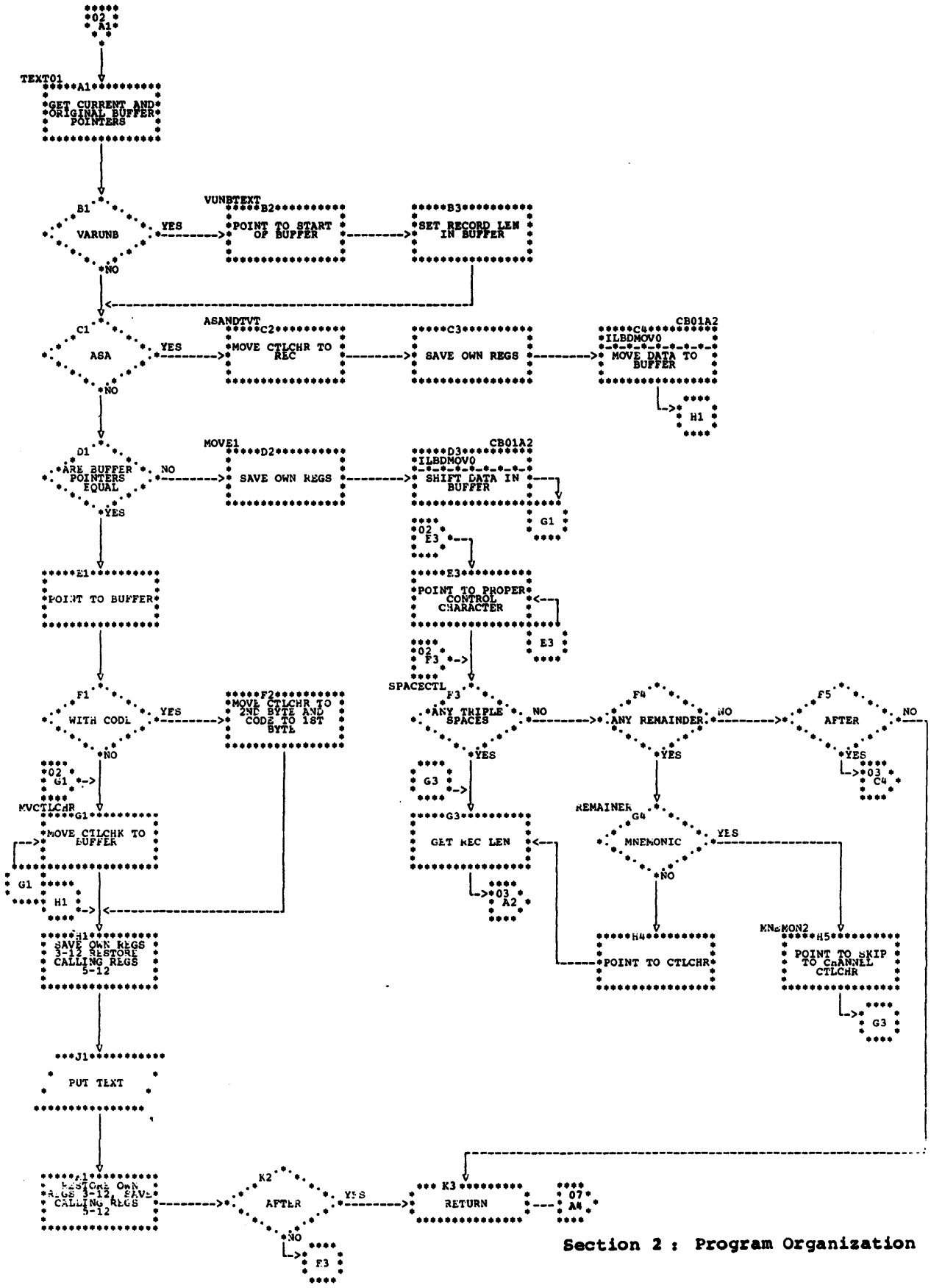


Chart FA. SA Printer Spacing (ILBDSPA0) (Part 3 of 7)

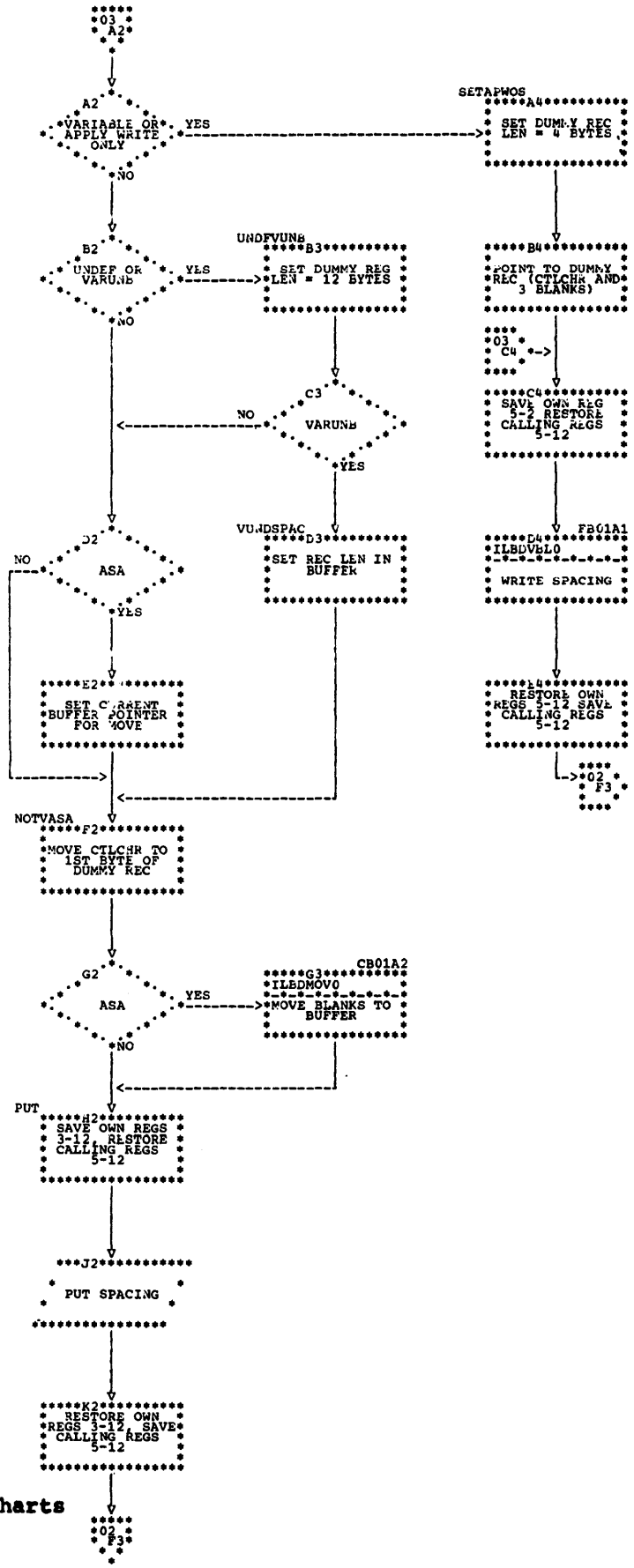


Chart FA. SA Printer Spacing (ILBDSPA0) (Part 4 of 7)

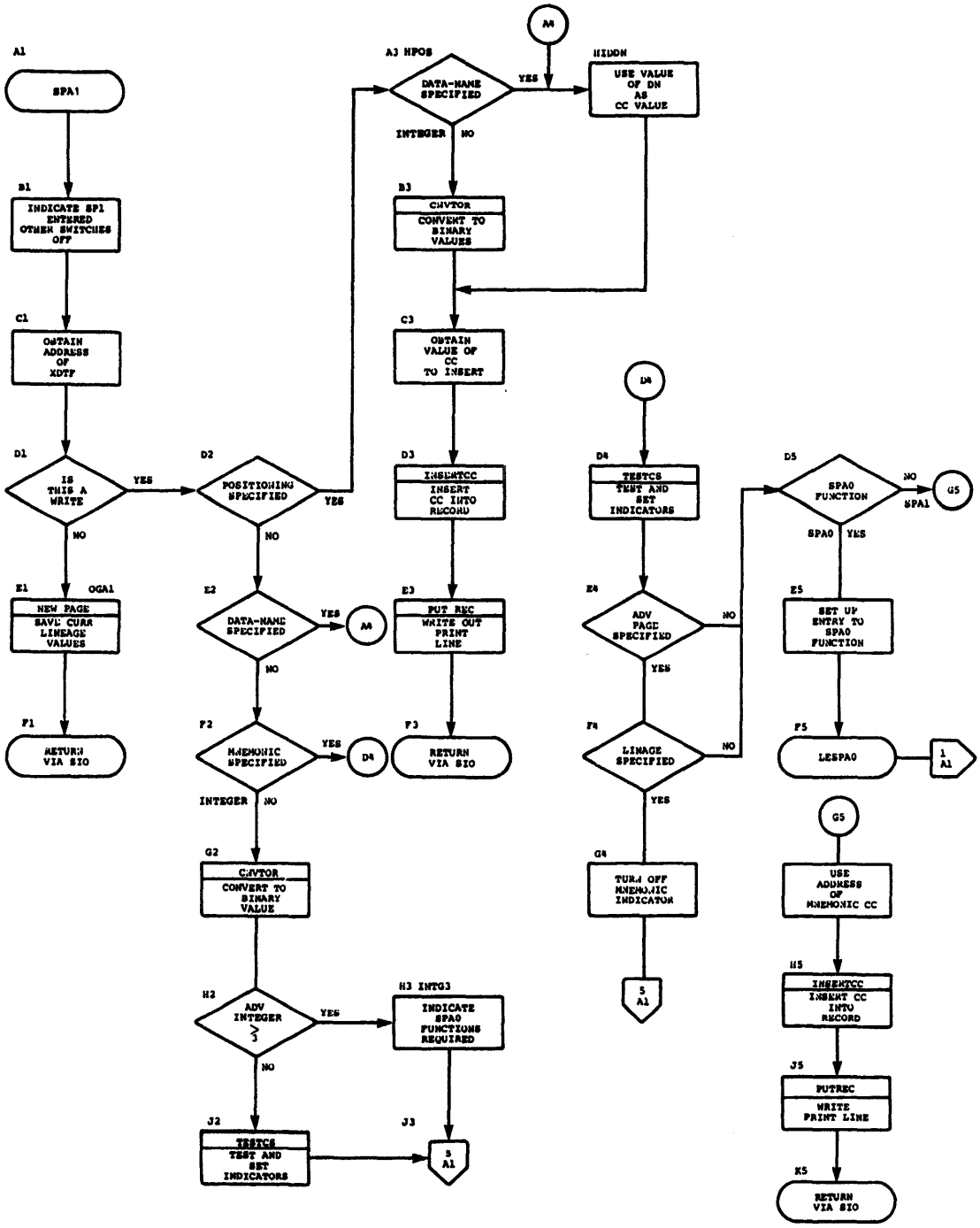


Chart FA. SA Printer Spacing (ILBDSPA0) (Part 5 of 7)

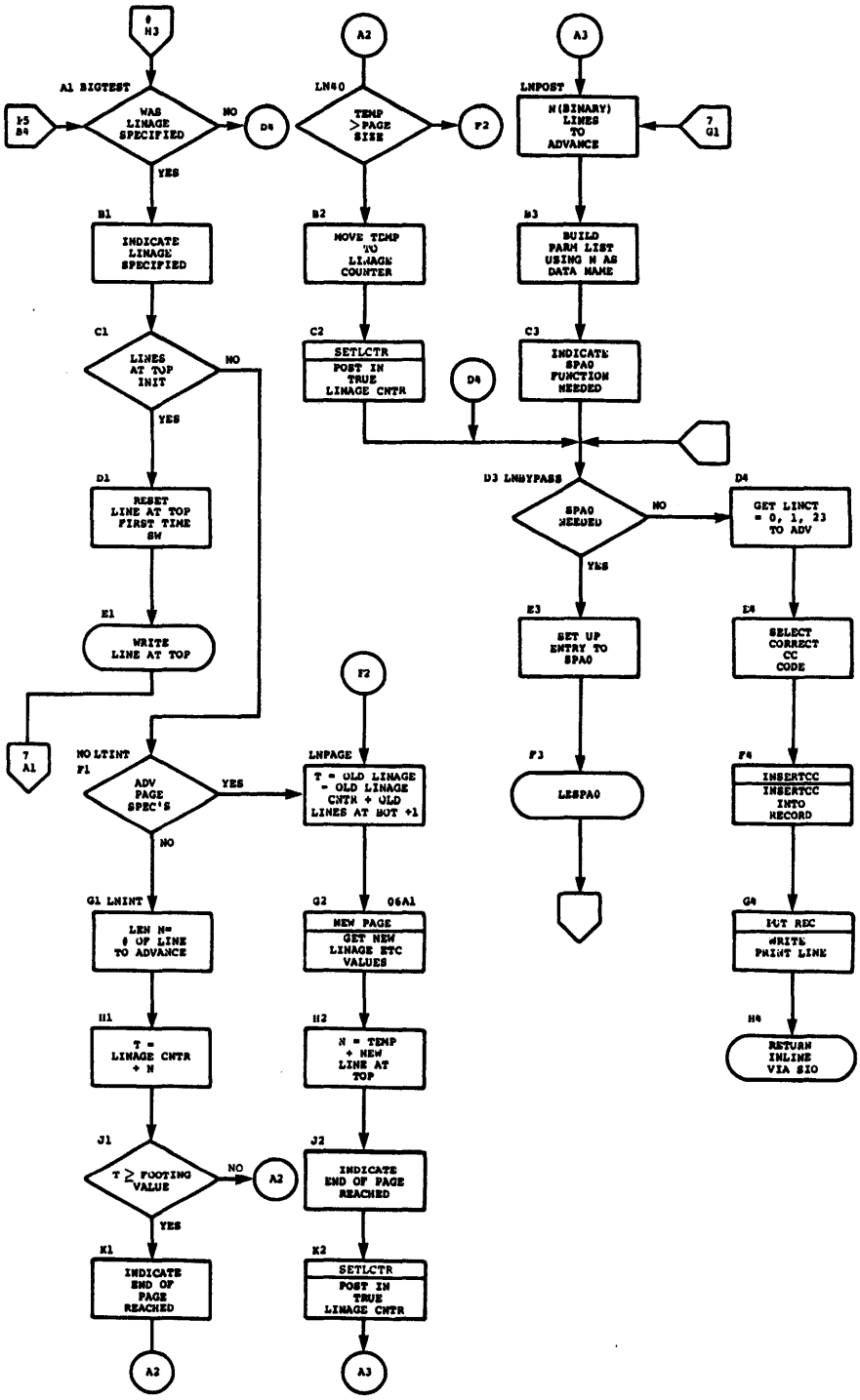


Chart FA. SA Printer Spacing (ILBDSPA0) (Part 6 of 7)

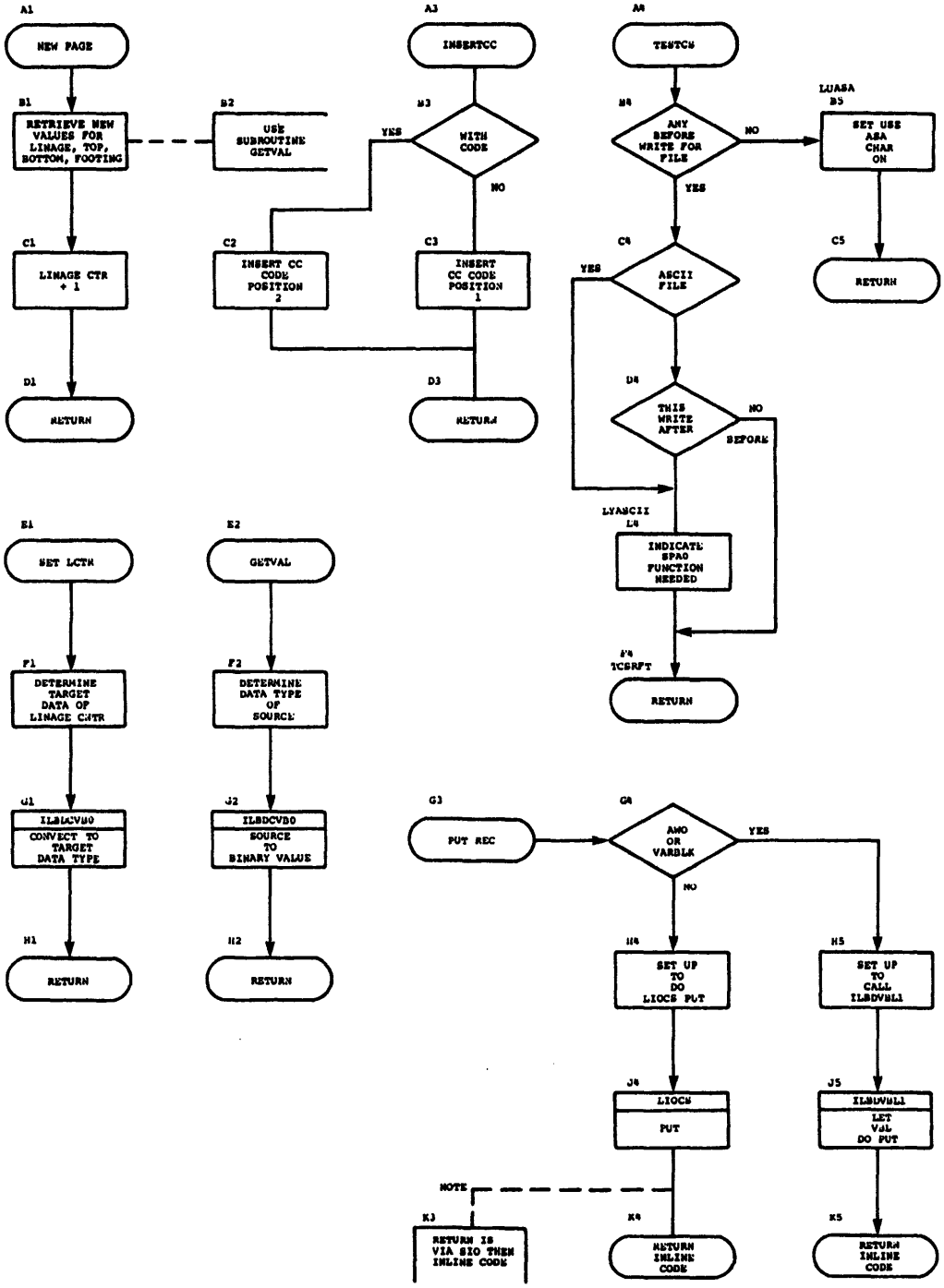


Chart FA. SA Printer Spacing (ILBDSPA0) (Part 7 of 7)

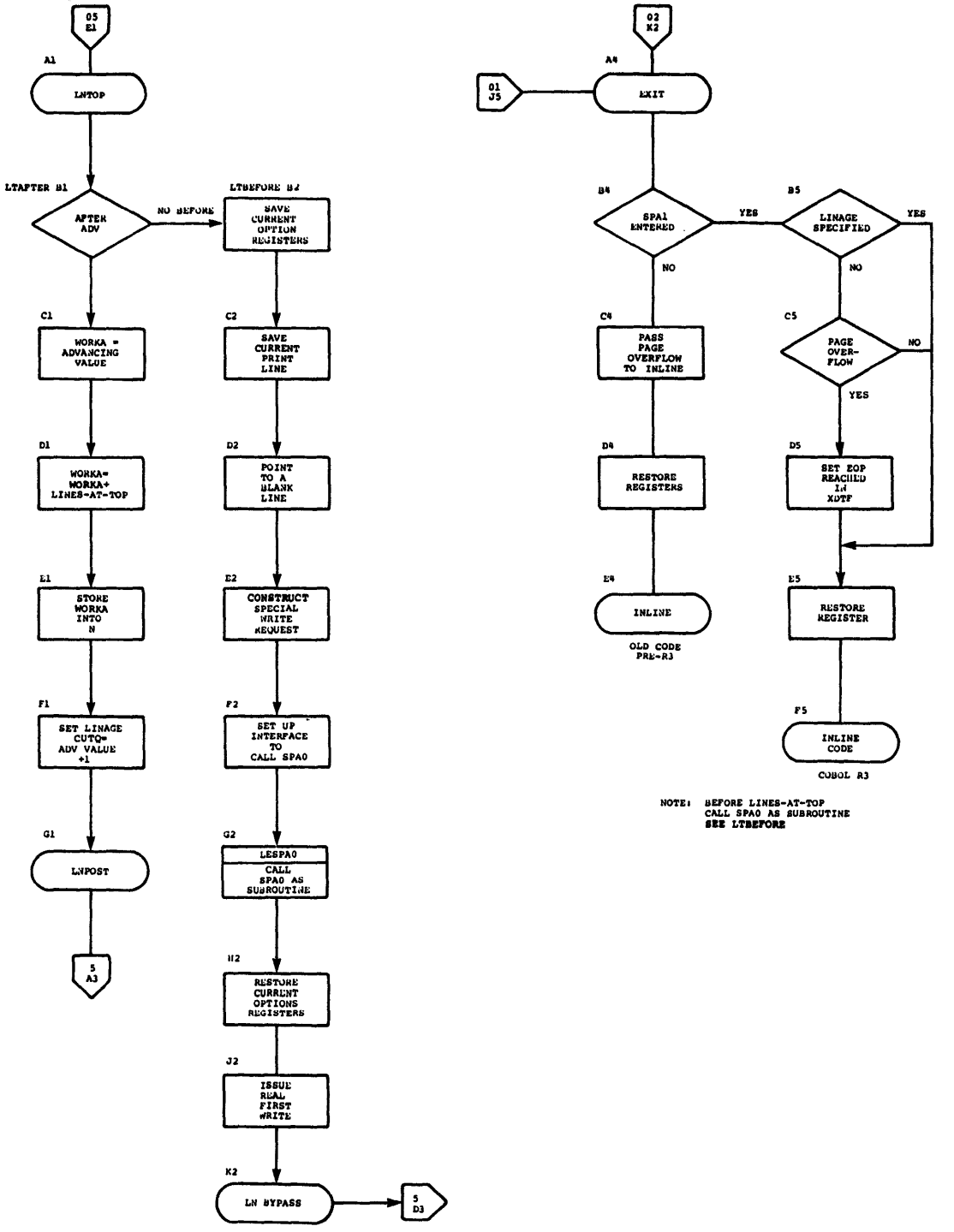


Chart FB. SA Variable Length Record Output (ILBDVBL0)

