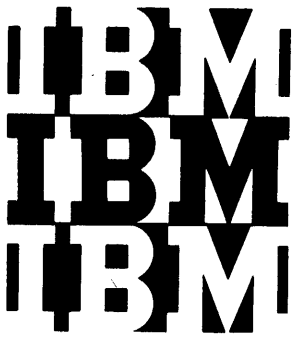


File No. S360-32
Form Y28-6614-4

Program Logic

IBM System/360 Operating System

Utilities

Program Logic Manual

Program Number 360S-UT-506

This publication describes the internal logic of the utility programs provided for the IBM System/360 Operating System:

- System utilities, which are executed under the operating system to manipulate system data sets such as catalogs.
- Data set utilities, which are executed under the operating system to work with data sets at the logical-record level.
- Independent utilities, which are executed outside of the operating system to dump, restore, and recover data, and to initialize and assign alternate tracks on direct access devices.

In addition to descriptive text, this publication contains flowcharts of the programs, figures showing the formats of the major tables and records, and an appendix that lists the modules of the utility programs.

Program Logic Manuals are intended for use by IBM customer engineers responsible for program maintenance, and by system programmers involved in altering the program design. Because program logic information is not necessary for program operation and use, distribution of this manual is restricted to persons with program maintenance or modification responsibilities.

Preface

The purpose of this publication is to enable the reader to locate specific areas of the utility programs provided for the IBM System/360 Operating System, and to relate those areas to the corresponding program listings.

The publication is divided into three major sections, corresponding to the three major types of utility programs: system utilities, data set utilities, and independent utilities. Each section contains descriptions of the programs of the corresponding type; these descriptions consist of text, flowcharts, and figures showing record and table formats.

The introduction provides a brief description of each utility program, and an appendix lists the modules of the utility programs.

To use this publication effectively, the reader should have an understanding of the material in the following publications:

IBM System/360 Operating System:

Principles of Operation, Form A22-6821

Utilities, Form C28-6586

Concepts and Facilities, Form C28-6535

System Control Blocks, Form C28-6628

Introduction to Control Program Logic
Program Logic Manual, Form Y28-6605

Fifth Edition (November, 1968)

This is a major revision of, and obsoletes, Y28-6614-3. It contains the following new or modified material.

- IEBDG. This is an added Data Set Utility Program. This program is used to generate test data sets to be used as a debugging aid.
- IEHMOVE, IEBUPDTE, IEBCOMPR, IEBGENER, IEBPTPCH. These programs have been modified for variable spanned records and/or user label exits.
- IEBCOPY. This program has been modified for data set compression.
- The blocking factor for data set utility programs and for system utility programs that use QSAM has been modified.

Other changes to text, and small changes to illustrations, are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol • to the left of the caption.

This edition applies to release 17 of IBM System/360 Operating System and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 SRL Newsletter, Form N20-0360, for the editions that are applicable and current.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

Contents

Illustrations

Figures

Tables

Introduction

IBM System/360 Operating System provides the user with utility programs that perform basic operations. These programs are grouped in three categories: system utilities, data set utilities, and independent utilities.

System utilities are executed under the operating system; these programs treat data associated with the structure of the operating system. They are:

- IEHPROGM, a program that modifies control data contained in catalog and volume structures.
- IEHMOVE, a program that duplicates collections of data sets to produce extra copies or rearrange existing ones.
- IEHLIST, a program that lists a catalog (or a portion thereof), a volume table of contents, and the directory of a partitioned data set.
- IEHIOSUP, a program that updates the Transfer Control (XCTL) tables embedded within load modules and access executor modules for the I/O support functions OPEN, CLOSE, and EOV.
- IFCDIP00, a program that writes the SYS1.LOGREC data set in initialized format.
- IFCEREP0, a program that edits and prints environmental records from SYS1.LOGREC.
- IEHUCSLD, a program that loads the 2821 generator storage with user-supplied character images.
- IEHINITT, a program that creates volume labels on magnetic tape.
- IEHDASDR, a program that dumps, restores, and initializes direct access volumes.