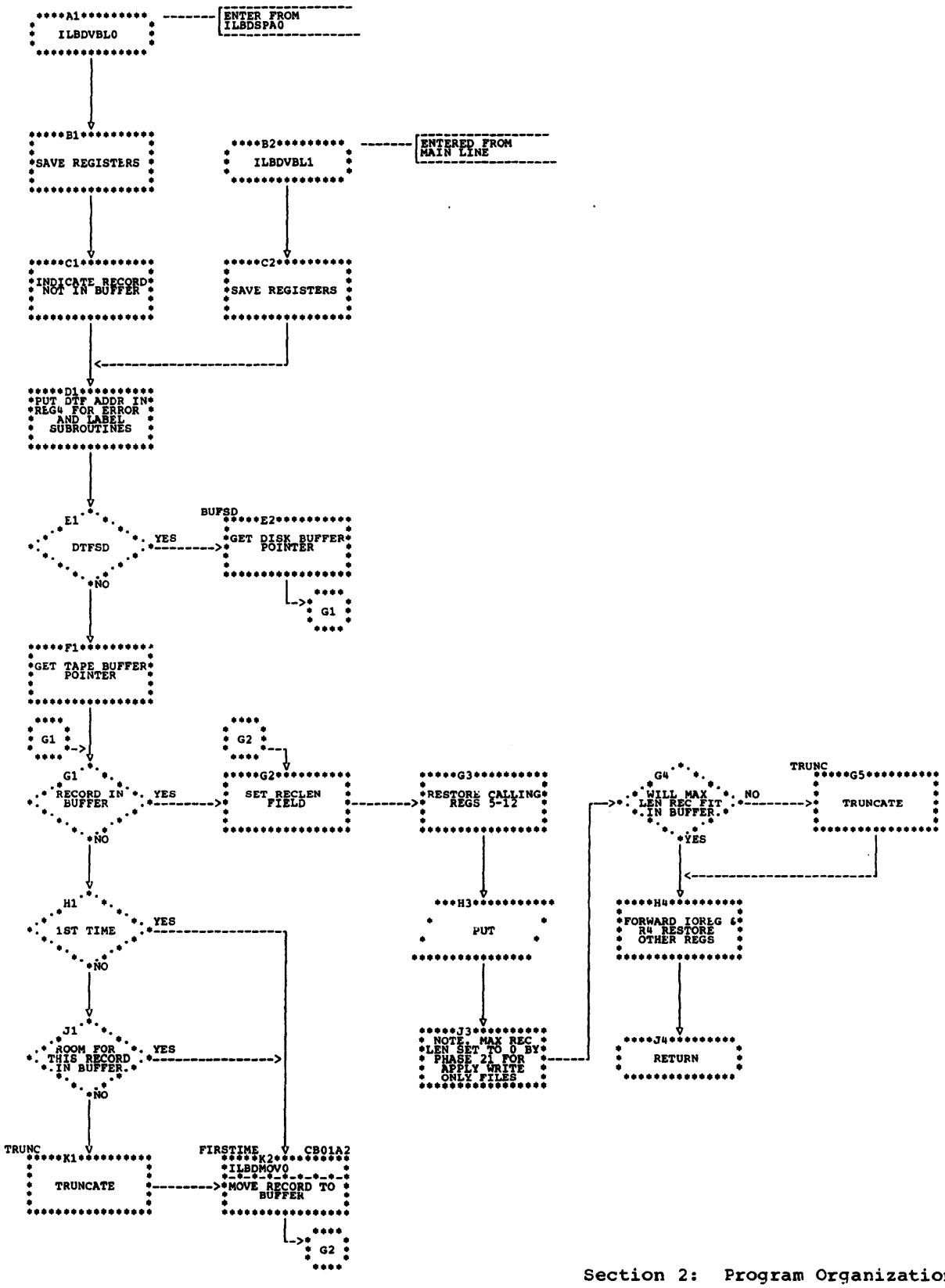


Chart FC. SA Error Routine (ILBDSAE0)

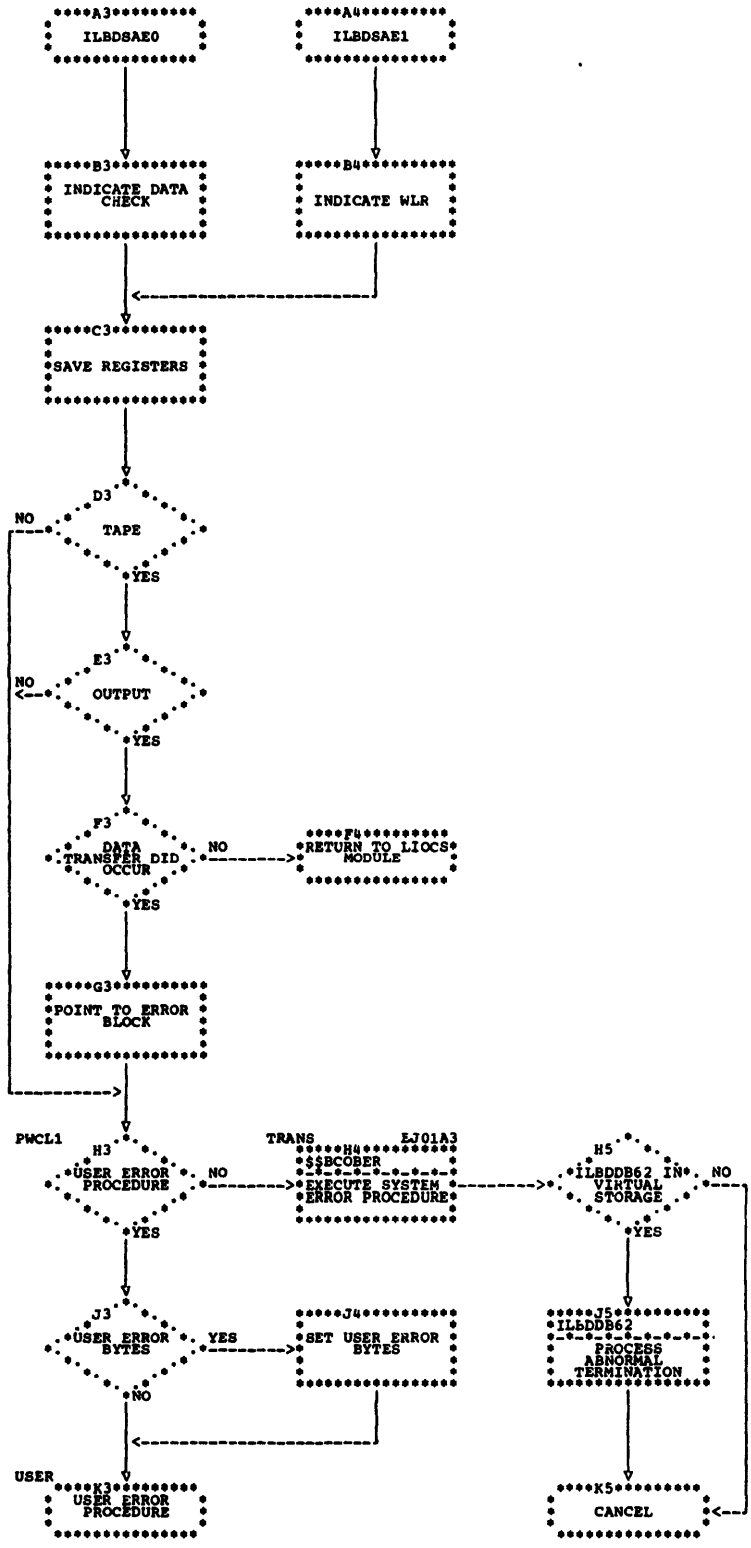


Chart FD. SA Tape Pointer (ILBDIML0)

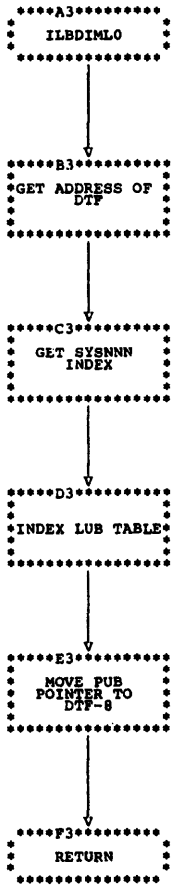


Chart FE. SA Position Multiple File Tapes (ILBDMFT0)

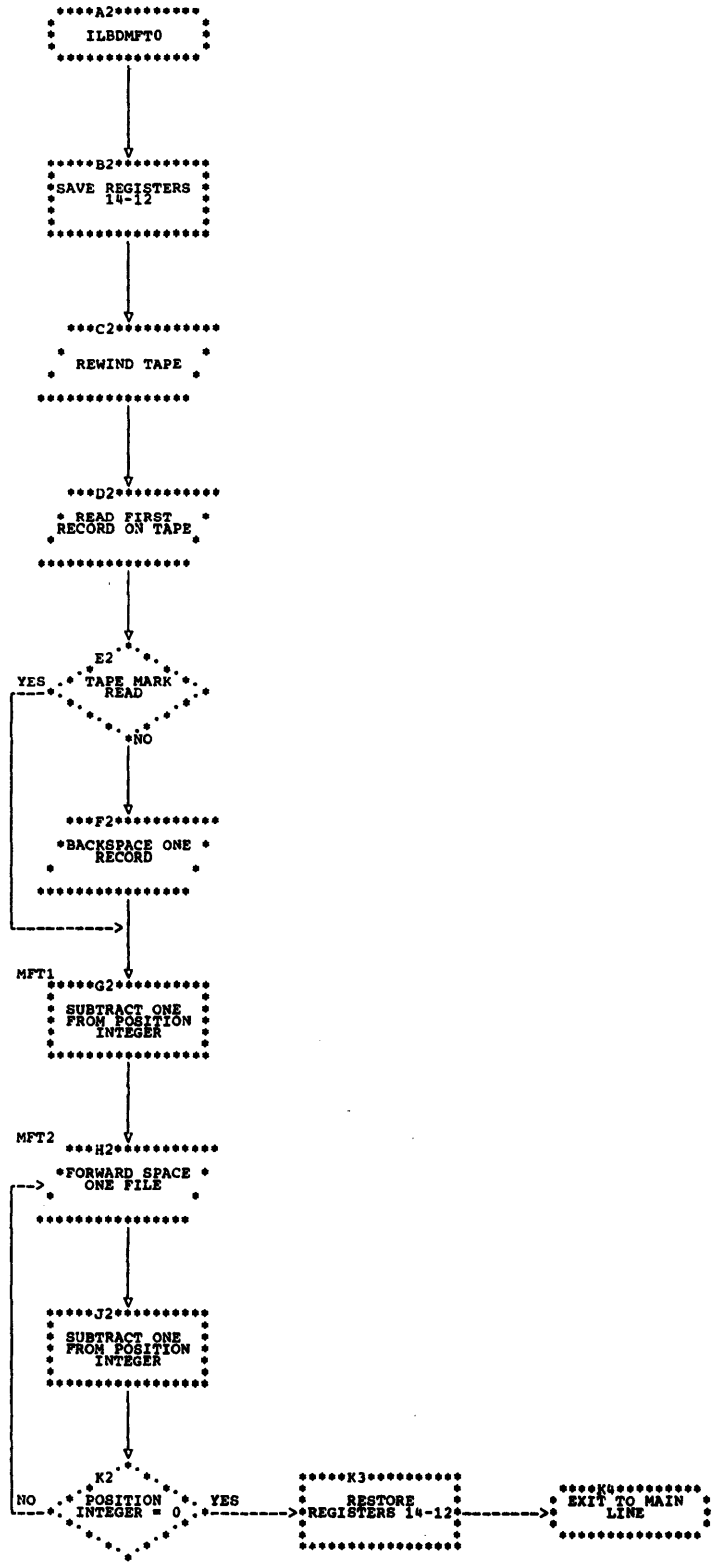


Chart FF. SA Test Tape File (ILBDMVE0)

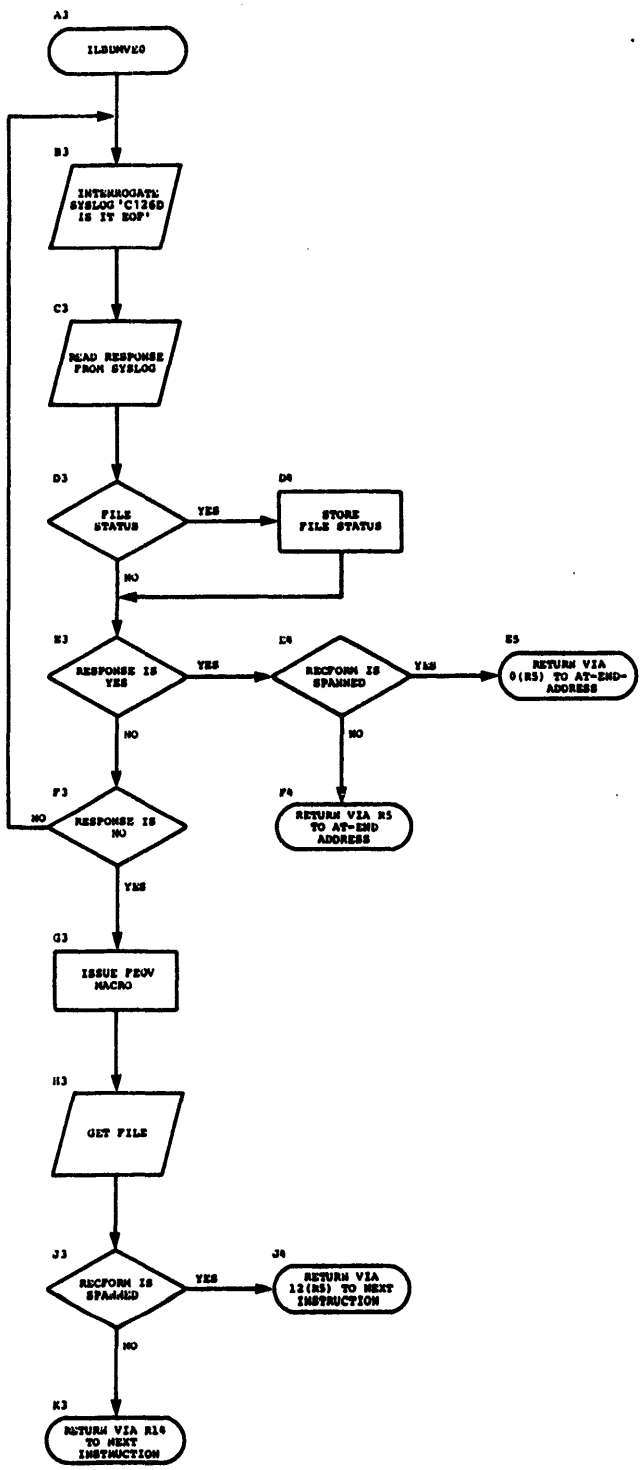


Chart FG. SA STXIT Macro Instruction (ILBDABX0)

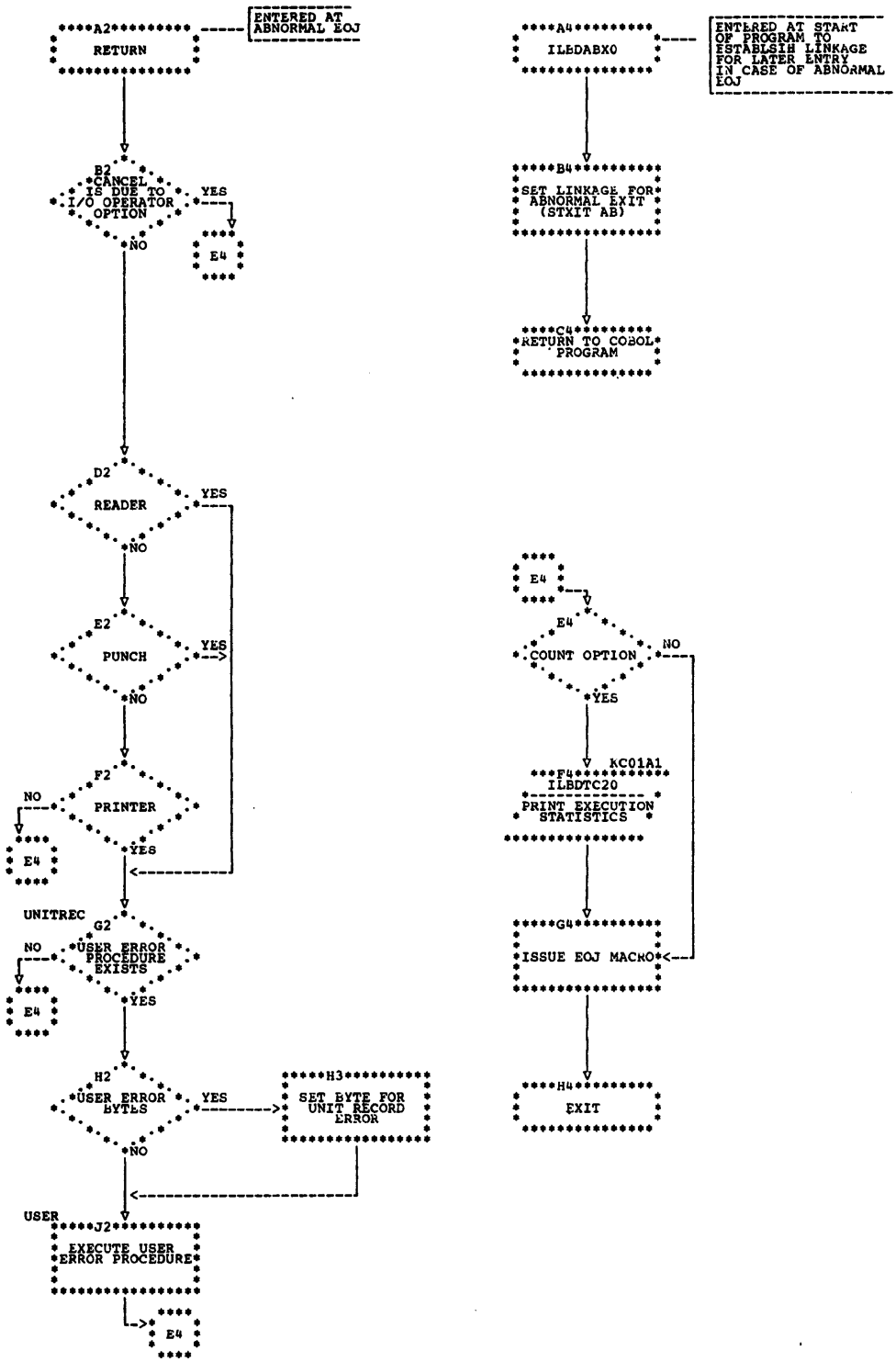


Chart FH. SA Reposition Tape ($$BFCMUL)

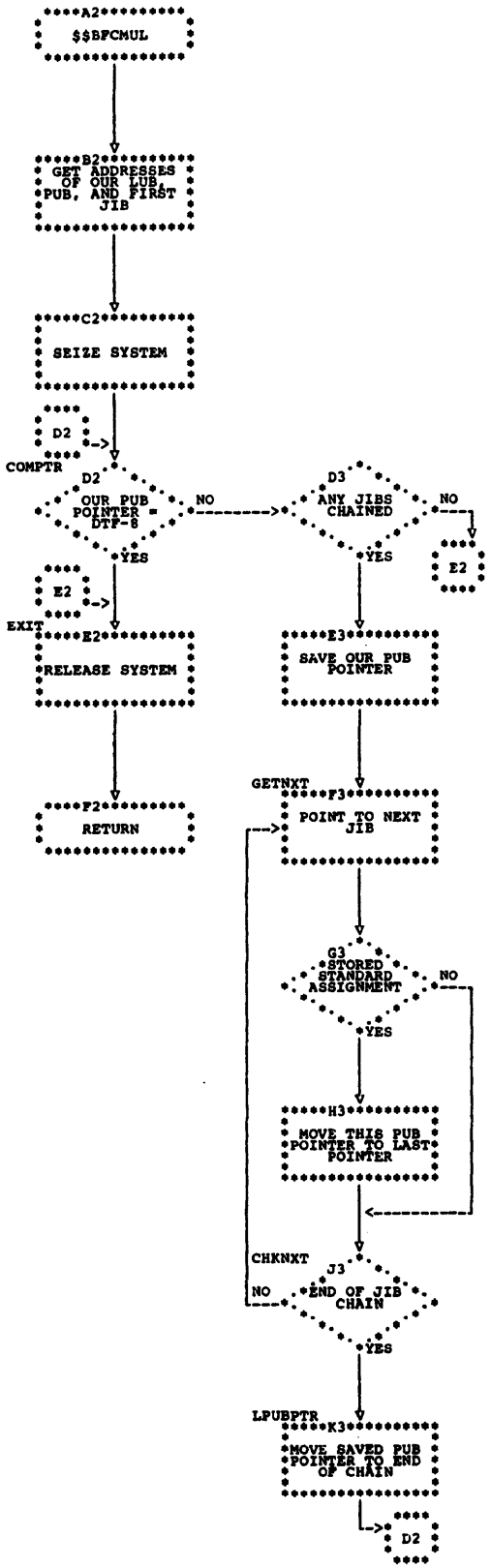


Chart GA. ISAM READ and WRITE (ILBDISM0)

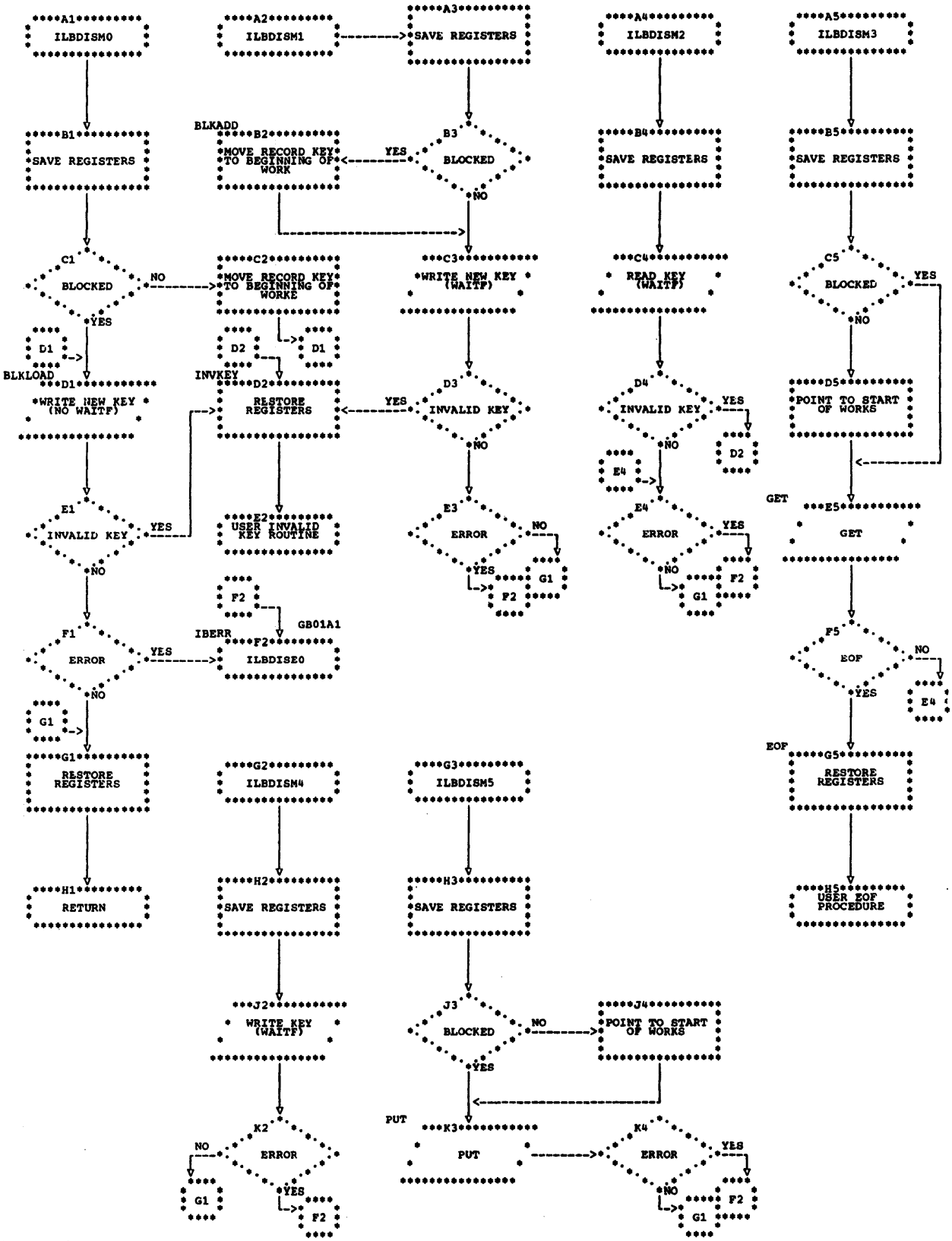


Chart GB. ISAM Error (ILBDISE0)

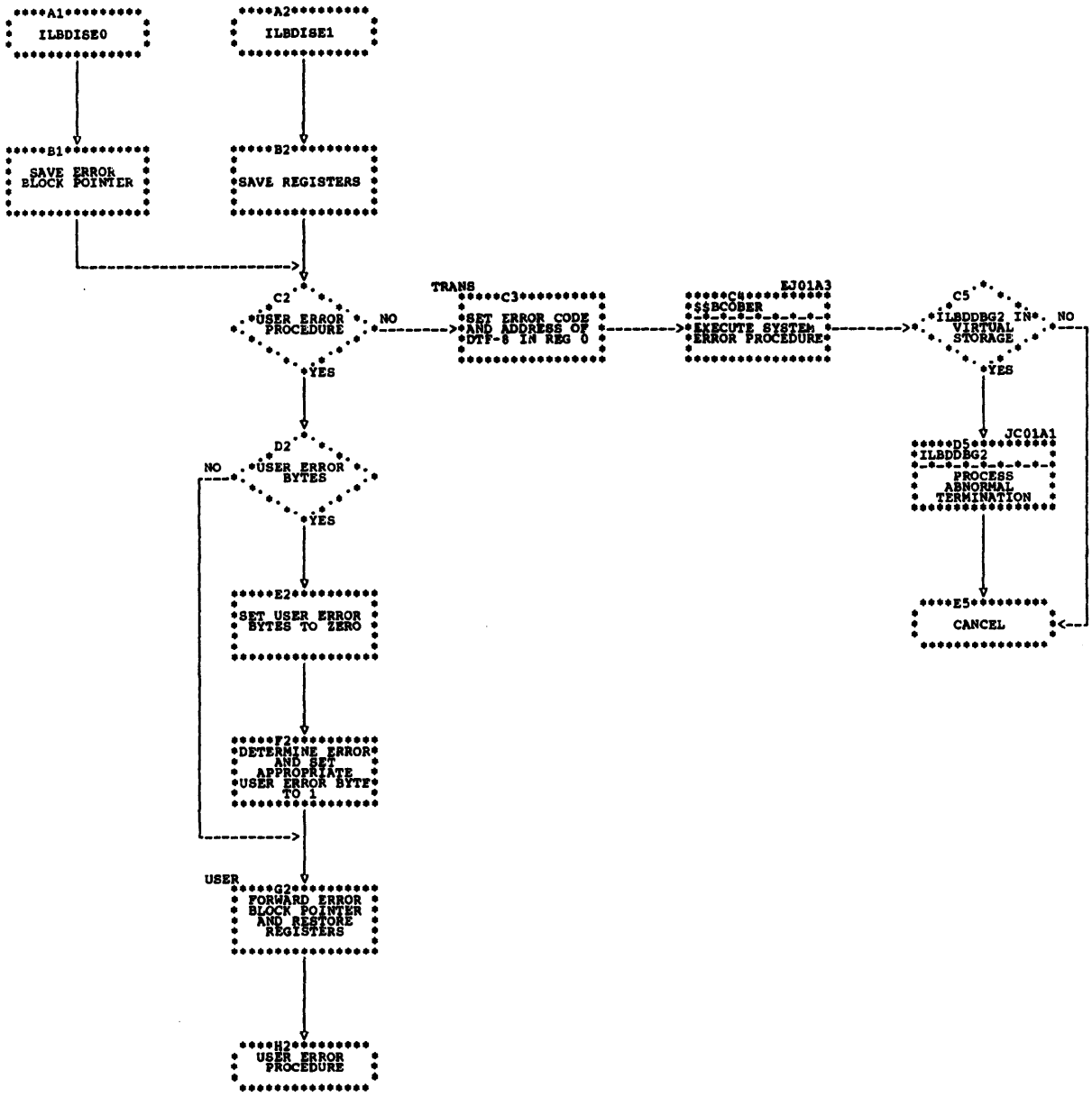


Chart GC. ISAM Start (ILBDSTR0)

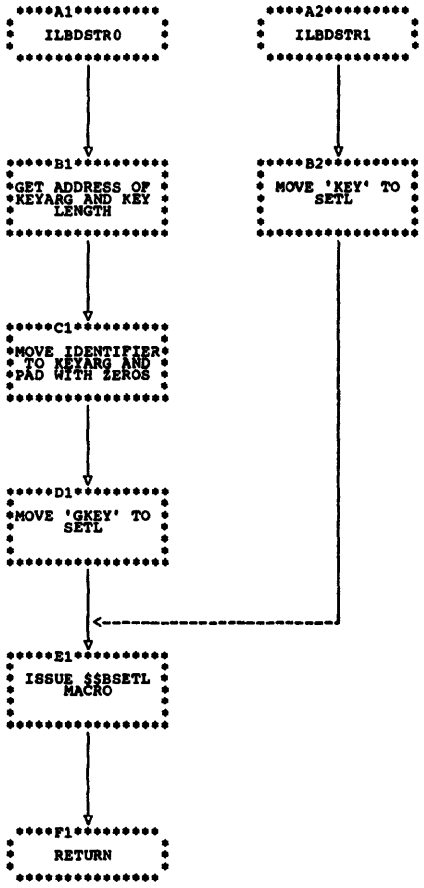


Chart HA. DA Close Unit (ILBDCRD0)

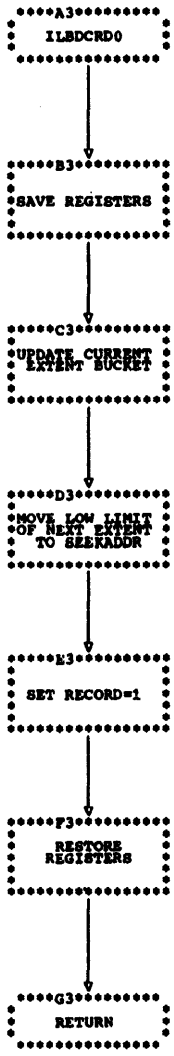


Chart HB. DA Close Unit for Relative Track (ILBDRCR0)

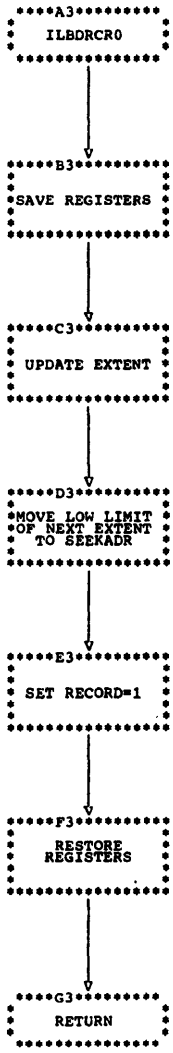


Chart HC. DA Extent Processor (ILBDXTN0)

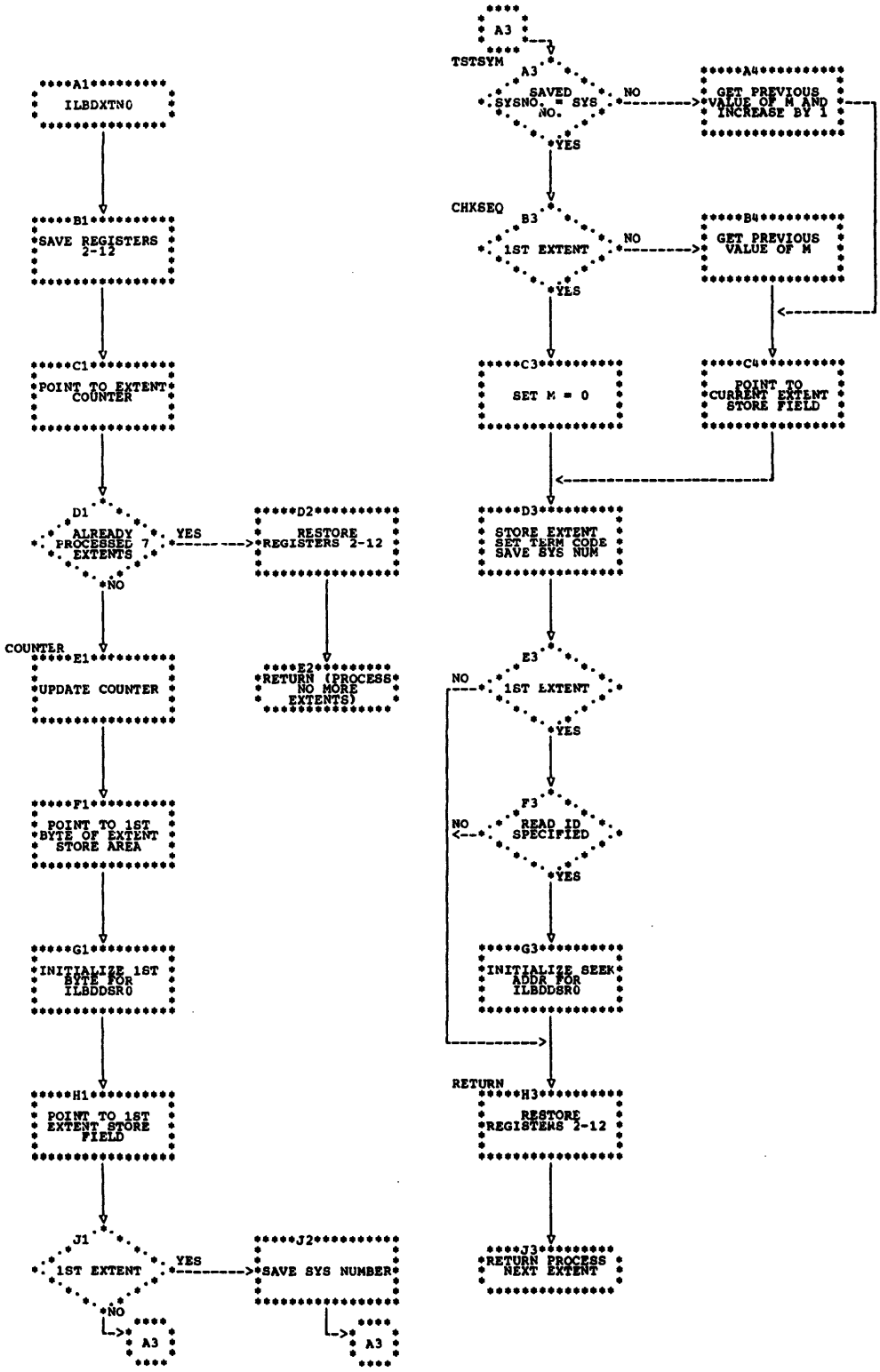


Chart HD. DA Sequential Read (ILBDDSR0)

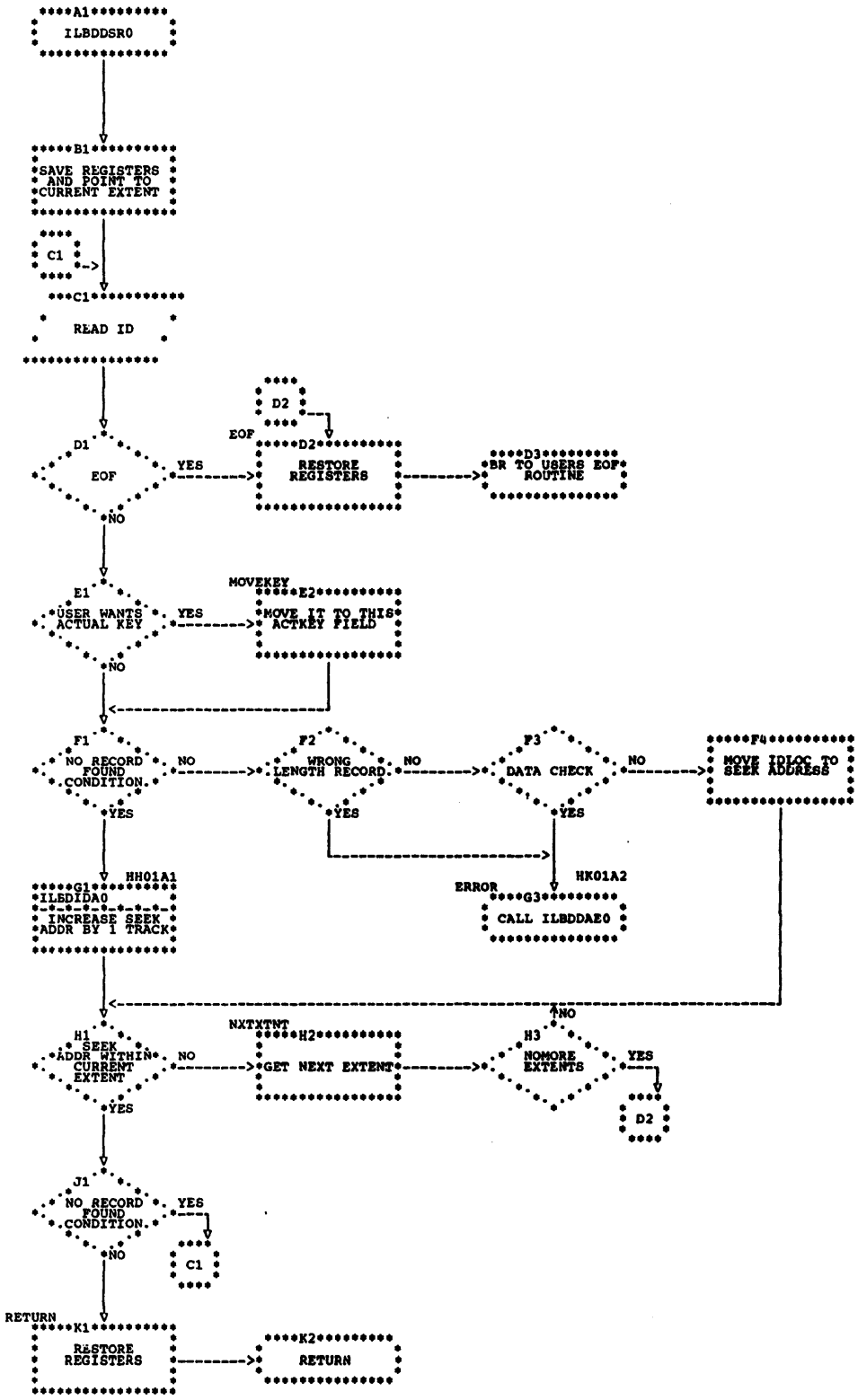


Chart HE. DA Sequential Read for Relative Track (ILBDRDS0)

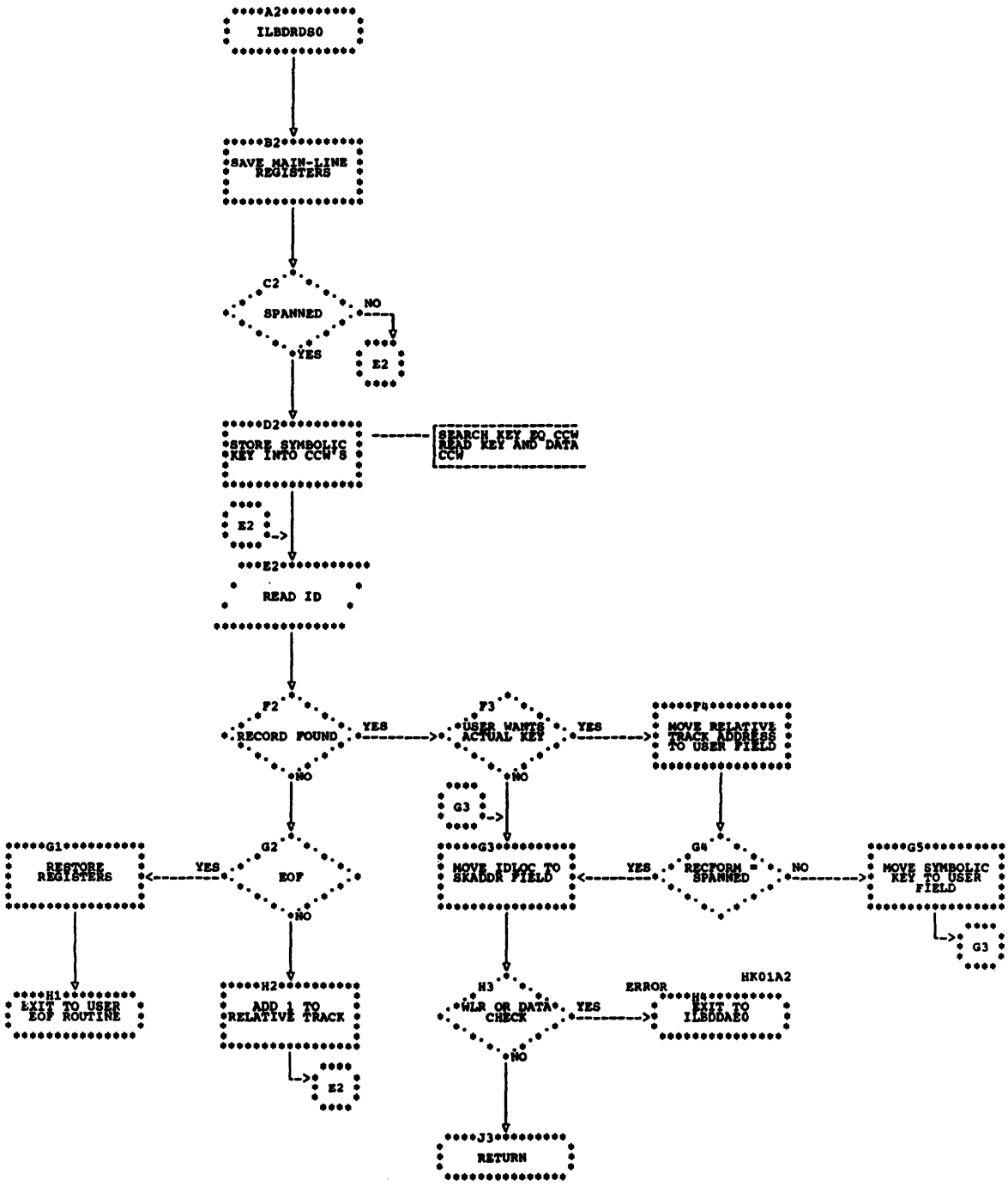


Chart HF. DA RZERO Record (ILBDFMT0)

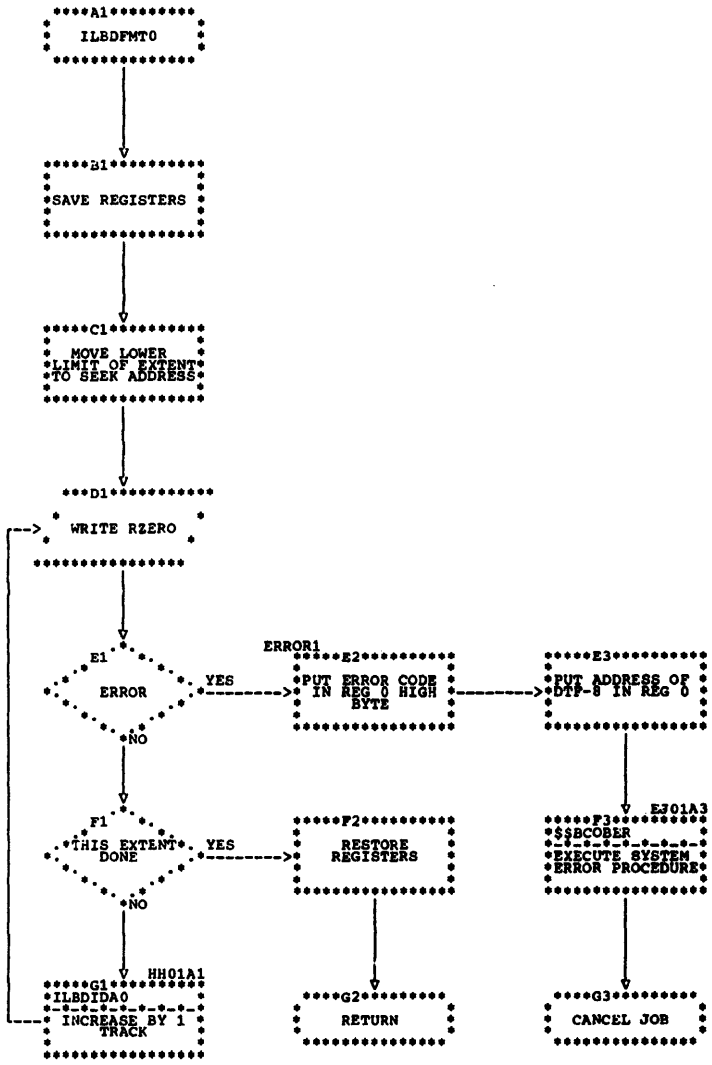


Chart HG. DA RZERO Record for Relative Track (ILBDRFM0)

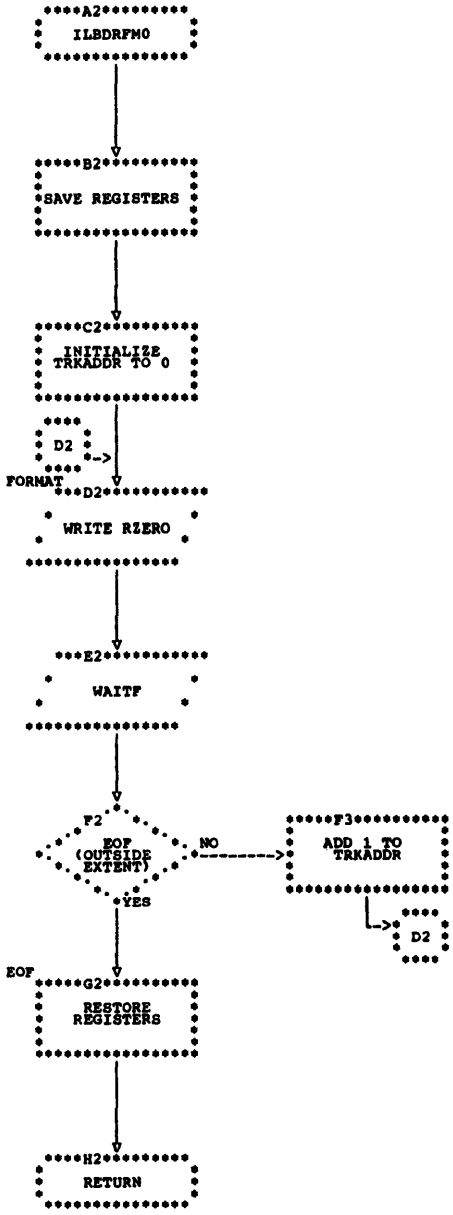


Chart HH. DA Increase SEEK Address (ILBDIDA0)

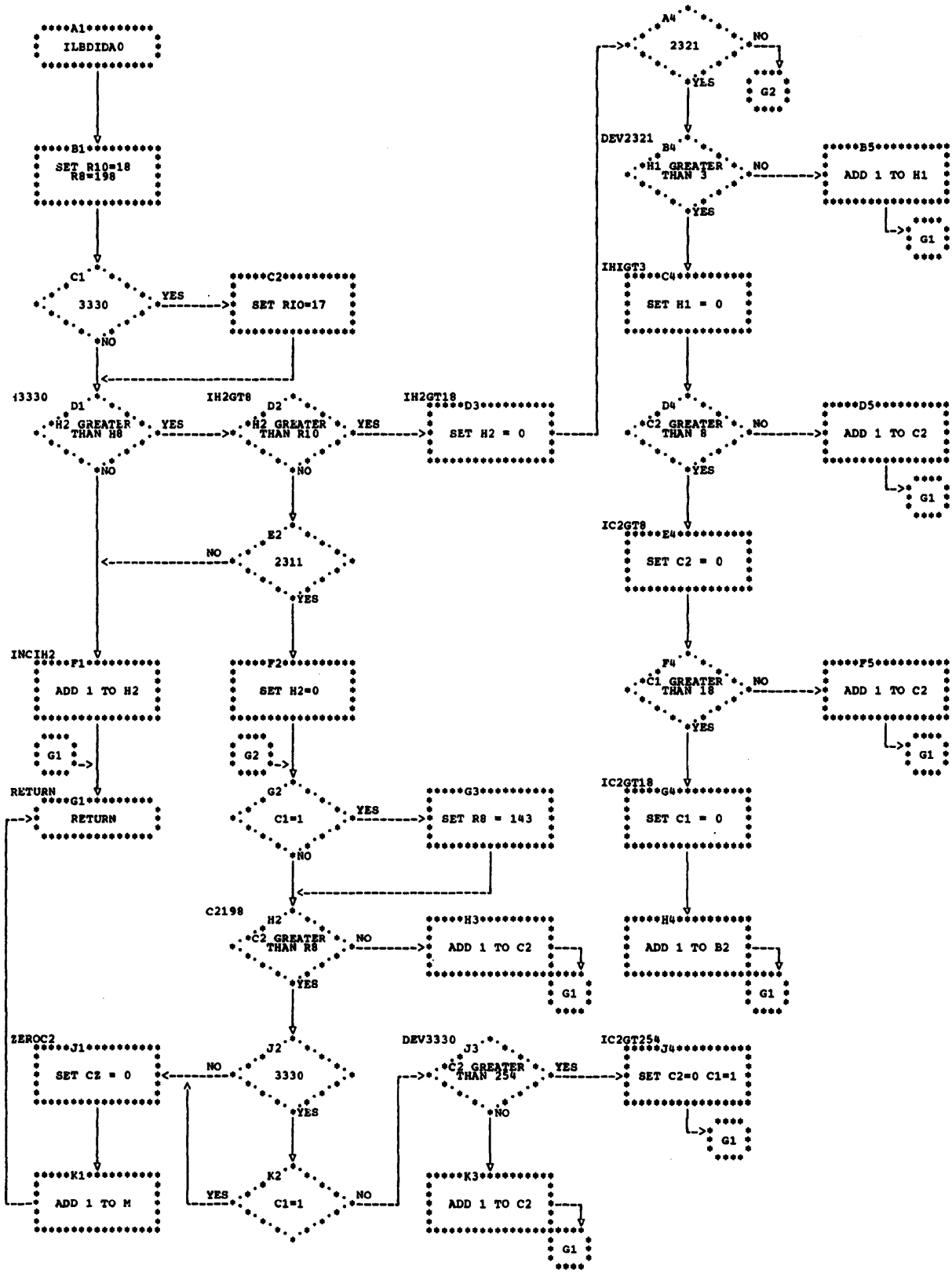


Chart HI. DA READ and WRITE (ILBDDIO0)