Data set utilities are executed under the operating system and perform operations on data sets at the logical record level. They are:

- IEBCOPY, a program that copies all or a specified portion of a partitioned data set.
- IEBCOMPR, a program that compares two data sets at the logical record level.
- IEBGENER, a program that copies or converts a sequential data set to a partitioned data set.
- IEBPTPCH, a program that prints or punches all or selected portions of a sequential data set, a partitioned data set, or specified members of a partitioned data set.
- IEBISAM, a program that copies, unloads, loads and prints indexed sequential data sets.
- IEBUPDAT, a program that modifies the symbolic library.
- IEBUPDTE, a program that incorporates source language modifications into sequential and partitioned data sets.
- IEBEDIT, a program that produces an edited input job stream data set from a master input job stream data set.
- IEBDG, a program that produces test data sets for use in program debugging procedures.

Independent utilities are executed outside and in support of IBM System/360 Operating System. They are:

- IBCDASDI, a program that initializes and assigns alternate tracks on direct access volumes.
- IBCDMPRS, a program that dumps and restores the data contents of a direct access volume.
- IBCRCVRP, a program that recovers data from a track on direct access storage, replaces defective records with data supplied by the user, and writes the composite data on an operative track of the original volume.

System Utility Programs

System utility programs are executed under the operating system in the problem program mode. These utilities treat data associated with the structure of the operating system. They are:

- IEHPROGM, a program that modifies control data in volume and catalog structures.
- IEHMOVE, a program that duplicates collections of data sets to provide backup copies or to rearrange existing ones.
- IEHLIST, a program that lists the catalog or a portion thereof, a volume table contents, and the directories of partitioned data sets.
- IEHIOSUP, a program that updates the transfer control (XCTL) tables contained within the I/O support routines OPEN, CLOSE, and EOV.
- IFCDIP00, a program that writes the SYS1.LOGREC data set in initialized format.
- IFCEREP0, a program that edits and prints environmental records from SYS1.LOGREC.
- IEHUCSLD, a program that loads the 2821 generator storage with user-supplied character images.
- IEHINITT, a program that creates volume labels on magnetic tape.
- IEHDASDR, a program that dumps, restores, and initializes direct access volumes.

The system utility programs IEHPROGM, IEHMOVE, IEHLIST, IEHIOSUP, IEHUCSLD, and IEHINITT use the queued sequential access method (QSAM) to read and write the SYSIN and/or SYSPRINT data sets or their (user-designated) equivalents. For these programs, SYSIN and SYSPRINT data sets also may have a blocking factor that is other than one.

AUXILIARY PARAMETERS

IEHPROGM, IEHMOVE, IEHLIST, IEHIOSUP, IEHUCSLD, IEHINITT, and IEHDASDR may be invoked by a problem program. In this case, the calling program provides the utility program with certain auxiliary parameters in main storage, as shown in Figure 1. If the utility program is invoked by job scheduler, only the pointer to the EXEC statement parameters is present.

DEVICE ALLOCATION AND VOLUME MOUNTING (IEHMVSSF AND IEHMVXSF)

IEHPROGM, IEHMOVE, and IEHLIST require that volumes be mounted dynamically. However, the serial numbers and device types of these volumes are not necessarily known to the user at the time the job is submitted. For example, in moving a group of data sets, the names of individual data sets in the group and their corresponding volume information are not known to the IEHMOVE program until it scans the catalog for the information. Once this information is known, data control blocks may be constructed within the program itself containing ddnames associated with units on which the appropriate volumes may be mounted, using the OPEN (type=J) routine.

In order to ensure that necessary volumes are mounted or mountable, two routines reside on LINKLIB:

- IEHMVSSF, which is used by IEHPROGM and IEHLIST.
- IEHMVXSF, which is used by IEHMOVE.

Each contains the control section IEHVOLMT. The difference between the two routines is that linkage to the first is via branch-and-link, whereas linkage to the second is via transfer control (XCTL).