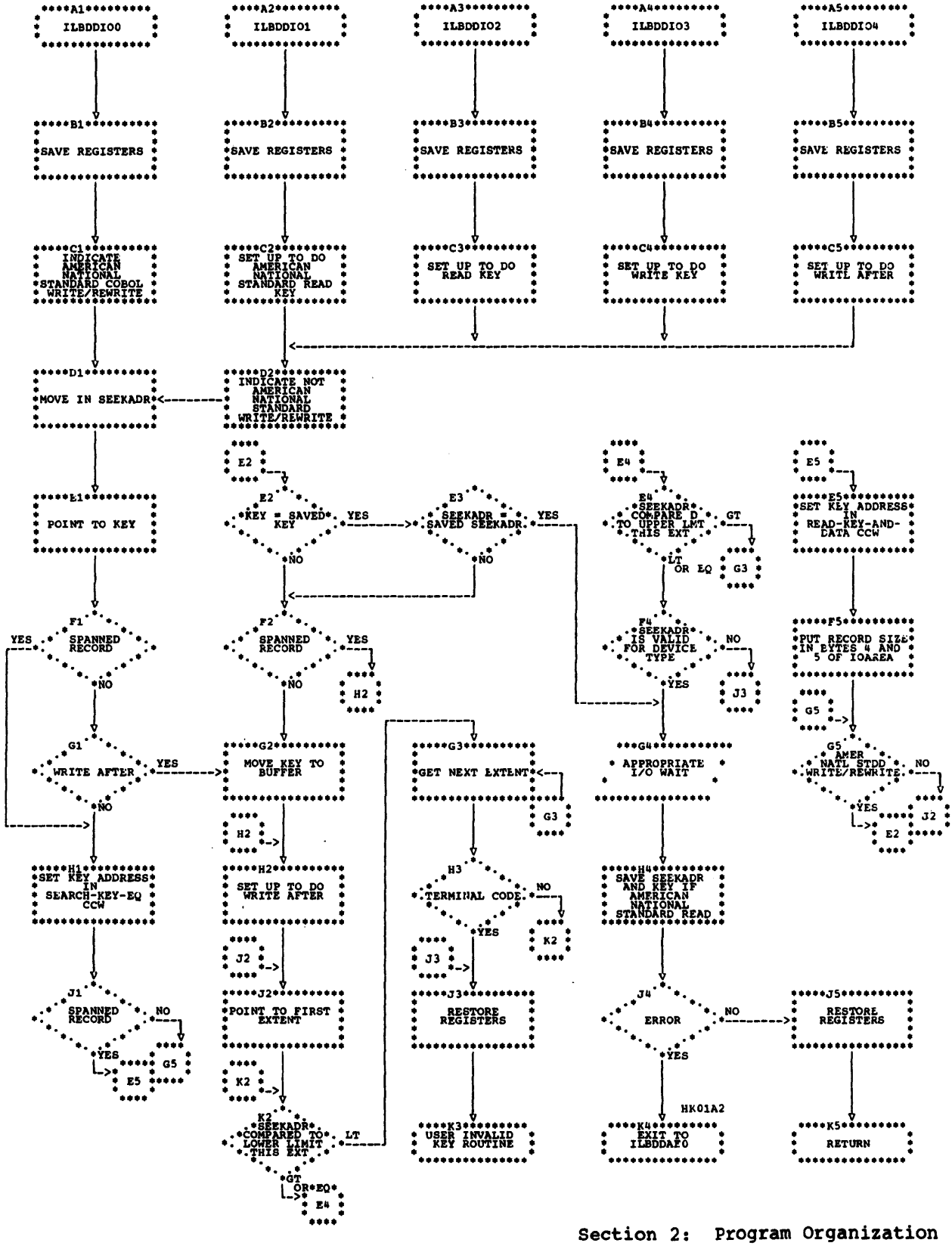


Chart HJ. DA READ and WRITE for Relative Track (ILBDRDI0)

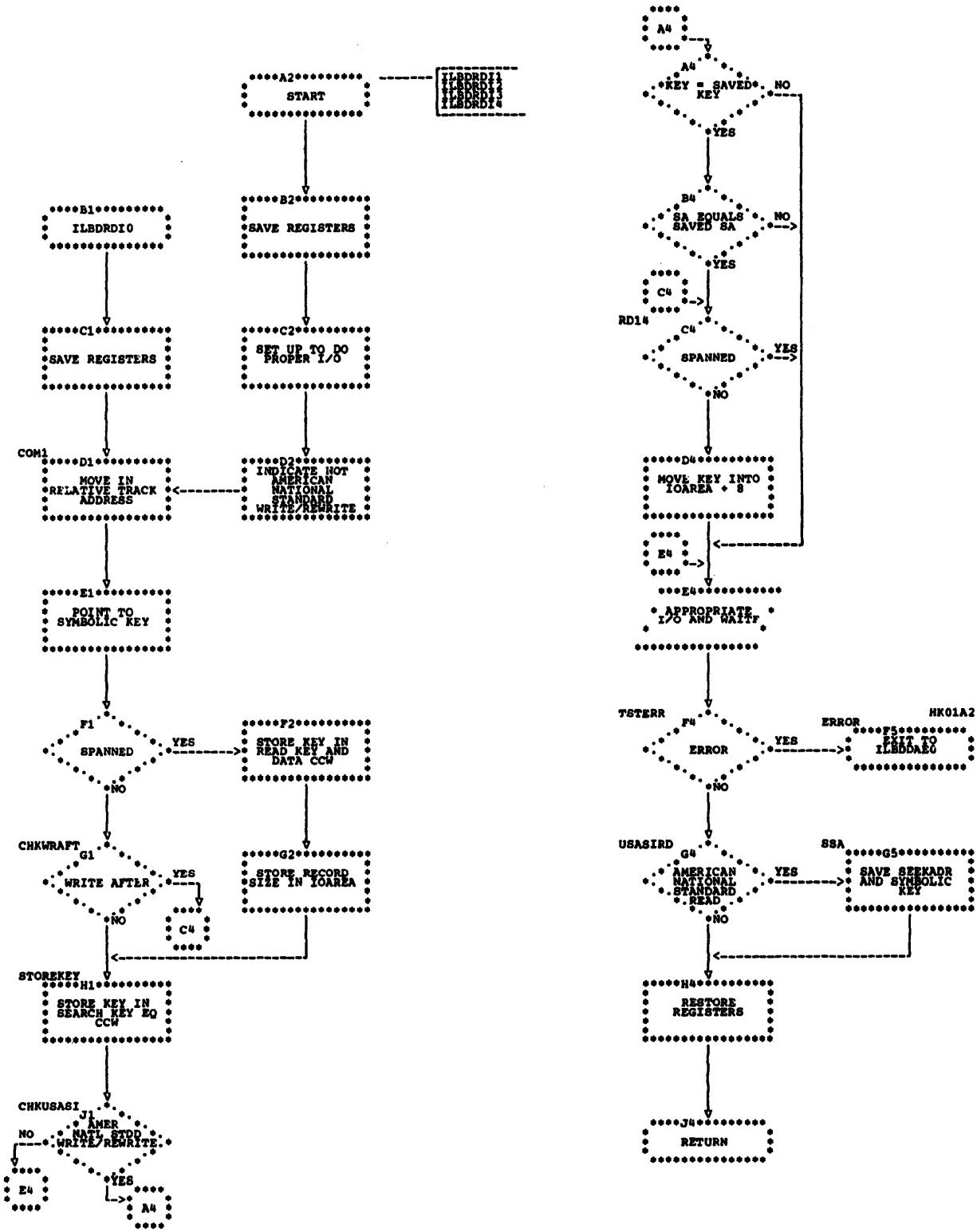


Chart HK. DA Error (ILBDDAE0)

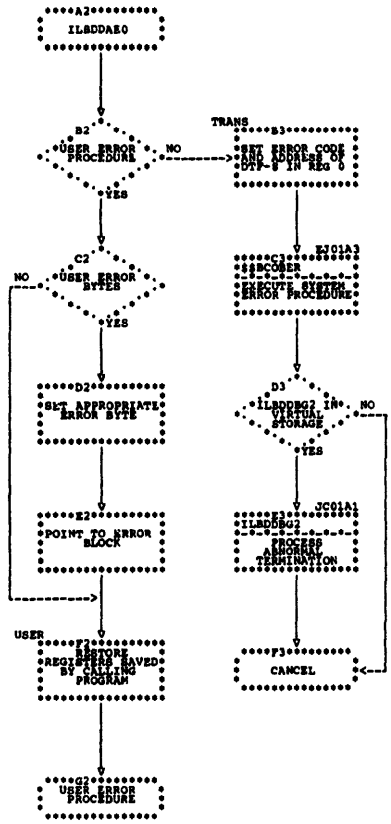


Chart HL. VSAM Initialization (ILBDINT0)

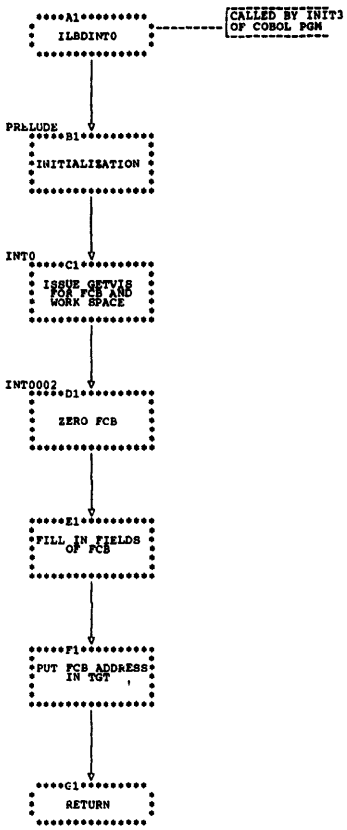


Chart JW. SYMSTATE (ILBDMP25)

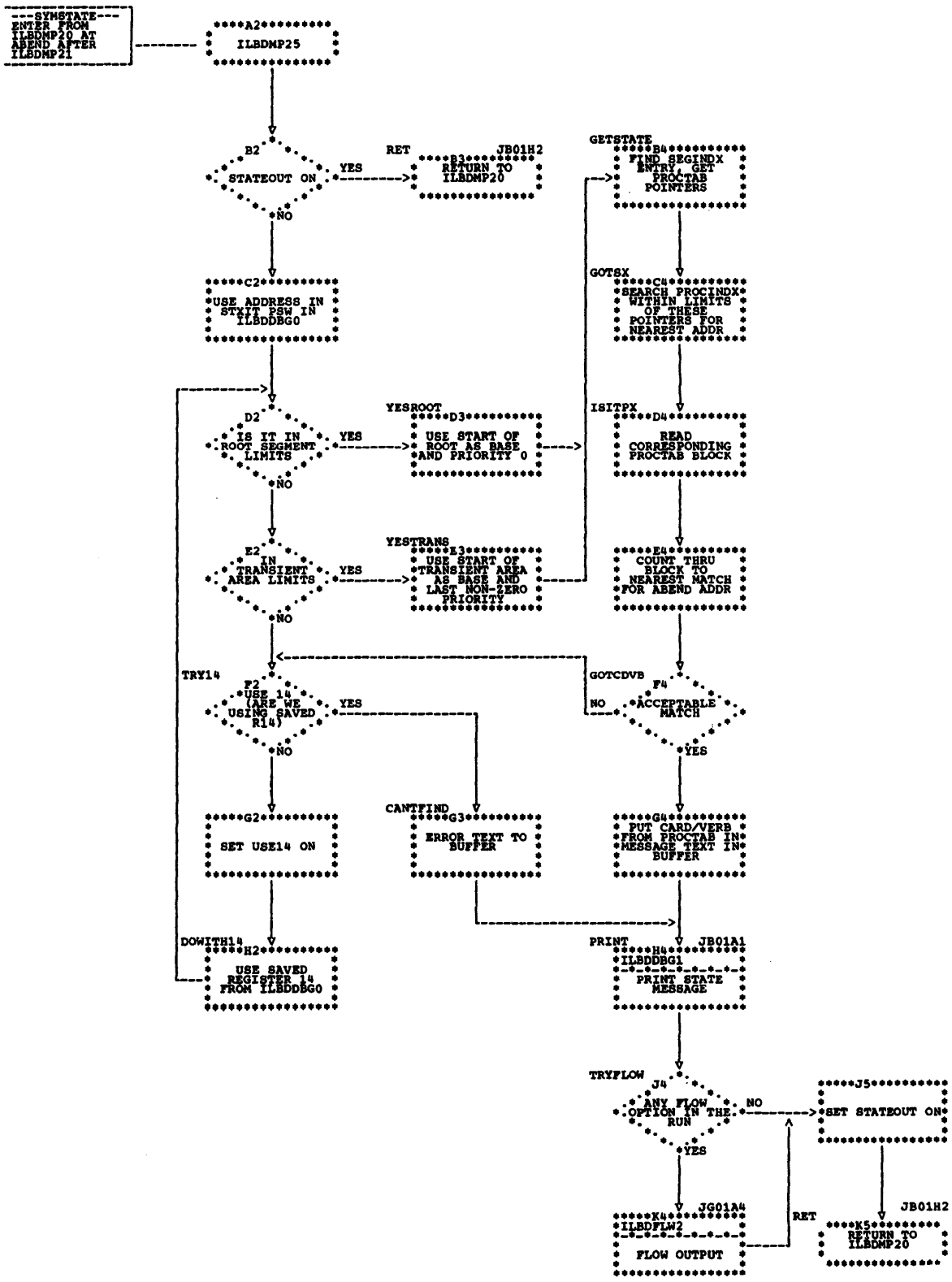


Chart JX. SRCHPUBS (ILBDMP04)

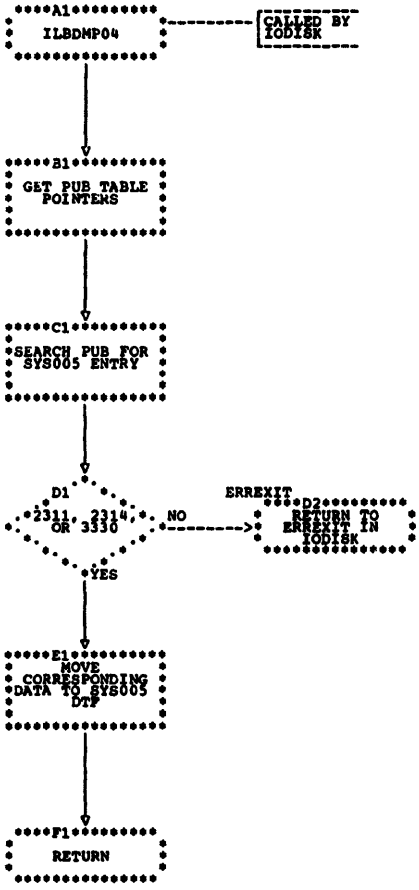


Chart JY. USE-FOR-DEBUGGING Declaratives Subroutine (ILBDBUG0) (Part 1 of 2)

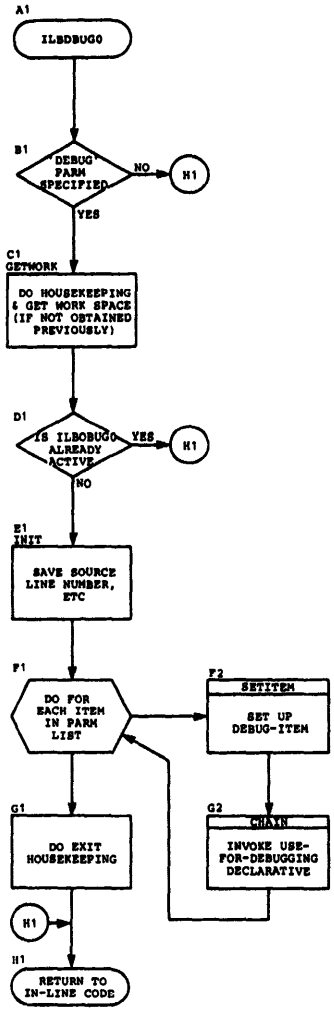


Chart JY. USE-FOR-DEBUGGING Declaratives Subroutine (ILBDBUG0)
(Part 2 of 2)

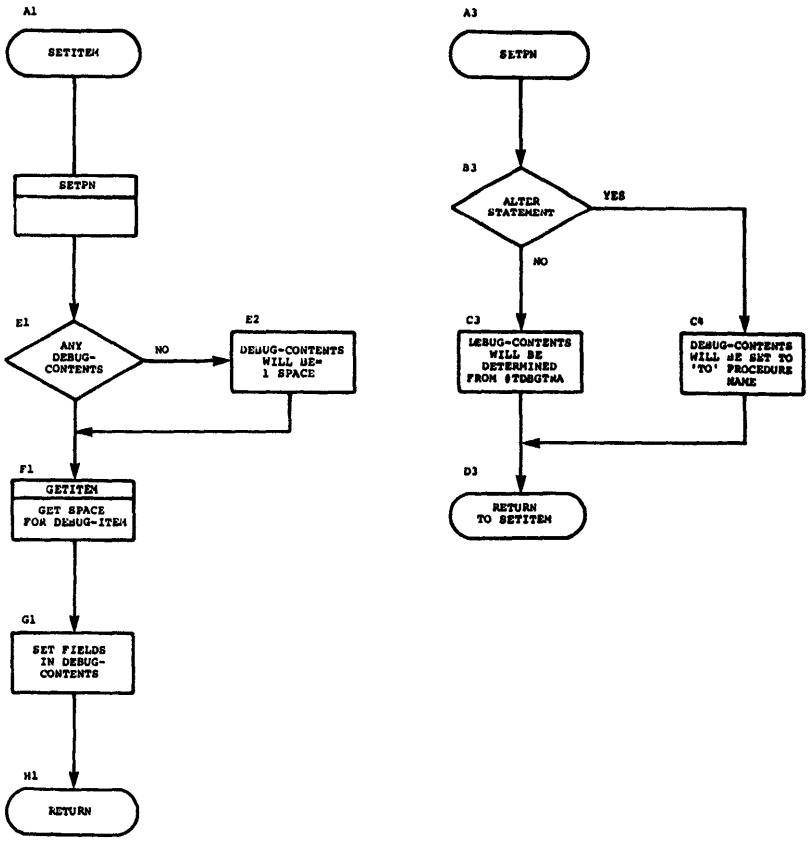


Chart KA. COUNT Initialization Subroutine (ILBDTC00)

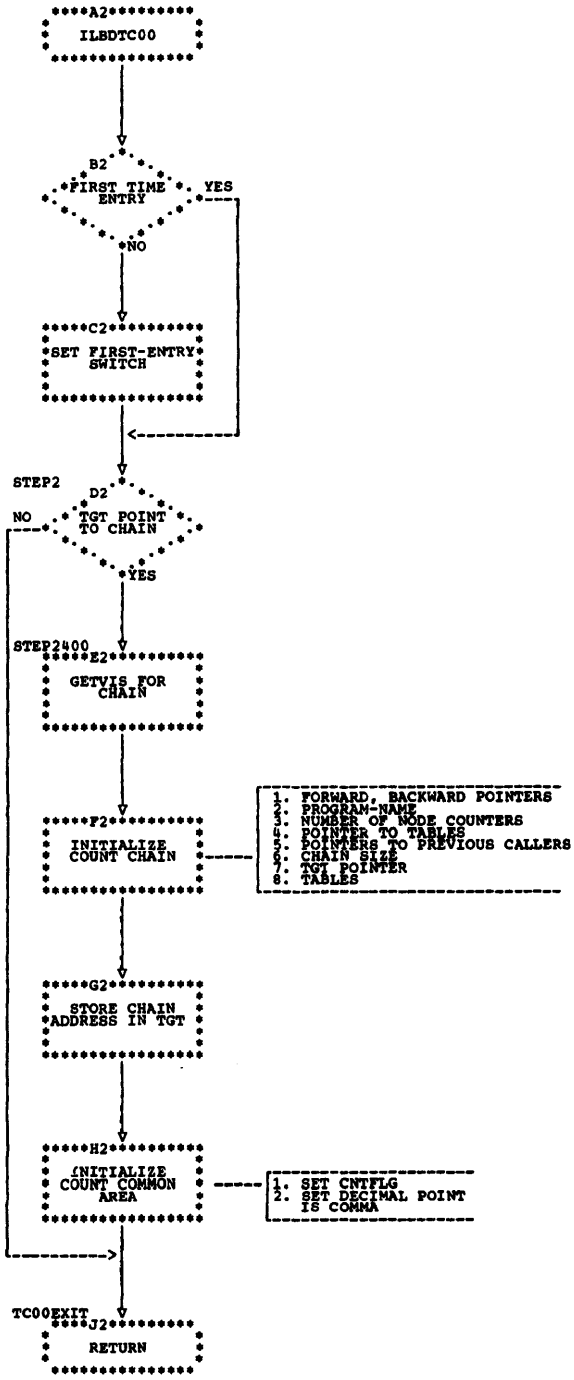


Chart KB. COUNT Frequency Subroutine (ILBDTC10)

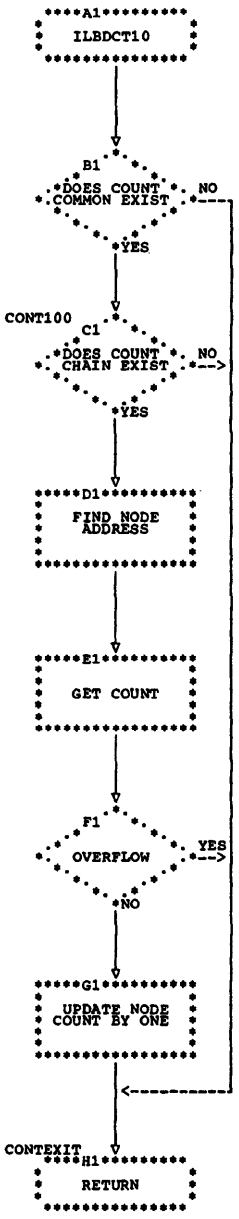


Chart KC. COUNT Termination Subroutine (ILBDTC20)

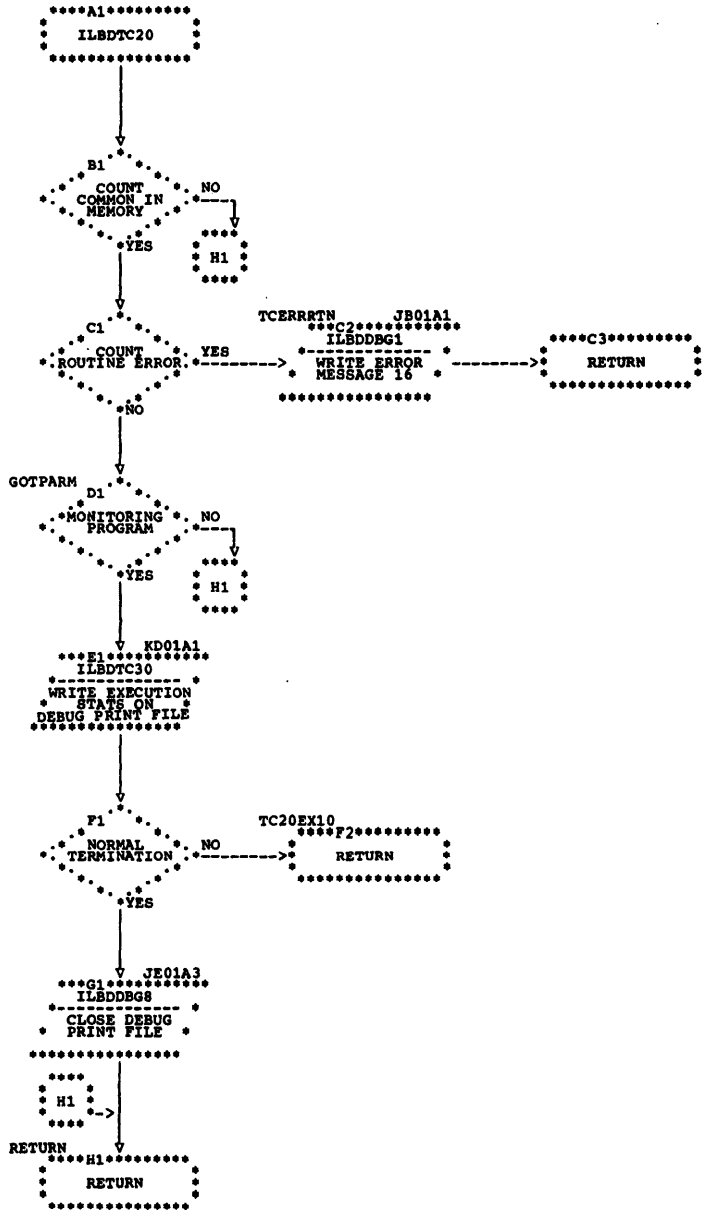
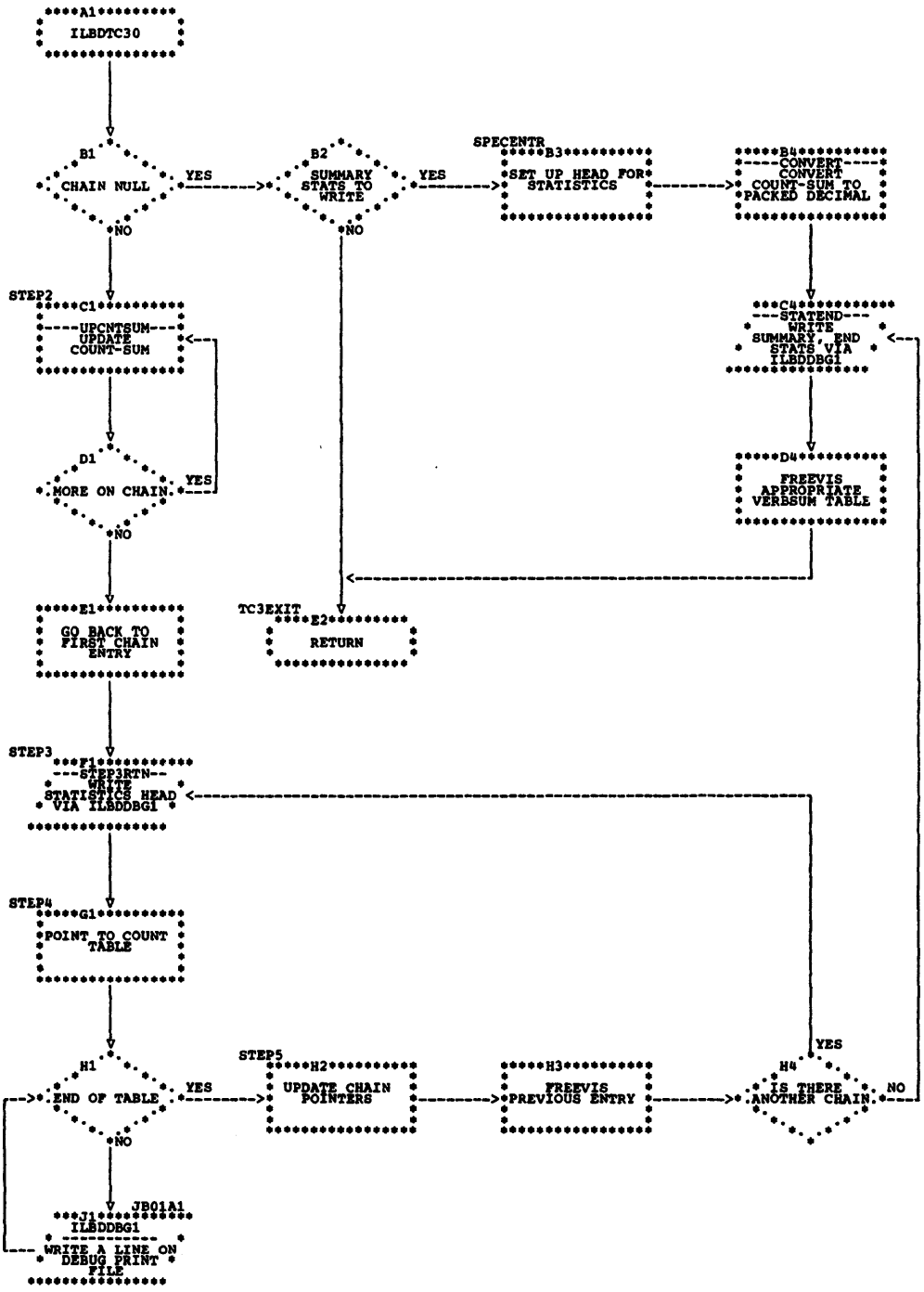


Chart KD. COUNT Print Subroutine (ILBDTC30)



DEBUG COMMON AREA (ILBDDBG7)

The debug common area resides in subroutine ILBDDBG0 with the DSECT name of DBG0COM. It is used by both the debugging subroutines and the object-time execution statistics subroutines. Its format is as follows:

Displacement Hex	Displacement Decimal	Field	No. of Bytes	Description
0	0	SAVEDBG0	72	ILBDDBG0 save area
48	72	SAVEDBG1	72	ILBDDBG1 save area
90	144	ADBG1	4	Address of ILBDDBG1
94	148	CURRBUFF	4	Subfield: ERRPARM, Bytes: 0, Contents: Error parameter Bytes: 1-3, Contents: Address of current buffer
98	152	STXITSA	8	STXIT save area, containing PSW at the time of program check (alternate name STXITPSW) Subfield: PSWL, Bytes: 0-3, Contents: Leftmost PSW bytes Subfield: PSWR, Bytes: 4-7, Contents: Rightmost bytes (bytes 5-7 named INITAD)
A0	160	STXITR0	52	Registers 0-12
D4	212	STXITR13	4	Register 13
D8	216	STXITR14	8	Registers 14 and 15
E0	224	FIRST	4	Address of highest TGT
E4	228	LAST	4	Address of current TGT
E8	232	SAVER14	4	Save register 14 for ABEND outside of COBOL program
EC	236	FLTEP	4	Virtual for floating-point subroutine used by SYMDMP
F0	240	STATEEP	4	Virtual for STATE entry point
F4	244	FLOW2EP	4	Virtual for print FLOW subroutine
F8	248	SORTSEP	4	Virtual for Sort subroutine
FC	252	DSPLEP	4	Virtual for display subroutine
100	256	SYMSIZE	2	SYMSIZE
102	258	FLAG1 (SWITCHA)	1	Switches

Equate Name	Code	Meaning
FRST	X'80'	First time through
DBG0	X'40'	ILBDDBG0 error
REC	X'20'	Debug ABENDed: recursion bit
DBG1	X'10'	First time in ILBDDBG1
DBG2	X'08'	In ILBDDBG2
COBSW	X'04'	Current program is not COBOL
SYMDEAD	X'02'	SYMDMP is dead switch
ENDFLOW	X'01'	Print FLOW when FIRST=LAST

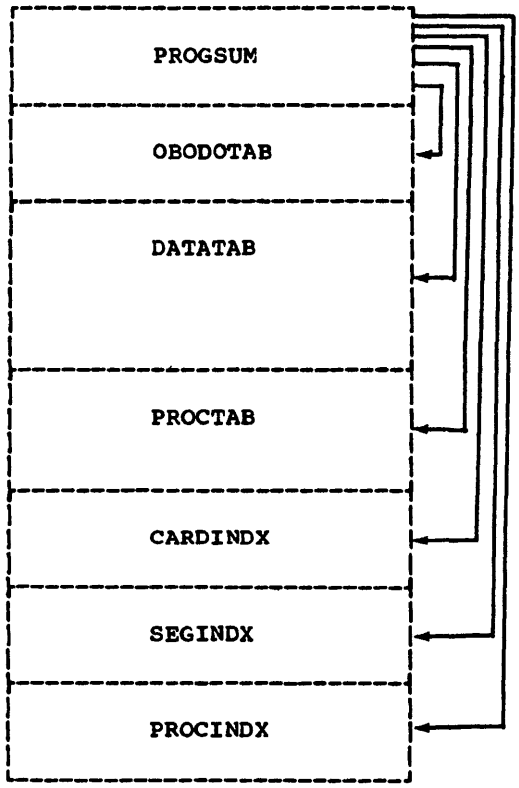
Displacement Hex	Displacement Decimal	Field	No. of Bytes	Description
103	259	FLAG2 (SWITCHC)	1	Equate Name / Code / Meaning STATEMSG X'80' Statement number already printed STATE1 X'40' Use register 14 for STATE STATE2 X'20' Found first segment SYMIN X'10' SYMDMP in core bit FLOWINIT X'08' FLOW already initialized bit DYNAM X'04' Dynamic dump mask FLW1RETN X'02' FLOW1 return bit RANGE X'01' ABEND occurred outside range of SYMDMP (for ILBDDBG2)
104	260	DBG1CODE	1	Print code for ILBDDBG1
105	261	ABCODES	1	ABEND codes
106	262	HEAD1	52	Page eject
13A	314		69	COBOL diagnostic aids
17F	483	HEAD2	121	Triple space
1F8	604	FOOTING	28	End of COBOL diagnostic aids
214	632	STATERR	1	STATE error byte
215	633	CURRPTY	1	Current priority for STATE

DEBUG INPUT FILE

The debug file is made up of fixed-length 512-byte blocks; a 1-byte field containing the hexadecimal value 'FF' marks the end of usable information within a block.

The seven tables described in the following pages exist in the debug file at object time. They are accessed by the subroutines of the SYMDMP program.

See Diagrams 8, 9, and 10 in "Section 2: Program Organization" for the relations among these tables and the object-time subroutines.



PROGSUM TABLE

The PROGSUM table is the first table on the debug file. It consists of a single fixed-length 108-byte entry and contains information about the program and the debug file itself.

Displ Dec	Displ Hex	Field Name	Bytes	Field Description
0	0	PGPROGID	8	PROGRAM-ID
8	8	PGDECLEN	4	Length of Declaratives Section
12	C	PGBL1	4	BL1 address relative to the start of the TGT
16	10	PGBLL1	4	BLL1 address relative to the start of the TGT
20	14	PGSBL1	4	SBL1 address relative to the start of the TGT
24	18	PGDTF1	4	DTF1 address relative to the start of the TGT
28	1C	PGVLC1	4	VLC1 address relative to the start of the TGT
32	20	PGINDEX1	4	INDEX1 address relative to the start of the TGT
36	24	PGENDDTF	4	End of the DTFs relative to the start of the TGT
40	28	PGENDNDX	4	End of the indexes relative to the start of the TGT
44	2C	PGDTDVAD	4	Device address of first block of DATATAB
48	30	PGDTNUM	2	Number of blocks in DATATAB
50	32	PGDTDSP	2	Displacement in the block of the first DATATAB entry
52	34	PGPTDVAD	4	Device address of PROCTAB
56	38	PGCXDVAD	4	Device address of CARDINDX
60	3C	PGSXDVAD	4	Device address of SEGINDX
64	40	PGPXDVAD	4	Device address of PROCINDX
68	44	PGCXNUM	2	Number of entries in CARDINDX
70	46	PGSXNUM	2	Number of entries in SEGINDX
72	48	PGPXNUM	2	Number of entries in PROCINDX
74	4A	PGSXDSP	2	Displacement in the block of the first SEGINDX entry
76	4C	PGPXDSP	2	Displacement in the block of the first PROCINDX entry
78	4E	PGFPDSP	2	Displacement of floating-point virtual from the start of the PGT
80	50	PGODONUM	2	Number of bytes in OBODOTAB, including the unused bytes at the end of the blocks
82	52	PGHASH	2	Identifier to insure match between this debug file and compiled COBOL program
84	54	PGFIB	4	Address of first FIB relative to start of TGT
88	58	PGLEN	1	Length of PROGSUM
89	59	PGFILL	19	Reserved for later use

Note: The only fields that may be zero in this table are PGDECLEN, PGODONUM, and PGFPDSP when the referenced areas are absent from the program. For TGT addresses which do not exist, the address of the first byte following the previous cell is used because these cells are used in calculating the number of TGT cells of a given kind to dump.

OBODOTAB TABLE

The OBODOTAB table is an abstract of the DATATAB entries for all objects of OCCURS...DEPENDING ON clauses in the program. The OBODOTAB table, if present, immediately follows the PROGSUM table and contains one variable-length entry for each unique object of an OCCURS...DEPENDING ON clause. Each entry begins on a fullword boundary within the block.

The entries are essentially the same as the DATATAB entries for the same name. See the entries for elementary numeric items in the format of the DATATAB table. OBODOTAB entries differ only in that the card-number field is zero and the renaming information is omitted. Table-locators within the DATATAB entries are used to access the OBODOTAB entries. See the subscripting information portion in the format of the DATATAB table.

COUNT-NAME-TYPE FIELD

Displ Dec	Displ Hex	Field Name	Bytes	Field Description
0	0		1	Count Field: Number of bytes (c) in name field
1	1		c	Name Field: Number of bytes, varies between 1 and 30
1+c	1+c		1	Count Field: Number of bytes in remainder of this entry
2+c	2+c	CARDNUM	3	Card number where name is defined
5+c	5+c	MAJMIN	1	Type of Entry (For description of this field see corresponding field in DATATAB table)

VARIABLE ATTRIBUTES FIELD

For description of this field see corresponding field in DATATAB table.

DATATAB TABLE

The DATATAB table is the third table in the debug file. It immediately follows the last entry of the ORODOTAB table, if that table is present. Otherwise, it follows the PROGSUM table.

The table consists of two fields, the Count-Name-Type field (shown below) and the Variable Attributes field. The Count-Name-Type field has the same format for all entries. It varies in length between 7 and 36 bytes. The Variable Attributes field differs for each type of entry and is described in the diagrams on the following pages.

COUNT-NAME-TYPE FIELD

Displ Dec	Displ Hex	Field Name	Bytes	Field Description
0	0		1	Count field: Number of bytes (c) in name field
1	1		c	Name field: Number of bytes varies between 1 and 30
1+c	1+c		1	Count field: Number of bytes in remainder of entry
2+c	2+c	CARDNUM	3	Card number where name is defined (contains zeros for RENAMES items)
5+c	5+c	MAJMIN	1	Type of entry

Bits	Bit Settings	Meaning
0-3	1000XXXX	FD entry
	1001XXXX	SD entry
	1110XXXX	RD entry
	1111XXXX	Index-name
	0000XXXX	Level description under FD
	0001XXXX	Level description under SD
	0110XXXX	Level description under RD
	0100XXXX	Level description in Working-Storage
	0101XXXX	Level description in Linkage
4-7	XXXX0001	Fixed length group
	XXXX0010	Alphabetic
	XXXX0011	Alphanumeric
	XXXX0100	Variable length group
	XXXX0101	Numeric edited
	XXXX0110	Sterling report
	XXXX0111	Usage index
	XXXX1000	External decimal
	XXXX1001	External floating point
	XXXX1010	Internal floating point
	XXXX1011	Binary
	XXXX1100	Internal decimal
	XXXX1101	Sterling non-report
	XXXX1110	Alphanumeric edited
	XXXX1111	RENAMES (level 66)

DATATAB TABLE: VARIABLE ATTRIBUTES FIELD

SD item: There are no variable attributes for an SD entry.

RENAMES item (level 66):

6+c	6+c	RENAMES	1	Bit		
				Bit	Settings	Meaning
				7	XXXXXXX1	Next DATATAB entry RENAMES the same item as this one does
					XXXXXXX0	This is the last (or only) item renaming an item

INDEX name:

6+c	6+c	INDXCELL	2	Index cell number in TGT

FD item:

6+c	6+c	DTFNUM	1	DTF number
7+c	7+c	ACCESFLG	1	Access method

Bit	Settings	Meaning
0-3	0001XXXX	DTFCD
	0010XXXX	DTFPR
	0011XXXX	DTFMT
	0100XXXX	DTFSD
	0101XXXX	DTFDA
	0110XXXX	DTFIS
7	XXXXXXX1	Sequential access method
	XXXXXXX0	Random access method

RD item:

6+c	6+c	LINECTR	3	Addressing parameters of line counter

Bit	Settings	Meaning
0-3	0000XXXX	BL entry
	0001XXXX	BLL entry
	0100XXXX	SBL entry
4-15		Displacement from BL
16-23		BL Number

9+c	9+c	PAGECTR	3	Addressing parameters of page counter (same form as addressing parameters above)

Level Description Item:

Variable attributes for level description items are divided into two portions: (1) the type-dependent portion, (2) subscripting information portion. The subscripting information portion is the same for all level description item entries. It follows and is described after the type dependent portion descriptions

(1) Type Dependent Portion of Level Description Item:

FIXED LENGTH GROUP:

6+C	6+C	IDKFLD	3	Addressing parameters (same form as above)
9+c	9+c	LVLRDEFN	3	

Bit	Settings	Meaning
0-5		Normalized level number
6	xxxxxx1x	REDEFINES
7-23		Object time virtual storage size (in bytes)

DATATAB TABLE: VARIABLE ATTRIBUTES FIELD (Continued)

VARIABLE LENGTH GROUP:

6+c	6+c		3	Addressing parameters (same form as above)		
9+c	9+c	MAXSIZE	3	Bit		
				Bit	Settings	Meaning
				0-5		Normalized level number
				6	xxxxxx1x	REDEFINES
				7-23		Maximum object time virtual storage size (in bytes)
12+c	C+c	VLCNUM	2	Bit		
				Bit	Settings	Meaning
				0	1XXX	ODO Master
				1-3		Unused
				4-15		VLC number

ELEMENTARY, ALPHABETIC, ALPHANUMERIC, REPORT, EDITED, STERLING, EXTERNAL FLOATING POINT:

6+c	6+c		3	Addressing parameters (same form as above)		
9+c	9+c	JUSTRGT	3	Bit		
				Bit	Settings	Meaning
				0-5		Normalized level number
				6	xxxxxx1x	REDEFINES
				7	xxxxxxx1	JUSTIFIED RIGHT
				8-23		Object time virtual storage size (in bytes)

INTERNAL FLOATING POINT:

6+C	6+C		3	Addressing parameters (same form as above)		
9+c	9+c	FLPTYPE	1	Bit		
				Bit	Settings	Meaning
				0-5		Normalized level number
				6	xxxxxx1x	REDEFINES
				7	XXXXXXX0	COMP-1
					XXXXXXX1	COMP-2
10+c	A+c		2	Unused		

BINARY, INDEC, INTERNAL DECIMAL, EXTERNAL DECIMAL:

6+c	6+c		3	Addressing parameters (same form as above)		
9+c	9+c	NUMINFO1	1	Bit		
				Bit	Settings	Meaning
				0-5		Normalized level number
				6	xxxxxx1x	REDEFINE
				7	xxxxxxx1	S in PICTURE
10+c	A+c		2	0	1XXXXXXX	Leading sign
					0XXXXXXX	Trailing sign
				1	X1XXXXXX	Separate sign
					X0XXXXXX	Overpunch
				2	XX1XXXXX	Significant digits left of decimal point
					XX0XXXXX	No significant digits left of decimal point
				3	XXX1XXXX	Significant digits right of decimal point
					XXX0XXXX	No significant digits right of decimal point
				4-8		If bit 2 equals 1, number of digits to left of decimal point. If bit 2 equals 0, number of digits to right of decimal point.
				9-13		If bits 2 and 3 both equal 1, number of digits to right of decimal point. If only bit 2 or bit 3 equals 1, number of Ps in PICTURE
				14-15		Unused

DATATAB TABLE: VARIABLE ATTRIBUTES FIELD (Continued)

(2) Subscripting Information Portion of Level Description Item:

This portion of the Variable Attributes section begins immediately after the type-dependent portion.

It ranges in size from 1 byte unsubscripted item to a maximum of 20 bytes for an item belonging to 3 variable-length gruups.

1 Guide to RENAMES and subscripting

Bit	Bit Settings	Meaning
0	1XXXXXXX	This item is renamed. The next DATATAB entry renames it.
1	X1XXXXXX	This item contains an ODO clause.
2	XX1XXXXX	Item requires at least 1 subscript.
3	XXX1XXXX	OCCURS clause connected with the most inclusive or only group; or elementary item contains an ODO.
4	XXXX1XXX	Item requires at least two subscripts
5	XXXXX1XX	OCCURS clause connected with the less inclusive group of 2 or the middle inclusive group of 3 or elementary group contains an ODO.
6	XXXXXX1X	Item requires 3 subscripts
7	XXXXXXX1	OCCURS clause connected with the least inclusive group of three or elementary item contains an ODO.

1 VLC information

Bit	Bit Settings	Meaning
0	1XXXXXXX	Most inclusive group of 3 or only group
1	X1XXXXXX	Less inclusive group of 2 or middle inclusive group of 3
2	XX1XXXXX	Least inclusive group of 3

If any of these bits equals 1, bytes 2 and 3 of the group length information for the associated group contain a VLC number rather than the length of the group.

DATATAB TABLE: VARIABLE ATTRIBUTES FIELD (Continued)

1st subscript (if present)	2	Number of occurrences (Maximum number if ODO) specified in OCCURS clause governing this item.
	2	Displacement of next occurrence governed by OCCURS clause (See Note)
2nd subscript (if present)	2	Number of occurrences (as above)
	2	Displacement of next occurrence governed by OCCURS
3rd subscript (if present)	2	Number of occurrences (as above)
	2	Displacement of next occurrence governed by OCCURS
1st subscript with ODO (if present)	2	OBODOTAB pointer for most inclusive group or elementary item containing an ODO
2nd subscript with ODO (if present)	2	OBODOTAB pointer for less inclusive group (as above)
3rd subscript with ODO (if present)	2	OBODOTAB pointer for least inclusive group (as above)

Bits	Contents
0-8	Relative block number in OBODOTAB
9-15	Displacement within block (in fullwords)

Note: If the applicable OCCURS clause is on an elementary item, the displacement is the machine length of that item; if the applicable OCCURS clause is on a fixed-length group, the displacement is the length of the group as stored in the group's DATATAB entry; if the applicable OCCURS clause is on a variable-length group, the displacement field contains the VLC number for the group.

PROCTAB TABLE

The PROCTAB table contains one 5-byte entry for each card and/or verb in the source listing of the COBOL Procedure Division. The table is ordered on three levels:

1. Priority (in ascending order of independent segments, with the root segment last)
2. Card-number within priority
3. Verb-number within card

The last PROCTAB entry for a priority has a card and/or verb number of zero. In addition, the relative address field contains the address of the first byte following all instructions for the segment with that priority.

For the relationships among this table and the PROCINDX, SEGINDX, and CARDINDX tables, see Diagrams 8, 9, and 10 in "Section 2: Program Organization."

Displ Dec	Displ Hex	Field Name	Bytes	Field Description
0	0	PTCDVB	3	Card-number and verb-number on source listing Bit — Contents 0-19 — Card-number 20-23 — Verb-number
3	3	PTRELAD	2	Relative address of instructions for this entry within program fragment to which it belongs

CARDINDX TABLE

The CARDINDX table is a directory to the SEGINDX table and contains one 5-byte entry for each program fragment and one entry for each discontinuity in the COBOL instructions within a segment. Entries in the CARDINDX table are in ascending card-number order and are accessed by indexing through the table sequentially.

The CARDINDX table starts at the beginning of a block.

For the relationships among this table and the PROCTAB, PROCINDX, and SEGINDX tables, see Diagrams 8, 9, and 10 in "Section 2: Program Organization."