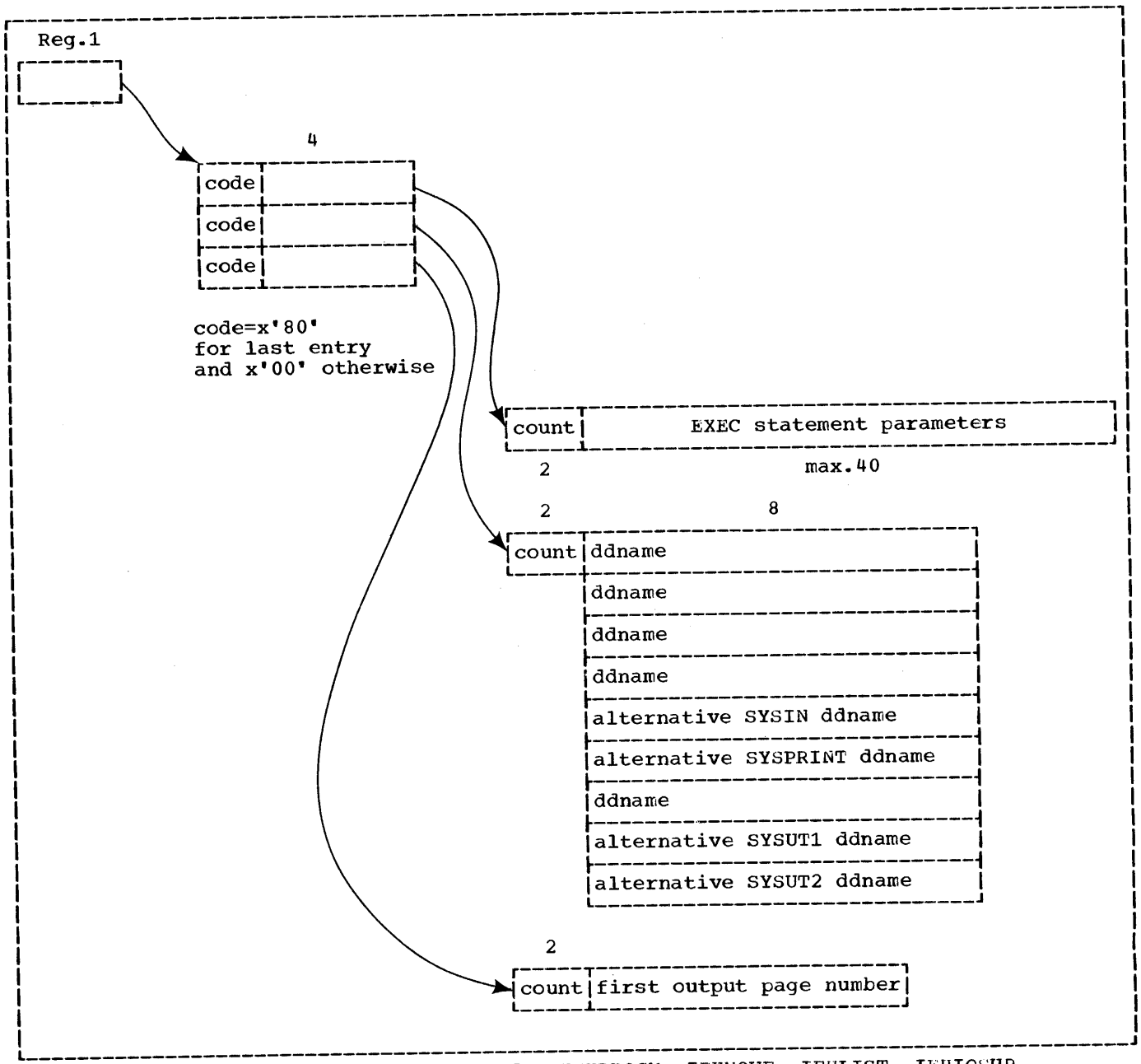
The logical flow of IEHVOLMT is shown in Chart 01. Figures 2 and 3 show the format of data supplied to IEHVOLMT by the calling routine. Figure 4 shows the format of an internal table maintained by IEHVOLMT in allocated main storage; the internal table is built once for each execution of IEHMOVE and IEHLIST, and once for each time IEHPROGM gives control to the volume mounter.

For each volume mounting request, IEHVOLMT returns to the calling routine a pointer to a ddname associated with a unit on which the desired volume may be mounted. The ddname is inserted into a field of the DCB and the desired volume is mounted by the open routine (type J). Actual mounting is accomplished by either IEHVOLMT or the calling routine, as indicated by Field 3 of the internal table header (Figure 2).

The essential processing in IEHVOLMT lies in the comparison of two masks: the first is obtained from the device mask table, using the device type supplied by the calling routine; the second is constructed by IEHVOLMT, using the UCBs allo-

cated to the current task.[1] In each mask, each bit represents a unit: in the first mask, an "on" condition means that the unit will accept the device type under consideration; in the second, an "on" condition means that the unit has been allocated to the current task. When both conditions occur for a given unit, IEHVOLMT checks to see if the desired volume is already mounted; if it is, an indication to that effect is returned. If the volume is not already mounted, the ddname associated with that unit (as found in the TIOT) can be used by the open routine (type J) to mount the desired volume on the allocated unit. As explained earlier, the open routine may be invoked by either IEHVOLMT or the calling routine.