Displ Dec	Displ Hex	Field Name	Bytes	Field Description
0	0	CXCDVB	3	Card-number and verb-number of first card represented by this entry Bits — Contents 0-19 — Card-number 20-23 — Verb-number
3	3	CXPRIOR	1	Priority number associated with this card
4	4	CXFRAG	1	Relative fragment number within the priority to which this card belongs

SEGINDX TABLE

The SEGINDX table contains one 10-byte entry for each program fragment. The table is ordered on two levels:

1. Ascending priority number
2. Ascending fragment number within a priority

For the relationships among this table and the PROCTAB, PROCINDX, and CARDINDX tables, see Diagrams 8, 9, and 10 in "Section 2: Program Organization."

Displ Dec	Displ Hex	Field Name	Bytes	Field Description
0	0	SXPRIOR	1	Priority number
1	1	SXRELAD	3	Address of this fragment relative to the beginning of the segment
4	4	SXPTLOC1	3	Table locator for PROCTAB entry of first card number and/or verb-number in this fragment Bits 0-14: Relative Block number in PROCTAB Bits 15-23: Displacement within block
7	7	SXPTLOC2	3	Table locator for PROCTAB entry of last card and/or verb in this fragment

PROCINDX TABLE

The PROCINDX table is a summary index of the PROCTAB table and contains one 10-byte entry for each block of PROCTAB entries. PROCINDX entries are ordered by relative block number in the PROCTAB table and are accessed by searching sequentially after indexing to a starting point determined by the block number from the CARDINDX or SEGINDX table.

For the relationships among this table and the PROCTAB, SEGINDX, and CARDINDX tables, see Diagrams 8, 9, and 10 in "Section 2: Program Organization."

Displ Dec	Displ Hex	Field Name	Bytes	Field Description
0	0	PXCDVB	3	Card-number and verb-numner of first entry in block of PROCTAB table. Bits 0-19: Card-number Bits 20-23: Verb-number
3	3	PXRELAD	3	Relative address of instructions for this entry within segment to which it belongs
6	6	PXDEVADR	4	Device address of PROCTAB table block related to this entry.

EXECUTION-TIME TABLES FOR DEBUG OPERATIONS

The following four tables are built in virtual storage by the SYMDMP subroutines from information in the Debug File and the control cards for a program compiled with the SYMDMP option. They are used for producing the dump to meet dynamic dump request and at abnormal termination.

DATADIR TABLE

The DATADIR table is a directory to the DATATAB table and only exists when a DYNAMTAB table exists. There is one fixed-length 8-byte entry for each distinct DATATAB block which contains an identifier specified on a line-control card. Entries are in the order in which requests appeared on line-control cards.

Displ Dec	Displ Hex	Field Name	Bytes	Field Description
0	0	DDDEVADR	4	Device address of DATATAB block in debug file
4	4	DDSW	1	Switch - If bit 0 is equal to 1, the block is not in virtual storage. If bit 0 is equal to 0, the block is in virtual storage.
5	5	DDCORE	3	Address of DATATAB block in virtual storage

Note: This table is limited by the 7-bit indexes in the DYLOCNM field of the DYNAMTAB table to a maximum of 128 entries. If the maximum is exceeded, a message is produced and further dynamic dumping requests are ignored.

DYNAMTAB TABLE

The DYNAMTAB table summarizes dynamic dump requests and contains one entry for each line-control card. The table entries are composed of a fixed and variable portion. Entries are variable in length with a minimum length of 17 bytes. DYNAMTAB entries are chained together, and each entry begins with the address of the next entry. The end of a group of entries for one program is marked by the DYLASTDY switch. The DYNAMTAB table is searched sequentially; the search ends at the entry in which the DYLASTDY switch is on.

Fixed Portion

Displ Dec	Displ Hex	Field Name	Bytes	Field Description
0	0	DYNXTDY	3	Address of next DYNAMTAB entry
3	3	DYSW	1	Switch Name / Bit / Contents DYALL / 0 / If 1, ALL specified DYHEXALL / 1 / If 1, HEX with ALL specified DYON / 2 / If 1, ON specified DYLASTDY / 3 / If 1, Last DYNAMTAB entry DYSKPDMP / 4 / If 1, No dump - ON value is wrong
4	4	DYCDVB	3	Card-number and/or verb-number Bits / Contents 0-19 / Card-number 20-23 / Verb-number
7	7	DYPRIOR	1	Priority of this card
8	8	DYCOBINS	6	Machine instruction corresponding to card and verb
14	E	DYINSADR	3	Address of this instruction in virtual storage
17	11	DYONS	8	Only present if ON specified Name / Bytes / Contents DYON1 / 2 / Start value DYON2 / 2 / Increment value DYON3 / 2 / End value DYONCUR / 2 / Current value

Variable Portion

For each request for a single identifier:	DYIDSW	1	Switch Name / Bit / Contents DYHEXID / 0 / If 1, HEX specified for this request DYTHRUID / 1 / If 1, THRU specified (entry is 5 bytes long) DYERRID / 2 / If 1, error in this request; ignore it
	DYLOCNM	2	Table-locator for this identifier, consisting of: Bits / Contents 0-6 / Entry number in DATADIR to find device address of DATATAB entry for this identifier 7-15 / Displacement in DATATAB block of entry for this identifier
For each request for identifier THRU identifier:		2	If THRU is specified, table-locator (same format as above) for identifier which is the object of THRU

Note: A dummy table-locator of hex '0001' is used to represent TALLY; a dummy table-locator of hex '0002' is used to represent SORT-RETURN.

PCONTROL TABLE

The PCONTROL table contains information about each program requesting the symbolic dump option within a run unit and consists of one fixed-length 76-byte entry for each program-control card. Entries begin on a fullword boundary and are chained together. Each entry is followed by the DYNAMTAB table, DATADIR table, and, if necessary for dynamic dumping, the OBODOTAB table for the program.

Displ Dec	Displ Hex	Field Name	Bytes	Field Description
0	0	PCPROGID	8	PROGRAM-ID
8	8	PCFILNAM	7	File name of debug file (the default is IJSYS05)
15	F	PCSYSNNN	1	nnn of SYSnnn for the debug file in binary
16	10	PCBL1	4	BL1 address
20	14	PCBLL1	4	BBL1 address
24	18	PCSBL1	4	SBL1 address
28	1C	PCDTF1	4	DTF1 address
32	20	PVCLC1	4	VLC1 address
36	24	PCINDEX1	4	INDEX1 address
40	28	PCANXTPC	4	Address of the next PCONTROL entry (if this is the last entry, this field contains zeros)
44	2C	PCADYTAB	4	Address of DYNAMTAB for this program (if there is no DYNAMTAB, this field contains zeros)
48	30	PCAOBODO	4	Address of OBODOTAB in virtual storage (If OBODOTAB is not in virtual storage, this field contains zeros)
52	34	PCACOB	4	Address of start of overlayable virtual storage in root segment
56	38	PCATRANS	4	Address of start of Transient Area
60	3C	PCADATDR	4	Address of DATADIR, if present (if there is no DATADIR, this field contains zeros)
64	40	PCDMPLNG	2	Length of overlayable virtual storage in root segment
66	42	PCTRLNG	2	Length of Transient Area
68	44	PCDDNUM	2	Number of DATADIR entries
70	46	PCPRIOR	1	Last non-root segment entered (if any)
71	47	PCSW	1	Switch
				Name / Bit / Contents
				PCHEX 0 If 1, HEX specified
				PCENTRY 1 If 1, ENTRY specified
				PCPDUMP 2 If 1, PDUMP specified
				PCMT 3 If 1, MT specified
				PCDYNAM 4 If 1, No DYNAMTAB exists
				PCRELOC 5 If 1, Address (BL1 through INDEX1) have been relocated
				PCIOERR 6 If 1, I/O error found on debug file
72	48	PCFIB1	4	Address of first FIB cell

QUALNAMS TABLE

The QUALNAMS table is an area overlaying SCANP (ILBDMP11), in which identifiers are entered in a manner to permit a batched sequential search through the DATATAB table for the names requested on line-control cards. The QUALNAMS table contains one entry for each identifier named on a line-control card. Each entry is composed of a fixed and variable portion. Entries are in the order in which identifiers and names (qualifiers) appeared on line-control cards.

Fixed Portion

Displ Dec	Displ Hex	Field Name	Bytes	Field Description
0	0	QCODE	2	Switch and displacement

Name	Bits	Contents
QID	0	If 1, beginning of an entry for an identifier
QFOUND	1	If 0, identifier is not resolved If 1, identifier has been found in DATATAB
QTHRU	2	If 1, request for this identifier followed by THRU
	3-6	Unused
QDISP	7-15	Contains zeros until identifier has been found in DATATAB; then it contains the displacement in the DATATAB block containing the entry for the identifier

2	2	QDEVADR	4

Contains zeros until identifier has been found in DATATAB; then it contains the device address of the DATATAB block containing the entry for the identifier

Variable Portion

For each name (qualifier) making up the identifier:	ONMLEN	1	Number of bytes (n) in the following name
	QNAME	n	Name (qualifier)

Name	Bits	Contents
QNMZONE	0-3	These bits are used as a switch to indicate whether the name has been found in DATATAB. When it has not been found, they contain normal zone bits for the letter or number which begins the COBOL name. When the name has been found, they are set to zero to prevent searching for the name again.

Note: There is no special end marker for the QUALNAMS table, but the address of the last byte of the table is entered in the Common Data Area. The search of the QUALNAMS table is sequentially forward through the indentifiers, and sequentially backward within an identifier entry, from the most inclusive to the least inclusive qualifier.

The following two control blocks are required to process input/output requests for VSAM files.

VSAM FILE INFORMATION BLOCK (FIB)

The file information block, a portion of the completed object module, is used at execution time by the ILBDINT0, ILBDVOC0, and ILBDVIO0 COBOL library subroutines for processing input/output verbs used with VSAM files. The FIB is built by phase 21 and completed by the ILBDVOC0 subroutine.

Fixed Portion:

Displacement Hex	Displacement Decimal	Field	No. of Bytes	Description
0	0	IFIBID	1	FIB identification code: X'I'
1	1	IFIBLVL	1	FIB level number
2	2	INAMED	7	External name
9	9	INAMEDB	1	External name
A	10		1	Reserved
B	11	IORG	1	Organization

Code:

Bits	Equate Name	Bit Settings	Meaning
0-7	IORGVPS	1000 1000	VSAM ADDRESSED SEQUENTIAL
	IORGVIX	0100 1000	VSAM INDEXED

C	12	IACCESS	1	ACCESS MODE

Code:

Bits	Equate Name	Bit Settings	Meaning
0-7	IACCSEQ	1000 0000	SEQUENTIAL
	IACCRAN	0100 0000	RANDOM
	IACCDYN	0010 0000	DYNAMIC

D	13	IRCDMODE	1	0-7	IRCDFIX	1000 0000	Fixed length records
E	14	ISW1	1	Miscellaneous switches			

Code:

Bits	Equate Name	Bit Settings	Meaning
0-7	ISOPTNL	1000 0000	OPTIONAL specified
	ISSAMREC	0010 0000	SAME RECORD AREA specified
	ISSAME	0001 0000	SAME RECORD specified

F	15		7	Reserved
16	22	IRECLEN	2	Number of bytes in longest 01-entry
18	24	IRECDBL	2	Displacement in TGT of record's first base locator cell
1A	26	IRECNBL	1	Number of base locators for RECORD AREA
1B	27		1	Reserved
1C	28	ISTATDBL	2	Displacement in TGT of base locator for STATUS data-name
1E	30	ISTATDDN	2	Displacement from base locator of STATUS data-name
20	32	ISTATLDN	2	Length of STATUS data-name
22	34		1	Reserved
23	35	IKEYNO	1	Number of entries in key list
24	36	IKEYFNTL	2	Length of each entry in key list
26	38	IPSWISW	1	Miscellaneous switches

Displacement Hex	Displacement Decimal	Field	No. of Bytes	Description
27	39	IPSWNO	1	Number of entries in password list
28	40	IPSWENTL	2	Length of each entry in password list
2A	42		14	Reserved
38	56	IMISCAD	4	Address in variable length portion of FIB for miscellaneous clauses
3C	60		4	Reserved
40	64	IKEYLSTA	4	Address of first key list entry
44	68	IPSWLSTA	4	Address of first password list entry
48	72		16	Reserved

Variable Length Portion:

Supplementary Information for miscellaneous clauses (one for each clause):

Displacement Hex	Displacement Decimal	Field	No. of Bytes	Description
0	0	IMSW1	2	Switch bytes

Code:

Bits	Equate Name	Bit Settings	Meaning
0-7	IMRREOV	1000 0000	RERUN at end of volume
8-15			Reserved

2	2	IRERUNI	4	RERUN integer (field contains zeros if RERUN not specified)
6	6		2	Slack bytes
8	8	IRERUNN	8	External-name of RERUN clause

Key List Entry: (one per user-defined key--RECORD/ALTERNATE/RELATIVE)

Displacement Hex	Displacement Decimal	Field	No. of Bytes	Description
0	0	KEYSW	1	Miscellaneous switches

Code:

Bits	Equate Name	Bit Settings	Meaning
0-7	IKEYCOMP	1000 0000	Key is USAGE COMP (binary)

1	1	IKEYLDN	1	Length of key data-name
2	2	IKEYDBL	2	Displacement of key data-name's locator in TGT
4	4	IKEYDDN	2	Data-name displacement from locator

Password List Entry: (one per password)

0	0	IPSWDIXN	1	Associated index number 0 = none 1 = primary
1	1	IPSWDLDN	1	Length of password data-name
2	2	IPSWDDBL	2	Displacement of password data-name's locator in TGT
4	4	IPSWDDDN	2	Data-name displacement from locator

VSAM FILE CONTROL BLOCK

The VSAM File Control Block is created by the ILBDINT0 COBCL library subroutine. It is used by the ILBDVIO0 and ILBDVOC0 subroutines to interface with the VSAM system control subroutines.

Displacement Hex	Displacement Decimal	Field	No. of Bytes	Description
0	0	FCBID	1	FCB identification code: 'F'
1	1	FCBLVL	1	FCB level number
2	2	FOPENOPT	4	Save area for OPEN options
6	6	FCLOSOPT	4	Save area for CLOSE options
A	10		2	Reserved
C	12	FCOBRTN	4	Address of COBOL transmitter routine
10	16	FUSERR	4	Address of USE...ERROR declarative
14	20	FUSELIST	4	Address of USE declarative Exit List
18	24		6	Reserved
1E	30	FRECKEY	1	Number of RECORD KEY
1F	31	FADVANC	1	Reserved
20	32	FENDINV	4	Return address from INVALID KEY, AT END, or end-of-page
24	36		12	Reserved for compilation-dependent fields
30	48	FOPENOPS	4	Options for VSAM OPEN verb

Code:

Bits	Equate Name	Bit Settings	Meaning
0-7	FOPIN	1000 0000	INPUT
	FOPOUT	0100 0000	OUTPUT
	FOPIO	0010 0000	I-C
	FOPEXT	0001 0000	EXTEND
8-15	Reserved		
16-23	FOPUERR	1000 0000	USE...ERROR declarative address in FUSERR cell
24-31	FTSORT	1000 0000	Called from ILBDSPT0

34	52	FCLOSOPS	4	VSAM CLOSE options

Code:

Bits	Equate Name	Bit Settings	Meaning
0-7	FCLLOCK	0001 0000	LOCK
8-31	Reserved		

38	56	FSW1	4	Miscellaneous switches

Code:

Bits	Equate Name	Bit Settings	Meaning
0-7	FSOPEN	1000 0000	File is open
	FSLOCKED	0100 0000	File is closed with lock
	FSOPTNL	0010 0000	Optional file not present
	FSOKACT	0001 0000	Successful action has occurred since open
	FSEOF	0000 1000	Sequential read has encountered end-of-file
	FSVCORE	0000 0100	Main storage to process this open has been acquired
8-31	Reserved		

Displacement Hex	Displacement Decimal	Field	No. of Bytes	Description
3C	60	FTRSTMT	4	Transmission statement switches

Code:

0-7	FTREAD	0000 0100	READ statement
	FTWRITE	0000 1000	WRITE statement
	FTREWRT	0000 1100	REWRITE statement
	FTSTART	0001 0000	START
	FTDELET	0001 0100	DELETE statement
8-15	FTINVKEY	1000 0000	INVALID KEY
	FTATEND	0100 0000	AT END
	FTNEXT	0000 0010	NEXT
	FTKEY	0000 0001	KEY
16-23	FTSRCHGT	1000 0000	GREATER THAN
	FTSRCHEQ	0100 0000	EQUAL TO
	FTSRCHGE	0010 0000	NOT LESS THAN
24-31	Reserved		

40	64	FSYSCBAL	4	Address of system control blocks address list
44	68	FSYSCBLL	4	Address of system control blocks lengths list
48	72	FSYSCBNO	2	Number of system control blocks (DTF, DCB, ACB)
4A	74	FKEYLEN	2	Length of KEY data-name
4C	76	FRECCNT	4	Record count for checkpoint subroutine, if RERUN specified
50	80	FFIBAD	4	Address of File Information Block (FIB)
54	84	FWORKAD	4	Address of system-dependent work area
58	88	FRECA	4	Address of current record area
5C	92	FSAMRECA	4	Address of SAME RECORD AREA
60	96	FSTATKEY	2	STATUS KEY work area
62	98	FLASTREQ	1	Last I/O statement

Code:

Bits	Equate Name	Bit Settings	Meaning
0-7	FLASTRD	0000 0100	READ
	FLASTWRT	0000 1000	WRITE
	FLASTRWT	0000 1100	REWRITE
	FLASTSTR	0001 0000	START
	FLASTDLT	0001 0100	DELETE
	FLASTOPN	0001 1000	OPEN
	FLASTCLO	0001 1100	CLOSE

63	99		13	Reserved

COUNT PROGRAM DATA AREAS

The COUNT subroutines use the following data areas:

- The verb translate, verb, and verb text tables, contained in subroutine ILBDTC30
- The count table, contained in each object module
- The verbsum table, space for which is obtained dynamically by ILBDTC30. There is one table for all program units being monitored
- The count chain, space for which is gotten dynamically by subroutine ILBDTC00 for each program unit
- The node count table, which is part of the count chain
- The count common area, which is in subroutine ILBDTC00
- The debug common area, which is in subroutine ILBDDBG0

All these data areas are described below except the debug common area, which is described elsewhere in this section.

COUNT SUBROUTINE TABLES

Diagram 13 in "Section 2: Program Organization" shows the relationship among the six tables used by the COUNT subroutines. Their formats are shown below.

Verb Translate, Verb, and Verb Text Tables

The basic input to subroutine ILBDTC30 is count table entries, which desbribe the occurrence of verbs in the source program. These verbs are expressed in P1-code form to save space. ILBDTC30 uses the verb translate, verb, and verb text tables to translate the P1-code into the EBCDIC characters for the verb names.

Each verb translate table entry is one byte. Its hexadecimal displacement within the table corresponds to a unique P1-code. An entry with a displacement of X'25', for instance, represents the verb ACCEPT, the P1-code for which is X'25'. Each entry contains either X'FF' (if there is no verb for the corresponding code) or the entry number for the verb in the verb table.

The verb table contains a four-byte entry for each COBOL verb, the entries being arranged by verb name in alphabetic order. Byte one of each entry contains the length of the entry for the verb in the verb text table. The remaining three bytes contain the address of the verb text table entry.

The verb text table consists of all the COBOL verbs in EBCDIC format, listed in alphabetic order.

Count Table

The count table contains an entry for each verb encountered in the source program, in the order of its appearance. The format is as follows:

Byte	Contents
0	Identity code, as follows: 00 End of table 01 Procedure-name 02 Verb
1	Length of rest of entry (n)
2-4	Card number (omitted if byte 0=00)
5-6	Count-block number (X'00' if non-executable verb)
7	P1-code for verb or
7 though n + 1	Procedure-name in EBCDIC

Verbsum Table

The verbsum table contains an entry for each verb in the COBOL language, arranged in alphabetic order. The format is as follows:

Byte	Contents
0-1	Static verb count: the number of times the verb occurs in the program.
2-3	Dynamic count: the number of these verbs that are actually executed
4-7	The total number of times these verbs are executed.

For instance, if the source program contains three ACCEPT statements, only two of which are executed--one twice and the other three times, the static count is 3, the dynamic count 2, and the total execution 5.

COUNT CHAIN

There is one chain element for each program being monitored. The format of an element is as follows:

Displacement Hex	Displacement Decimal	Field	No. of Bytes	Description
0	0	TCFORPTR	4	Forward pointer
4	4	TCBACKPT	4	Backward pointer
8	8	TCPGMID	8	Program name
10	16	TCTMCNTB	4	Pointer to count table
14	20			Reserved
18	24	TCNODNUM	4	Number of counters in node count table
1C	28			Reserved
24	36	TCNODTBA	4	Pointer to start of node count table
28	40			Reserved
2C	44	TCPRVCNT	4	Pointer to previous COUNT COUNT TABLE
30	48	TCCHAINL	4	Length of this chain element
34	52	TCTGTPTR	4	Pointer to TGT of program being monitored
38	56			Node count table (variable length)

Node Count Table

Each entry is a halfword counter for a count-block in the source program. The position of the entry in the table corresponds to the number of the block.

Count Common Area

There is only one count common area, regardless of how many programs are being monitored. The contents of the area are as follows:

Displacement Hex	Displacement Decimal	Field	No. of Bytes	Description
0	0	TMCNFLG	1	Flags
				Equate Name / Code / Meaning: CNTFLG X'40' Programs being monitored; TCINIT X'20' First entry bit; CTPROCOF X'04' Count percents off; INTCRT X'10' In a count subroutine; CNTFLGOF X'BF' Turn off CNTFLG; INTCRTOF X'EF' Turn off INTCRT
1	1	TMCNFLG2	1	Flags
				Equate Name / Code / Meaning: TCERRFLG X'80' Processing count error; TCSVHDSW X'40' Save HEAD1 switch; NEEDSUMS X'20' Need count summary statistics
2	2	TMCNFLG3		Reserved
3	3	TMCNDECP	1	Decimal point is comma
4	4	CNTSUM	4	Count-sum
8	8	TCAVBSUM	4	Address of verbsum table
C	12	TCLVBSUM	4	Length of verbsum table
10	16		4	Reserved

Displacement Hex	Displacement Decimal	Field	No. of Bytes	Description
14	20	TCSAVR14	4	Save area for register 14
18	24		24	Reserved
30	48	TMCNCHN	4	First chain address of count chain
34	52	TMCNSV	72	Save area for ILBDTC00 and ILBDTC30
7C	124	TMCN2SV	72	Save area for ILBDTC20 and secondary save area for ILBDTC00
C4	196	TMCNWK1 through TMCNWKP	100	Work areas of four bytes each with names in the following series: TMCNW1 through TMCNWK9 and then TMCNWKA through TMCNWKP. Note: TMNCNWK2 must always be on a doubleword boundary.
128	296	TCSVHED1	121	Save area for HEAD1 of DBG0COM
1A1	417		3	Filler
1A4	420		56	Reserved

SECTION 4: DIAGNOSTIC AIDS

This section provides a few diagnostic aids for use in case an execution-time error occurs which is not a user error. Such an error may produce one of two results: an abnormal termination or an erroneous output from a compiled program.

Note: The compiling program-name, its version numbers, its modification number, and the PROGRAM-ID can be found at the end of the INIT1 routine in the listing of the program. INIT1 is at the end of the object module listing.

EXECUTION-TIME MESSAGES

A few messages, not specified by the user directly or by the system, may be printed during execution of the problem program. These messages originate in the COBOL library subroutines.

If the SYMDMP option is in effect, these messages are followed by the SYMDMP abnormal termination message and dump of the Data Division.

If the SYMDMP option is not in effect but the DUMP option is in effect, a partial dump is taken from the problem program origin to the highest virtual storage location of the last phase loaded. When this occurs, the eight bytes immediately preceding the DTF are destroyed.

The format of messages C112I through C125I is:

```
CmmmI SYSnnn filename DTFaddress text
```

See Figure 6 for mmm (message number) and text. These messages are issued on SYSLST and SYSLOG prior to cancellation of the job.

The debugging routines (SYMDMP, STATE, FLOW, and COUNT) themselves may, in addition to their normal diagnostic output, issue messages. The format of these is:

```
CmmmI {program-id      }text
      {card/verb-number}
```

See Figure 7 for mmm (message number) and text. These messages are written on SYSLST.