[1]The UCBs are found as follows: location 16 in main storage points to the communications vector table, which in turn contains a pointer to a list of UCB pointers. The task I/O table (TIOT) is then used to distinguish the appropriate UCBs.



•Figure 1. Auxiliary Parameter Format for IEHPROGM, IEHMOVE, IEHLIST, IEHIOSUP, IEHUCSLD, IEHINITT, and IEHDASDR

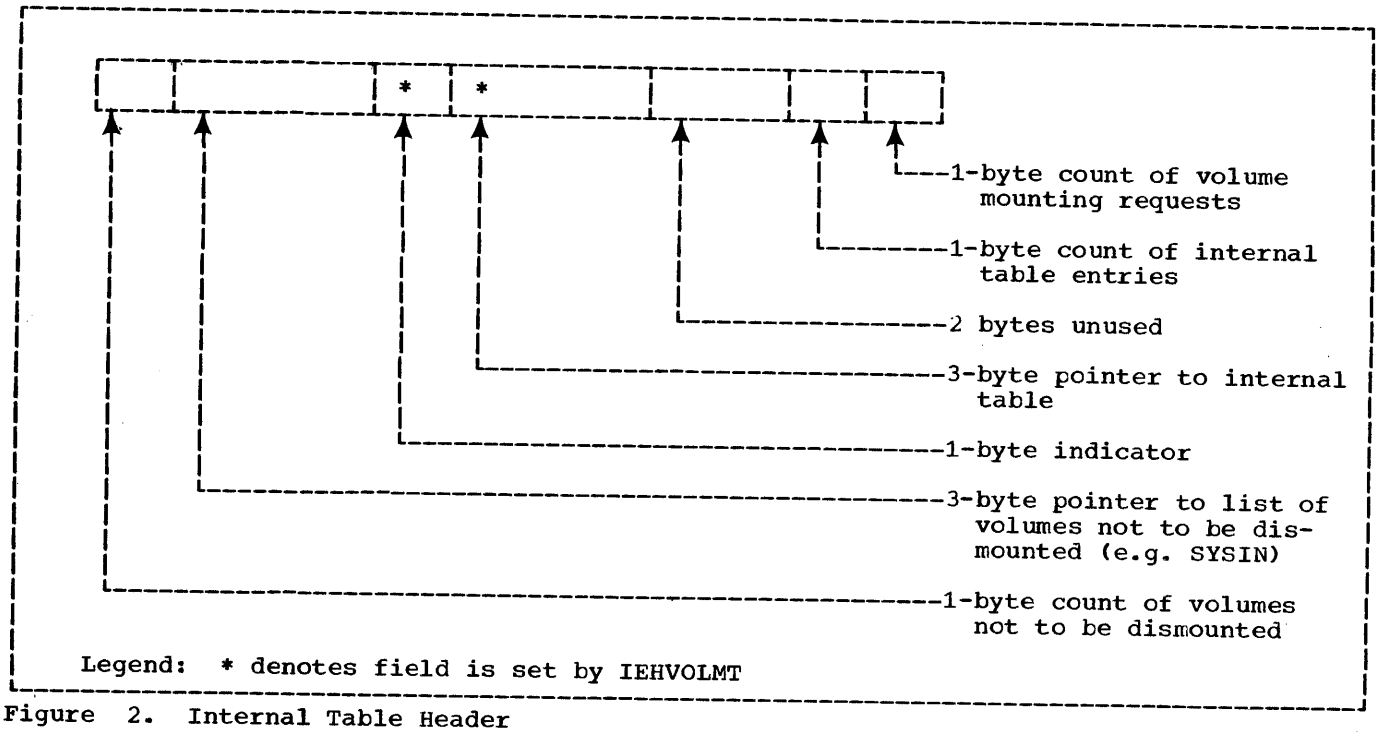


Figure 2. Internal Table Header

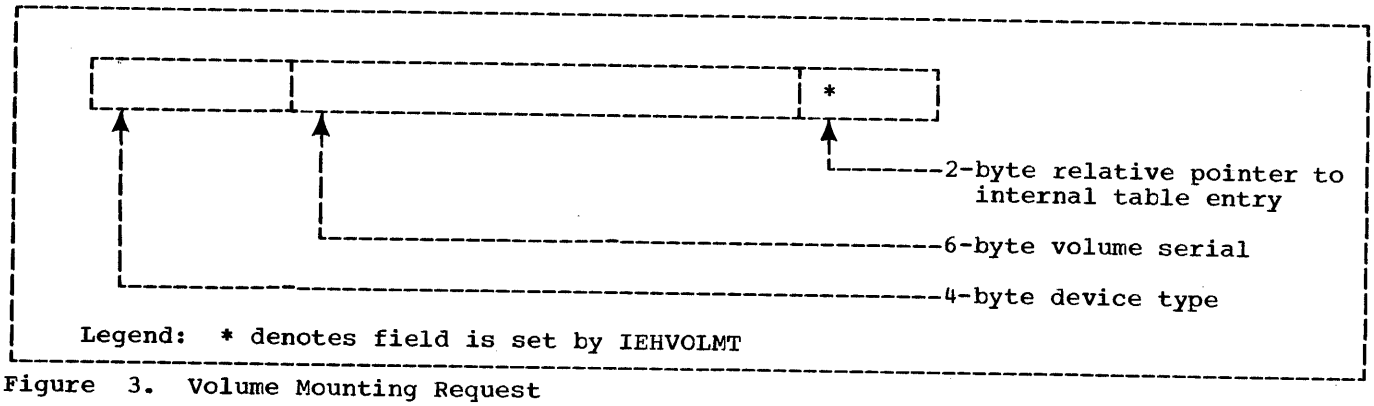


Figure 3. Volume Mounting Request

Indicator Settings: Field 3 of the internal table header (Figure 2) may have the following settings:

Bit	Value	Meaning
0	0	No mounting is to be done.
	1	Mounting is to be done.
1	0	No dismounting.
	1	Volume mounting requests having the high-order bit of the relative table address set to 1 are to be dismounted (the units made available).
2	0	Ignore old requests (ignore usage code, Figure 4).
	1	Old requests are valid.
3	0	Build internal table.
	1	Internal table is already built.
4		Unused.
5		Unused.
6-7	01	All volume mounting requests accomplished.