The ILBDMVE0 subroutine issues the following message on SYSLOG.

```
C126D  IS IT EOF?
```

The ILBDSSN0 subroutine issues the following message on SYSLOG and SYSLST:

```
C140I  INVALID SEPARATE SIGN
       CONFIGURATION
```

MESSAGE NUMBER	TEXT	SUBROUTINE
C112I	PARITY ERROR	ILBDSAE0
C113I	WRONG LENGTH RECORD	ILBDSAE0
C114I	PRIME DATA AREA FULL	ILBDISE0
C115I	CYLINDER INDEX TOO SMALL	ILBDISE0
C116I	MASTER INDEX TOO SMALL	ILBDISE0
C117I	OVERFLOW AREA FULL	ILBDISE0
C118I	DATA CHECK IN COUNT	ILBDISE0
C119I	DATA CHECK IN KEY OR DATA	ILBDDAE0
C120I	NO ROOM FOUND	ILBDDAE0
C120I	DASD ERROR	ILBDISE0
C122I	DASD ERROR WHILE ATTEMPTING TO WRITE RECORD ZERO	ILBDFMT0
C123I	FILE CANNOT BE OPENED AFTER CLOSE WITH LOCK	ILBDCLK0
C124I	CYLINDER AND MASTER INDEX TOO SMALL	ILBDISE0
C125I	NO EXTENTS	$$BCOBR1

Figure 6. Execution-Time Messages for I/O Error Conditions

MESSAGE NUMBER	TEXT	SUBROUTINE/ACTION
C150	IDENTIFIER NOT FOUND	ILBDMP13 - Dump request on line-control card for this identifier is ignored.
C151I	CARD NUMBER NOT FOUND	ILBDMP14 - Line-control card with non-existent card number is skipped.
C152I	VERB NUMBER NOT FOUND	ILBDMP14 - The nearest verb number is used instead of the specified one.
C153I	NO ROOM TO DUMP	ILBDMP11 and ILBDMP21 - Data Division dump (and sometimes COBOL statement number message) not given.
C154I	I/O ERROR ON DEBUG FILE	ILBDMP12, ILBDMP13, ILBDMP14, ILBDMP21, ILBDMP22, ILBDMP25 - SYMDMP output is cancelled for the program.
C155I	WRONG DEBUG FILE FORE PROGRAM	ILBDMP12 and ILBDMP21 - SYMDMP output is cancelled for the program.
C156I	NO ROOM FOR DYNAMIC DUMPS	ILBDMP12, ILBDMP13, ILBDMP14 - Dynamic dumping (but not abnormal termination dumping) is cancelled for the program.
C157I	INVALID FILE-NAME	ILBDMP11 - All SYMDMP output is cancelled for program.
C158I	INVALID LOGICAL UNIT	ILBDMP11 - All SYMDMP output is cancelled for program.
C159I	MISSING PARAMETERS	ILBDMP12 - The option with missing parameter is ignored.
C160I	INVALID OPTION	ILBDMP11 - The option is ignored
C161I	SUBSCRIPTING ILLEGAL	ILBDMP12 - The subscripts are ignored.
C162I	ON PARAMETER TOO BIG	ILBDMP12 - The number is reduced to 32767.
C163I	FLOW TRACE NON-CONTINUOUS. MORE THAN 10 PROGRAMS ENCOUNTERED	ILBDFLW0 - Tracing is terminated upon encountering an 11th PROGRAM-ID. Tracing resumes only upon returning to one of the original 10 programs.
C164	FLOW TRACE IN EFFECT BUT NO PROCEDURES TRACED	ILBDFLW0 - No action.
C165I	SYMDMP/STATE/FLOW/COUNT INTERNAL ERROR. EXECUTION CANCELLED	Job is cancelled.
C169I	STATE OPTION CANCELLED	ILBDSTN0 - Output cancelled.
C170I	INVALID ADDRESS	ILBDADR0 - Symbolic Dump is produced.
C171I	SPACE NOT FOUND FOR THE COUNT CHAIN. CONTINUING.	ILBDTC00 - Count output for the program is cancelled for this entry into the program unit.
C172I	SPACE NOT FOUND FOR THE VERBSUM TABLE. CONTINUING.	ILBDTC30 - Verb statistics suppressed.
C173I	FREEVIS FAILED. EXECUTION CANCELLED.	Job is cancelled.
C175I	INVALID COUNT TABLE ENTRY. EXECUTION CANCELLED.	Job is cancelled.

Note: Messages C150I through C162I may appear interspersed among the SYMDMP control cards at the point at which the error is recognized. PROGRAM-ID is specified for messages C153I through C162I. For C150I through C152I, the PROGRAM-ID is that of the nearest preceding program-control card, and the card/verb number of the corresponding line-control card is given instead.

Messages C153I through C155I may also appear in the midst of the dump output if the error condition is not recognized until dumping has started.

Figure 7. Error Messages from Debugging Subroutines

STORAGE LAYOUT

An example of the general storage usage for a COBOL program being executed in the background area is given in Figure 8.

The memory map printed as a result of the LISTX option contains the relative afdresses of the TGT fields, the literal pool, PGT fields, the instructions generated from the Procedure Division, and the INIT2, INIT3, and INIT1 routines, in that order. (See the publication IBM DOS/VS Compiler Program Logic, Order No. LY28-6423, for a discussion of these ion of these fields.) The absolute addresses can then be found with the assistance of the phase map (see Figure 9).

LOCATING A DTF

A particular DTF may be located in an execution time dump as follows:

1. Determine the order of the DTF address (DTFADR) cells in the TGT from the DTF numbers shown for each file-name in the GLOSSARY.

   Note: Since the order is the same as the order of FD's in the Data Division, it can be determined from the source program whether the SYM option was not used (that is, no GLOSSARY was printed).

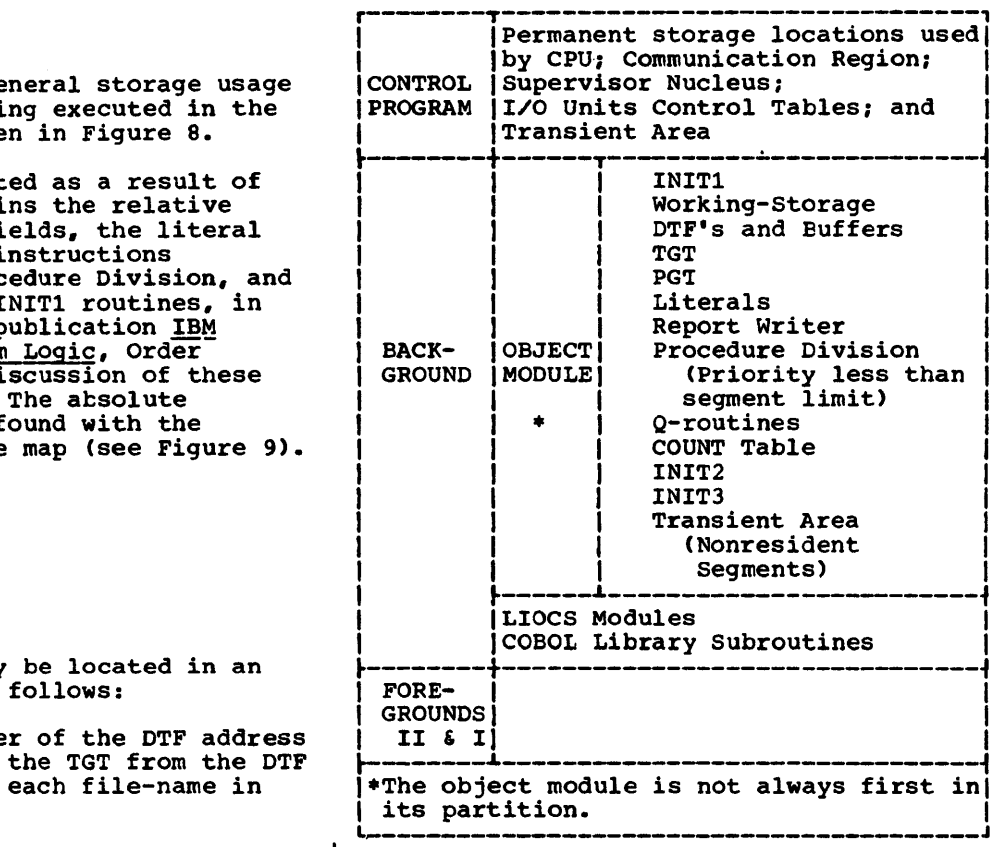


Figure 8. Example of Storage Used During Execution

```
        PHASE  XFR-AD  LOCORE  HICORE  DSK-AD    ESD TYPE   LABEL    LOADED  REL-FR
     PHASE***  07D878  07D878  07F1FF  05F OF 4  CSECT      TESTRUN  07D878  07D878  RELOCATABLE

                                                 CSECT      IJFFBZZN 07E1C8  07E1C8
                                                 *  ENTRY   IJFFZZZN 07E1C8
                                                 *  ENTRY   IJFFBZZZ 07E1C8
                                                 *  ENTRY   IJFFZZZZ 07E1C8

                                                 CSECT      ILBDSAE0 07F078  07F078
                                                    ENTRY   ILBDSAE1 07F0C0

                                                 CSECT      ILBDMNS0 07F070  07F070

                                                 CSECT      ILBDIML0 07F018  07F018

                                                 CSECT      ILBDDSP0 07E578  07E578
                                                    ENTRY   ILBDDSP1 07E978

                                                 CSECT      ILBDDSS0 07ECF0  07ECF0
                                                    ENTRY   ILBDDSS1 07EF50
                                                    ENTRY   ILBDDSS2 07EF48
                                                    ENTRY   ILBDDSS3 07F008
                                                    ENTRY   ILBDDSS4 07ED16
                                                    ENTRY   ILBDDSS5 07EDC2
                                                    ENTRY   ILBDDSS6 07EE22
                                                    ENTRY   ILBDDSS7 07EDEC
                                                    ENTRY   ILBDDSS8 07ED46

                                                 CSECT      IJJCPDV  07EAA8  07EAA8
                                                    ENTRY   IJJCPDV1 07EAA8
                                                 *  ENTRY   IJJCPDV2 07EAA8

* UNREFERENCED SYMBOLS                           WXTRN      STXITPSW
                                                 WXTRN      ILBDDBG2

002 UNRESOLVED ADDRESS CONSTANTS
```

Figure 9. Example of a Phase Map

2. Determine the relative starting address of the block of DTFADR cells from the TGT listing in the Memory map.

3. Calculate the absolute starting address of the block by adding the hexadecimal relocation factor for the beginning of the object module as given in the linkage editor map.

4. Allowing one fullword per DTFADR cell, count off cells from the starting address found in Step 3, using the order determined in Step 1, to locate the desired DTFADR cell.

5. The DTFADR cell contains the absolute address of the desired DTF.

   Note: The procedure for locating a secondary DTF is essentially the same, the only differences being that the SUBDTF address cells pointed to by the PGT are used and that the order of the cells is input, output, input/output, or input reversed.

LOCATING DATA

The location assigned to a given data-name may be similarly be found by using the BL number and displacement given for that entry in the GLOSSARY, and locating the appropriate one-word BL cell in the TGT. The hexadecimal sum of the GLOSSARY displacement and the contents of the cell should give the relative address of the desired area. This can then be converted to an absolute address as above.

SPECIAL DIAGNOSTIC AIDS FOR DEBUGGING SUBROUTINES

VIRTUAL STORAGE LAYOUT

The virtual storage layout of the Debug Subroutines when SYMDMP is in effect is shown in Diagram 6. (See "Program Organization" section.)

TABLES USED BY SYMDMP

The status of the tables built or referenced by the SYMDMP subroutines may reveal how much processing the SYMDMP subroutines had done before the dump occurred. Each of the tables on the Debug File is brought into virtual storage by the subroutine or subroutines which access it. The OBODOTAB table, however, is brought into virtual storage by subroutine ILBDMP21 and remains in virtual storage throughout the execution of the program. The other tables, listed in Figure 10, are built by the subroutines themselves from information located in the Debug File. The Debug File is designated as SYS005 during compilation; but it may be designated according to the user's option at execution time.

Figure 10 shows the tables used by the SYMDMP subroutines together with the compiler phases or the subroutines which use them.

TABLE	Built by	Used by	Location
CARDINDX	Phase65	ILBDMP14, ILBDMP25,	Debug File*
DATADIR	ILBDMP13	ILBDMP21, ILBDMP22	Virtual Storage
DATATAB	Phase 25	ILBDMP13, ILBDMP21, ILBDMP22	Debug File*
DYNAMTAB	ILBDMP12, ILBDMP13, ILBDMP14	ILBDMP21, ILBDMP22	Virtual Storage
FLOW TRACE	ILBDFLW0, ILBDFLW1	ILBDFLW2	Virtual Storage
OBODOTAB	Phase 25 (if ODO in program)	ILBDMP21, ILBDMP22	Debug File, Main Storage*
PCONTROL	ILBDMP11 ILBDMP12	ILBDMP10, ILBDMP11, ILBDMP12, ILBDMP13, ILBDMP14, ILBDMP20, ILBDMP21, ILBDMP22, ILBDMP23, ILBDMP24, ILBDMP25	Virtual Storage
PROCINDX	Phase 65	ILBDMP14, ILBDMP25	Virtual Storage
PROCTAB	Phase 65	ILBDMP14, ILBDMP25	Debug File*
PROGSUM	Phase 65	ILBDMP12, ILBDMP21 ILBDMP22	Debug File*
QUALNAMS	ILBDMP12	ILBDMP13	Virtual Storage
SEGINDX	Phase 65	ILBDMP14, ILBDMP25,	Debug File*

*Note: Each of the tables on the Debug File is read into virtual storage when it is used by one of the subroutines. The OBODOTAB table, however, is read into virtual storage by ILBDMP21 and remains there throughout the execution of the program which is being debugged.

Figure 10. Tables Used by Debugging Subroutines

$$BFCMUL	FH	01	A2
$$BPDUMP	EK	01	J5
$$BCOBEM	JB	01	B3
$$BCOBEM	EL	01	A1
$$BCOBER	HK	01	C3
$$BCOBER	HF	01	F3
$$BCOBER	GB	01	C4
$$BCOBER	FC	01	H4
$$BCOBER	EJ	01	A3
$$BCOBER	EG	01	C3
$$BCOBR1	EJ	01	H3
$$BCOBR1	EK	01	A1
ACCP02	EC	01	H4
ACCP03	EC	01	D3
ACCP04	EC	01	E4
ACT00002	HN	01	D1
ACT00004	HN	01	E1
ACT00680	HN	01	C2
ACT00700	HN	01	F1
ACT00704	HN	01	F2
ACT00736	HN	01	G1
APARTN	JU	01	D2
APWOTEXT	FA	01	E5
ASANDTVT	FA	02	C2
BASEADDR	HM	01	C1
BEFTSTVU	FA	01	G4
BIGTEST	FA	05	A1
BIN02	AA	02	A1
BIN03	AA	02	A2
BIN04	AA	02	A3
BIN05	AA	02	A4
BIN06	AA	02	A5
BIN07	AA	02	B1
BIN08	AA	02	D1
BIN10	AA	02	D2
BIN11	AA	02	E4
BIN12	AA	02	F4
BIN055	AA	02	B4
BIN058	AA	02	C4
BIN065	AA	02	C5
BIN105	AA	02	D3
BIN108	AA	02	D4
BLKADD	GA	01	B2
BLKLOAD	GA	01	D1
BN03B4	AA	03	F3
BOF	EH	01	E3
BOFBOV	EH	01	D3
BOFBOV	EI	01	D1
BOF1	EI	01	E2
BOV	EH	01	E4
BOV	EI	01	F1
BUFSD	FB	01	E2
BYTELOOP	JQ	01	C3
CALLDBG1	JU	01	B4
CALLD1D2	JS	01	J3
CALLSYM	JA	02	G1
CANTFIND	JW	01	G3
CHECKIT	JI	01	D3
CHECKIT	JI	01	F4
CHECKIT	JI	01	B4
CHKDMP	EK	01	J1
CHKNXT	FH	01	J3
CHKPOINT	CA	04	A4
CHKSEQ	HC	01	B3
CHKUSASI	HJ	01	J1
CHKWRAFT	HJ	01	G1
CHK1ST	EI	01	D2
CKCODE	JB	01	C2
CLEANIT	JR	01	D3
CLOSEFIL	JI	01	B5
CLOSEGIV	CA	03	F2
CLOSEIT	JI	01	D2
CLOSEIT	JI	01	B5
CLOSPRLG	F	03	A4
CNTRLRTN	JU	01	A3
COBDBG2	JC	01	G2
COLLECT	JK	01	B5
COMCLOSE	CA	02	E5
COMCLOSE	CA	05	G3
COMPTR	FH	01	D2
COMRTN	CA	05	H1
COM1	HF	01	D1
CONSOL	EC	01	D2
CONSOLE	EB	02	B3
CONTEXIT	KB	01	H1
CONT100	KB	01	C1
CONVRT	EB	01	C4
COUNTER	HC	01	E1
CVB0	AA	01	B2
CVB1	AA	03	B2
CVB02	AA	01	E2
CVB03	AA	01	C3
CVB005	AA	01	E1
CVB13	AA	03	D2
CVB015	AA	01	F4
CVB021	AA	01	F2
CVB022	AA	01	G2
CVB025	AA	01	E1
CVB035	AA	01	D3
CVBOUT	AA	01	G3
C2128	HH	01	H2
DAMPDS	JA	01	B1
DATEDAY	CCB	01	F3
DAY	CCB	01	H4
DEG6RET	JE	01	D1
DEAD	JC	01	A3
DEC03	AA	03	E5
DEC04	AA	03	G2
DEC05	AA	03	H1
DEC10	AA	03	J2
DEC13	AA	03	H3
DEC16	AA	03	K2
DEC17	AA	03	K3
DEC089	AA	03	H2
DELFEND	CBB	02	C2
DELFLDBG	CBB	02	A2
DELFLUP	CBB	02	C3
DELIMBEG	CBB	02	A1
DELIMLUP	CBB	02	B1
DEV2321	HH	01	B4
DEV3330	HH	01	J3
DIAGTEST	JG	01	B5
DIEXIT	EA	02	C2
DILOOP	EA	02	E3
DISP01	EA	01	F3
DIVIDE	FA	01	A4
DKTYPE	EK	01	C4
DMPB01	EA	02	C1
DOWITH14	JW	01	H2
DTFDA	EH	01	J3
DUMP	JC	01	B3

DUMPFD	JV	01	D4
DUMPIT	JP	01	H2
DUMPRD	JV	01	D5
DUMPSD	JV	01	B5
DUMPTGT	JV	01	C2
DUMPTGT	JS	01	B2
DYNCARD	JM	01	G2
EF	EI	01	C5
EJECT1	JB	01	J2
EJHEAD	JB	01	J4
ENDINIT	JJ	01	D3
EOF	CA	05	G2
EOF	HG	01	G2
EOF	HD	01	D2
EOF	GFA	01	G5
EOF	EI	01	F3
EOF	EH	01	F2
EOFIN	JK	01	D2
ERREXIT	JX	01	D2
ERROR	JO	01	G5
ERROR	JN	01	F5
ERROR	JL	01	E3
ERROR	JL	01	B3
ERROR	JL	01	K2
ERROR	JK	01	A1
ERROR	JK	01	C1
ERROR	HJ	01	F5
ERROR	HE	01	H4
ERROR	HD	01	G3
ERROR1	HF	01	E2
ERRTN	JB	01	B2
EXHIB	EA	01	F4
EXIT	JA	02	H3
EXIT	FH	01	E2
EXIT	FA	02	K3
EXIT	EH	01	K5
EXIT	CC	01	K2
EXIT	CCA	01	C3
EXIT	CCB	01	K3
EXIT1	CC	01	K1
E15ROUT	CA	02	A4
E35ROUT	CA	03	A1
FETCH	JP	01	H3
FETCHPHS	JJ	01	G2
FGLST	EJ	01	C3
FILLDY	JN	01	C4
FIRSTIME	FB	01	K2
FORMAT	HG	01	D2
GC01A3	CCA	01	G4
GDO0	CCA	01	B2
GET	GA	01	E5
GETBLOCK	CCC	01	A4
GETELM	JL	01	D2
GETNXT	FH	01	F3
GETPTR	ED	01	E1
GETSPACE	JR	01	B4
GETSTATE	JW	01	B4
GIVECORE	JQ	01	D2
GIVING	CA	03	E1
GOCOBOL	CA	02	J5
GOODBYTE	JK	01	D5
GOODRET	JL	01	H3
GOTCDVB	JW	01	F4
GOTPARM	KC	01	D1
GOTPRID	JR	01	D1
GOTSX	JW	01	C4
GTPXBLK	JO	01	B4
GT256	CB	01	D2

HAVELEN	JV	01	E2
HEADERTN	JU	01	C2
HEADLINE	JG	01	D4
HEXDUMP	JV	01	F2
HEXDUMP	JV	01	G4
HEXDUMP	JU	01	C5
HEXDUMP	JR	01	H3
HEXDUMP	JQ	01	A2
HEXOUT	JQ	01	F5
HEXRTN	JU	01	C5
HIDDN	FA	04	A4
HPOS	FA	04	A3
H3330	HH	01	D1
IBERR	GA	01	F2
IC1GT18	HH	01	G4
IC2GT198	HH	01	G2
IC2GT254	HH	01	J4
IC2GT8	HH	01	E4
IH1GT3	HH	01	C4
IH2GT8	HH	01	D2
IH2GT18	HH	01	D3
ILBDABX0	FG	01	A4
ILBDACP0	EC	01	A1
ILBDACS0	CCD	01	A1
ILBDACS1	CCD	01	A3
ILBDASY0	EE	01	A3
ILBDBUG0	JY	01	A1
ILBDCKP0	CB	01	D4
ILBDCKP0	ED	01	A1
ILBDCKP1	EG	01	A2
ILBDCKP2	EG	01	A3
ILBDCKP3	EG	01	A4
ILBDCLK0	EG	01	A3
ILBDCMM0	CCC	01	A1
ILBDCMM1	CCC	02	A1
ILBDCRD0	HA	01	A3
ILBDCT10	KB	01	A1
ILBDCVB0	AA	01	A2
ILBDCVB1	AA	03	A2
ILBDDAE0	HD	01	G3
ILBDDAE0	HE	01	H4
ILBDDAE0	HI	01	K4
ILBDDAE0	HJ	01	F5
ILBDDAE0	HK	01	A2
ILBDDBG0	JH	01	B1
ILBDDBG0	JH	01	G5
ILBDDBG0	JP	01	J4
ILBDDBG0	JA	01	A1
ILBDDBG0	CC	01	G3
ILBDDBG1	JB	01	A1
ILBDDBG1	JW	01	H4
ILBDDBG1	JV	01	J5
ILBDDBG1	JV	01	F5
ILBDDBG1	JV	01	B5
ILBDDBG1	JV	01	E4
ILBDDBG1	JV	01	F3
ILBDDBG1	JU	01	B4
ILBDDBG1	JS	01	G4
ILBDDBG1	JR	01	G1
ILBDDBG1	JQ	01	E4
ILBDDBG1	JK	01	D3
ILBDDBG1	JK	01	E1
ILBDDBG1	JG	01	H4
ILBDDBG1	JG	01	J5
ILBDDBG1	JF	01	J2
ILBDDBG2	JH	01	G5
ILBDDBG2	JC	01	A1
ILBDDBG3	JC	01	A4
ILBDDBG4	JC	01	A5
ILBDDBG5	JD	01	A1

ILBDDBG6	JE	01	A1
ILBDDBG8	JE	01	A3
ILBDDIO0	HI	01	A1
ILBDDIO1	HI	01	A2
ILBDDIO2	HI	01	A3
ILBDDIO3	HI	01	A4
ILBDDIO4	HI	01	A5
ILBDDSR0	HD	01	A1
ILBDDTE0	CCB	01	A2
ILBDDTE1	CCB	01	A3
ILBDDTE2	CCB	01	A4
ILBDFLW0	JG	01	A1
ILBDFLW0	JA	01	E4
ILBDFLW1	JG	01	A2
ILBDFLW2	JG	01	A4
ILBDFLW2	JW	01	K4
ILBDFLW2	JC	01	H5
ILBDFLW3	JG	01	A3
ILBDFMT0	HF	01	A1
ILBDGDO0	CCA	01	A2
ILBDGDO1	CCA	01	A4
ILBDGDO2	CCA	01	A5
ILBDIDA0	HD	01	G1
ILBDIDA0	HF	01	G1
ILBDIDA0	HH	01	A1
ILBDIML0	FD	01	A3
ILBDINS0	CBC	01	A1
ILBDISE0	GB	01	A1
ILBDISE1	GB	01	A2
ILBDISM0	GA	01	A1
ILBDISM1	GA	01	A2
ILBDISM2	GA	01	A4
ILBDISM3	GA	01	A5
ILBDMFT0	FE	01	A2
ILBDMOV0	CB	01	A2
ILBDMOV0	EI	01	B3
ILBDMOV0	FA	02	C4
ILBDMOV0	FA	02	D3
ILBDMOV0	FA	03	G3
ILBDMOV0	FB	01	K2
ILBDMP01	JI	01	A2
ILBDMP01	JN	01	C1
ILBDMP01	JN	01	G2
ILBDMP01	JO	01	C2
ILBDMP01	JO	01	F2
ILBDMP01	JO	01	A4
ILBDMP01	JO	01	C4
ILBDMP01	JS	01	F2
ILBDMP01	JT	01	G1
ILBDMP02	JI	01	A2
ILBDMP02	JN	01	C1
ILBDMP02	JN	01	G2
ILBDMP02	JO	01	C2
ILBDMP02	JO	01	F2
ILBDMP02	JO	01	A4
ILBDMP02	JO	01	C4
ILBDMP02	JS	01	F2
ILBDMP04	JX	01	A1
ILBDMP10	JH	01	D1
ILBDMP10	JJ	01	A2
ILBDMP10	JL	01	K3
ILBDMP10	JM	01	H5
ILBDMP10	JA	02	H1
ILBDMP10	JA	01	D4
ILBDMP10	JA	01	C2
ILBDMP11	JH	01	F1
ILBDMP11	JJ	01	E2
ILBDMP11	JL	01	A2
ILBDMP12	JH	01	H1
ILBDMP12	JJ	01	G2
ILBDMP12	JM	01	A2
ILBDMP12	JN	01	K4
ILBDMP13	JH	01	K1
ILBDMP13	JM	01	D5
ILBDMP13	JN	01	A1
ILBDMP14	JH	01	K2
ILBDMP14	JJ	01	H2
ILBDMP14	JO	01	A2
ILBDMP20	JH	01	A3
ILBDMP20	JJ	01	F4
ILBDMP20	JP	01	A2
ILBDMP20	JR	01	K1
ILBDMP20	JR	01	E2
ILBDMP20	JR	01	K5
ILBDMP20	JS	01	E3
ILBDMP20	JS	01	B5
ILBDMP20	JW	01	B3
ILBDMP20	JW	01	K5
ILBDMP20	JC	01	J3
ILBDMP20	JA	01	D4
ILBDMP20	JA	01	C2
ILBDMP21	JH	01	D3
ILBDMP21	JH	01	B4
ILBDMP21	JP	01	F2
ILBDMP21	JP	01	H3
ILBDMP21	JR	01	A1
ILBDMP22	JH	01	F3
ILBDMP22	JP	01	H2
ILBDMP22	JS	01	A1
ILBDMP22	JT	01	K1
ILBDMP22	JV	01	J2
ILBDMP22	JV	01	J4
ILBDMP23	JH	01	H4
ILBDMP23	JS	01	J2
ILBDMP23	JU	01	A2
ILBDMP23	JV	01	K4
ILBDMP24	JH	01	J3
ILBDMP24	JS	01	C2
ILBDMP24	JS	01	J3
ILBDMP24	JV	01	A2
ILBDMP25	JH	01	E3
ILBDMP25	JP	01	G2
ILBDMP25	JW	01	A2
ILBDMRG0	CA	05	A1
ILBDMVE0	FF	01	A3
ILBDNSL0	EI	01	A1
ILBDNSL2	EI	01	A2
ILBDRCR0	HB	01	A3
ILBDRDI1	HJ	01	A2
ILBDRDS0	HI	01	A2
ILBDRFM0	HH	01	A2
ILBDOSY0	EF	01	A2
ILBDSAE0	FC	01	A3
ILBDSAE1	FC	01	A4
ILBDSEM0	CC	01	A1
ILBDSIO0	F	01	A3
ILBDSIO1	F	01	A1
ILBDSPA0	FA	01	A1
ILBDSPA1	FA	01	A2
ILBDSRT0	CA	01	A1
ILBDSSN0	IA	01	A1
ILBDSSN1	IA	01	F1
ILBDSTG0	CBA	01	A1
ILBDSTN0	JF	01	A1
ILBDSTN0	JC	01	K5
ILBDTC00	KA	01	A2
ILBDTC20	KC	01	A1

ILBDTC30	KD	01	A1
ILBDUSL0	EH	01	A1
ILBDUSL1	EH	01	A2
ILBDUSL2	EH	01	H1
ILBDUST0	CBB	01	A1
ILBDVBL0	FA	03	D4
ILBDVBL0	FA	01	G5
ILBDVBL0	FB	01	A1
ILBDXTN0	HC	01	A1
INCIH2	HH	01	F1
INCORE	CC	01	H1
INIT	CC	01	J2
INIT	F	01	B1
INIT	CCA	01	D5
INITDYN	JR	01	B5
INITFLS	CA	05	D3
INITLOOP	CA	05	D4
INITLOOP	F	01	E1
INIT60	F	01	D2
INITRTN	CA	05	B2
INPFINAL	CA	02	J4
INTG3	FA	04	H3
INT0	HL	01	C1
INT0002	HL	01	D1
INVKEY	GA	01	D2
IPTRL	EI	01	C4
ISDMP2IN	JS	01	C2
ISITALL	JQ	01	F4
ISITALL	JV	01	H3
ISITHEX	JM	01	B4
ISITLAST	JR	01	H5
ISITNEW	JR	01	C1
ISITPX	JW	01	D4
ISQFOUND	JN	01	D4
LASTPRNT	JG	01	F5
LBRET	EI	01	H5
LBRET	EH	01	H5
LBRET1	EH	01	J5
LINELOOP	JQ	01	E2
LNBYPASS	FA	05	D3
LNINT	FA	05	G1
LNPAGE	FA	05	F2
LNPOST	FA	05	A3
LN40	FA	05	A2
LOAD	CA	01	D4
LOAD	JA	02	F1
LOADP2	JJ	01	E4
LOOPDBG2	JC	01	F1
LPUBPTR	FH	01	K3
LTAFTER	FA	07	B1
LTBEFORE	FA	07	B2
LWRCASE	EC	01	G4
LYASA	FA	06	B5
LYASCII	FA	06	E4
MAIN	JA	02	C1
MDCLOSE	HM	02	A1
MDC1000	HM	02	B1
MDC1004	HM	02	C1
MDC1006	HM	02	D1
MDC1008	HM	02	E1
MDC1020	HM	02	F1
MDC1060	HM	02	G3
MDC1064	HM	02	H2
MDC3000	HM	02	J2
MDC3004	HM	02	K2
MDC3016	HM	02	K4
MDC3050	HM	02	K5
MDOPEN	HM	01	D1
MDO1000	HM	01	E1
MDO1002	HM	01	F2
MDO1004	HM	01	H1
MDO1006	HM	01	J2
MDO1008	HM	01	H2
MDO3000	HM	01	B4
MDO5002	HM	01	E4
MDO6000	HM	01	A5
MDO6004	HM	01	B5
MDO6016	HM	01	C5
MDO7000	HM	01	E5
MFT1	FE	01	G2
MFT2	FE	01	H2
MNEMONI	FA	01	G2
MNEMONIX	FA	01	H2
MNEMON2	FA	02	H5
MORETOGO	JN	01	B2
MOVE	CBB	04	A1
MOVE	JD	01	C3
MOVE	EI	01	B2
MOVECORE	CA	01	G1
MOVEKEY	HD	01	E2
MOVE1	FA	02	D2
MVCTLCHR	FA	02	G1
MVTOBF	EA	02	E4
MVTOBF	EB	01	G3
NEWDNAME	JN	01	H2
NEWFILE	JI	01	E2
NEWLINE	JQ	01	B2
NEWPROG	JR	01	D4
NEXTCELL	JV	01	D2
NEXTDBG2	JC	01	K1
NEXTENTRY	JS	01	G2
NLS5	EI	01	G4
NOCOB	JA	02	B2
NODUMP	EK	01	K1
NOLTINT	FA	05	F1
NONLD	JV	01	B3
NOPDUMP	JR	01	J3
NOPROC	EH	01	F4
NOSEGM	JA	02	D2
NOSYM	JA	01	E3
NOSYM	CA	01	C4
NOTBOMB	JR	01	E1
NOTDTFMT	ED	01	H1
NOTEND	JQ	01	G3
NOTFIRST	JA	01	D2
NOTFST	CC	01	B1
NOTOPEN	ED	01	C2
NOTOUT	JA	01	D3
NOTSORT	ED	01	E3
NOTVASA	FA	03	F2
NSL4	EI	01	A5
NUMMOVE	CBB	04	A5
NXTCRD	JL	01	E4
NXTDATAB	JN	01	D2
NXTDY	JO	01	J4
NXTDYCRD	JM	01	B5
NXTENTRY	JT	01	A1
NXTENTRY	JV	01	H3
NXTEXIT	JT	01	K5
NXTEXIT	JT	01	H2
NXTPRM	EA	02	G4
NXTPROG	JJ	01	E2
NXTXTNT	HD	01	H2

OBODORTN	JU	01	D3
OCBRET	F	02	J1
ODODONE	JR	01	K4
ONLIST	EK	01	E1
OPCL	F	02	A1
OPE20	F	07	J3
OPE22	F	07	A4
OPE30	F	07	D4
OPE32	F	07	F4
OPE34	F	07	H4
OPE36	F	07	K4
OPEL10	F	07	F2
OPEL20	F	07	A3
OPEL30	F	07	D3
OPEN	EF	01	E2
OPENDTF	ED	01	G5
OPENIT	JI	01	B2
OPENPRLG	F	03	A1
OPENEPLG	F	07	E1
OPEN1	EF	01	C4
OPEN2	JI	01	C2
OPESPA	F	07	C5
OPN	JB	01	E3
OUTFINAL	CA	03	K1
PACKED	FA	01	K1
PFLOW	JC	01	G5
PLCB01	AA	02	J2
PLCB02	AA	02	J3
POINT	EH	01	G1
POINTFIL	JI	01	B4
POINTNXT	JV	01	G2
PRELUDE	HL	01	B1
PRELUDE	HM	01	B1
PRELUDE	HN	01	B1
PRINT	JW	01	H4
PRINT	JV	01	E4
PRINT	JV	01	J5
PRINTDMP	JQ	01	E4
PUT	GA	01	K3
PUT	FA	03	H2
PUT	EB	02	D1
PUT	CA	03	H1
PWCL1	FC	01	H3
RCDEL1	CCB	03	F2
RDEOF	F	04	B4
RD14	HJ	01	C4
READ	CA	02	E4
READ	CA	05	F1
READ	F	04	B1
READEXIT	JI	01	J3
READEXIT	JK	01	K5
READFIL	JI	01	B3
READIPT	JM	01	K4
READIPT	JM	01	B5
READIPT	JM	01	D4
READIPT	JM	01	C3
READIPT	JM	01	H2
READIPT	JM	01	E2
READIPT	JL	01	E4
READIPT	JL	01	J3
READIPT	JL	01	H2
READIPT	JL	01	D2
READIPT	JK	01	A1
READIPT	JJ	01	D2
READIT	JI	01	C3
READIT	JI	01	H3
READIT	JI	01	E4
READLIB	EI	01	J1
READXX	F	04	C1
READ2	JK	01	B2
READ4	JK	01	C3
RECEIVBG	CBB	03	A1
RECMOVE	CBB	03	J1
REMAINER	FA	02	F4
RESTORE	JR	01	E3
RESUME	CA	01	B1
RESUME	CBB	01	D3
RET	JO	01	H5
RET	JP	01	J2
RET	JW	01	K5
RET	JW	01	B3
RET	JV	01	J2
RET	JV	01	J4
RET	JS	01	B5
RET	JR	01	K5
RETAD	CA	02	F4
RETLOGST	F	06	B5
RETN	JB	01	E1
RETURN	F	01	G2
RETURN	CB	01	B3
RETURN	EE	01	E3
RETURN	EF	01	F2
RETURN	EH	01	B4
RETURN	EB	02	D3
RETURN	HC	01	H3
RETURN	HD	01	K1
RETURN	HH	01	G1
RETURN	FG	01	A2
RETURN	KC	01	H1
RETURN1	EG	01	B2
REWEOX	F	06	B4
REWRITE	F	05	B1
REWRITEXX	F	05	C1
RFR	CCA	01	H3
ROOT	CCA	01	G2
SAVEREC	FA	01	E1
SEEKON	JM	01	J2
SEGLAB1	CC	01	E1
SEGWT	CC	01	G1
SETAPWOS	FA	03	A4
SETDISP	JT	01	B2
SETITEM	JY	02	A1
SETPN	JY	02	A3
SETSTOP	EC	01	C1
SIMPMFG	CBB	04	F2
SIMPMFL	CBB	04	E4
SIMPMOVE	CBB	04	E1
SINGLEON	JI	01	D4
SIOTAB	F	01	D3
SIO0	F	01	B3
SMPEXMOV	CBB	04	H1
SPACECTL	FA	02	F3
SPECDEL	CBA	02	B5
SPECENTR	KD	01	B3
SSA	HJ	01	G5
STACK	JG	01	F2
START	JJ	01	B2
STEPID	JS	01	C4
STEPUP	JN	01	E3
STEP2	KA	01	D2
STEP2	KD	01	C1
STEP2400	KA	01	J2
STEP3	KD	01	F1
STEP4	KD	01	G1
STEP5	KD	01	H2

STOQNAME	JM	01	J4
STOREKEY	HJ	01	H1
SYM	JA	01	C2
SYMDMP	JD	01	E1
SYMDMP	JH	01	A1
TAPEREC	ED	01	J3
TCERRRTN	KC	01	C2
TCSRET	FA	06	F4
TC00EXIT	KA	01	J2
TC3EXIT	KD	01	E2
TESTCORE	CA	01	F1
TESTMODE	CA	01	E1
TESTVAR	CA	03	G1
TEXTCTL	FA	01	C4
TEXT01	FA	01	D2
TGTLOOP	JA	02	B1
TIMER	CCB	01	F2
TRANS	FC	01	H4
TRANS	HK	01	B3
TRANS	GB	01	C3
TRUNC	FB	01	K1
TRUNC	FB	01	G5
TRYFLOW	JW	01	J4
TRYPDUMP	JR	01	F3
TRY14	JW	01	F2
TSTASA	FA	01	D2
TSTDTFSD	EH	01	H3
TSTERR	HJ	01	F4
TSTFILSZ	CA	01	D1
TSTINPRC	CA	02	C4
TSTLAST	CA	03	C4
TSTMNEM	FA	01	F2
TSTOF	EH	01	D1
TSTPCH	EF	01	B3
TSTPRM	EA	02	B3
TSTPRM	EB	01	B3
TSTSYM	HC	01	A3
TYPERTN	JU	01	F2
TYTYPE	EK	01	B3
UNDFVUND	FA	03	B3
UNIQUE	JR	01	G5
UNITCK	EA	01	C2
UNITCON	EA	01	D4
UNITREC	FG	01	G2
UNIT01	EA	01	D2
UNIT10	EA	01	E2
UNIT12	EA	01	F2
UNIT15	EA	01	H2
UNSOVFLO	CBB	01	C4
UNSTEND	CBB	01	F3
UNSTEND0	CBB	01	D3
UNSTLUP	CBB	01	A3
UNSTR1	CBB	01	G1
UNSTYPE	CBB	01	A5
UPDY	JO	01	J2
UPLAST	JC	01	G3
USASIRD	HJ	01	G4
USER	HK	01	F2
USER	GB	01	G2
USER	FG	01	J2
USER	FC	01	K3
USER	EI	01	K2
USING	CA	02	D4
VARET	EA	02	D4
VGIV	CA	03	G2
VIO0	HN	01	C1
VNOP	CC	01	D1
VUNBTEXT	FA	03	B3
VUNDSPAC	FA	03	D3
WILLFIND	JR	01	F5
WORDLOOP	JQ	01	B3
WRITDIAG	JG	01	C5
WRITE	CA	03	F1
WRITE	F	06	B1
WRITEEBUF	JG	01	E4
WRITELBL	EI	01	G3
WRITEXX	F	06	C1
WRITE1	JB	01	F5
WRITE1	JB	01	B4
WRITE1	JB	01	D1
WRITE1	JB	01	D2
WRITE1	JB	01	D3
WRTIE1	JB	01	D4
WRITE1	JB	C1	D5
WRITE1	JB	01	J2
WRITE1	JB	01	J3
WRTIE1	JB	01	J4
WRTFOOT	JB	01	D3
WRTFOOT	JB	01	J3
WRTHEAD	JB	01	D4
WRTNOEFJ	JB	01	D2
WRTPASS	JB	01	D1
XCCSRCH	EM	01	C1
XDELIM	CBC	02	A1
XDLMLOOP	CBC	02	D1
XFOUND	EM	01	E1
XINVOKE	EM	01	F1
XOVBACK	AA	03	K4
XSCAN	CBC	03	A1
XSCNCLS	CBC	03	G1
XSCNCONT	CBC	03	F5
XSCNEND	CBC	03	J3
SXCNFOUN	CBC	03	D3
XSCNTRT	CBC	03	C1
XSEARCH	EM	01	D1
XSSUCKPT	ED	01	D3
XTERM	CBC	04	A1
XTRMCLS	CBC	04	C1
YESROOT	JW	01	D3
YESTRANS	JW	01	E3
ZEROC2	HH	01	J1

GLOSSARY

The words listed below are defined according to their use in this book, and the definitions are not necessarily applicable elsewhere. Efforts have been made to exclude terms which are common in the programming profession unless they are used in a special sense.

Base Locator (BL): A 4-byte address cell in the TGT. There is one BL pointing to the Report Section, one to the Working-Storage Section, and one to each FD, SD, and RD entry. Any FD, SD, or RD entry exceeding 4,096 bytes has one BL assigned to each 4,096 bytes. The compiler loads a register with each address unless there are too many BL's. In that case, it loads registers with BL's as they are needed.

Base Locator for Linkage Section (BLL): A 4-byte address cell in the TGT. BLL's are assigned by counter and are unique. BLL1 points to a work area used to process label records. BLL2 through BLLn are assigned to each 77-level and each 01-level entry in the Linkage Section. Any 77- or 01-level entries exceeding 4,096 bytes have one BLL assigned per 4,096 bytes.

BL: See Base Locator.

BLL: See Base Locator for Linkage Section.

Count-block: A set of COBOL verbs such that (exclusive of ABENDs) each verb in the block is executed if, and only if, the first verb is executed.

Debug Text: A type of debugging text which contains card numbers, their displacement within the object module, the priority of each segment, and discontinuity elements.

Dummy PN: A procedure-name, defined by the user, but not referenced in any branch instruction.

Fragments: A portion of code having a maximum size of one less than 64K bytes (65,535). A fragment begins with the first byte of a verb and ends with the first byte of a verb preceding the verb with a final relative displacement greater than 64K bytes. This unit is used in processing for the SYMDMP or STATE option.

GN: See Procedure-name.

Initialization Routines: Collectively, routines INIT1, INIT2, and INIT3. These are generated by the compiler as part of the object module.

Linkage: As used in this book "linkage" is synonymous with "calling sequence."

Major Code: A 4-bit code identifying the different types of DATATAB and CBODOTAB table entries.

Master of an OCCURS Clause with the DEPENDING ON Option: A data-name for a variable-length group item which does not itself contain an OCCURS clause with the DEPENDING ON option, but at least one of its subordinate items at the next level does contain an OCCURS clause with the DEPENDING ON option.

Minor Code: A 4-bit code identifying the type of operand in the DATATAB table entry of an LD item.

Node: The beginning of a count-block.

Object module: The result of a successful compilation. It is the output of a single execution of the compiler and is the input to the linkage editor.

Optimizer (OPT) Option: An option which directs the compiler to produce an object module, optimized for PN addressability and register usage. The resultant code uses Procedure Blocks to address procedure-names that are referenced in branch instructions. This option reduces the number of instructions required for branches.

PGT: See Program Global Table.

PN: See Procedure-name.

Priority: See Segmentation.

Procedure-name (GN, PN, or VN): The name of a point in a program which can be the object of a branch instruction. PN's are the user-assigned procedure-names which correspond to paragraph or section names in the Procedure Division of the source program. GN's are compiler-generated and are inserted wherever a need for an additional name occurs. VN's are variable names; that is, they may vary at execution time because of a PERFORM or ALTER statement. All procedure-names are unique since they include a number assigned by a counter (e.g., PN1, VN2, GN1, and GN2).

Procedure Block: Unit of addressability in the object module where the optimizer (OPT) option is specified. Each Procedure Block

consists of approximately 4096 bytes of code. Most PN's and GN's within a Procedure Block are addressed as displacements added to a base register which contains the address of the first instruction of the Procedure Block.

Program Global Table (PGT): A part of the object module. The PGT contains virtuals, literals, and addresses used during execution.

Program unit: Any COBOL main program or any COBOL subprogram.

Q-routine: One of a set of routines generated by Phase 22. Q-routines calculate the length of variable-length fields and the location of variably located fields resulting from an OCCURS clause with the DEPENDING ON option.

Root Segment: See Segmentation.

SBL: See Secondary Base Locator.

Secondary Base Locator (SBL): A 4-byte address cell in the TGT. The compiler assigns a unique SBL, using a counter in COMMON, to each variably located field. At execution time, each SBL points to its field. Variably located fields are those which follow a variable-length field and which are not new files or records; they occur as a result of OCCURS...DEPENDING ON statements in the source program. If a variably located field exceeds 4,096 bytes, the compiler assigns one SBL to each 4,096 bytes.

Section: A series of source program procedure instructions grouped under the same section-name.

Segment: A section or a group of sections all having the same priority.

Segmentation: A special feature of the compiler which permits the programmer to organize his program into several load modules. Each section in the Procedure Division is assigned a priority number. All sections having the same priority are loaded together as a segment. One of these, the root segment, resides in virtual storage throughout execution of the program. The other segments are loaded in order of the priority number, each segment overlaying the one before.

Table: An area in virtual storage containing a number of entries of a fixed, often identical, format.

Task Global Table (TGT): A part of the object module. The TGT contains information, addresses, and work areas for use during execution.

TGT: See Task Global Table (TGT).

Transient area: A portion of virtual storage reserved during execution time to contain segments which are not permanently resident. It contains one such segment at a time and is large enough to hold the largest nonresident segment in the program.

UPSI: User Program Status Information. There are eight 1-bit UPSI switches provided by the DOS/VS system. The UPSI feature of this compiler provides the facility of naming and using these switches.

Verb string: A verb and its operands.

Virtual: The name of a procedure or table referenced by a procedure, but not defined in the source module. It is necessary because of a CALL to an external procedure or a branch to a CCBOL library subroutine. At execution time, the address of all procedures referred to by virtuals (which have been link edited into the load module) are stored in the Program Global Table.

VN: See Procedure-name.

INDEX

E

F

G

H

I

P

Q

R

S

T

IBM DOS/VS COBOL
Subroutine Library
Program Logic
LY28-6424-02

Reader's Comment Form

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name and address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

Fold and Staple

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

Postage will be paid by:

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and Staple

IBM DOS/VS COBOL
Subroutine Library
Program Logic
LY28-6424-02

Reader's Comment Form

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name and address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

LY28-6424-02

Fold and Staple

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

Postage will be paid by:

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and Staple

IBM DOS/VS COBOL Subroutine Library File No. S370-24 LY28-6424-02

IBM DOS/VS COBOL
Subroutine Library
Program Logic
LY28-6424-02

Reader's Comment Form

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name and address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

LY28-6424-02

Fold and Staple

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

Postage will be paid by:

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and Staple

IBM®

IBM DOS/VS COBOL Subroutine Library File No. S370-24 LY28-6424-02

IBM DOS/VS COBOL
Subroutine Library
Program Logic
LY28-6424-02

Reader's Comment Form

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name and address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

Fold and Staple

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

Postage will be paid by:

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and Staple

LY28-6424-02