	10	No volume mounting requests accomplished.
	11	Some volume mounting requests accomplished: those volume mounting requests having a relative table address of zero were not accomplished.

CONTROL CARD SCANNER (RDCDRT)

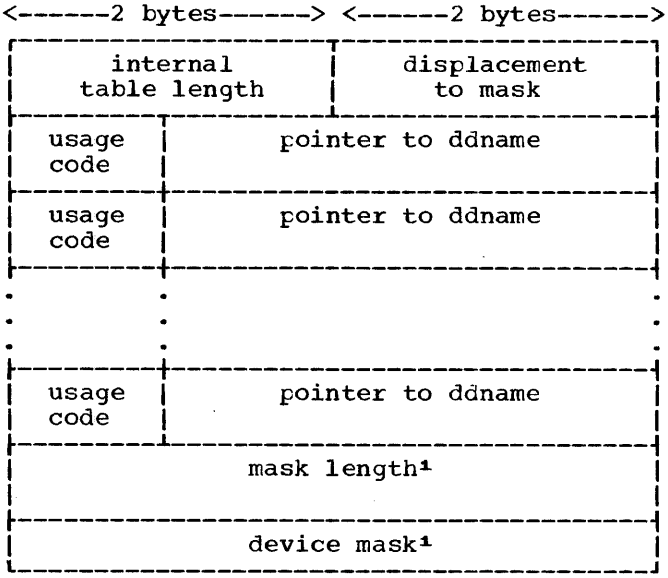
IEHMOVE and IEHLIST contain copies of a control card scan routine, RDCDRT. Each call to RDCDRT results in the return of the item scanned, together with an indication of the type (operation, keyword, or parameter). In IEHMOVE, the routine has the load module name IEHMVESJ; in IEHLIST, it has the name RDCDRT.

This routine reads control cards (using QSAM), checks syntax, and returns to the calling routine a command word, a keyword, or a parameter.

The calling routine supplies a 192-byte work area (on a fullword boundary) followed by the DCB for the control card data set. The calling routine must open this data set. The control card routine inserts the address of the end-of-file routine KEOF in the DCB.

The calling sequence for RDCDRT is as follows:

- Register 13 points to the first byte of the work area.
- Register 14 points to the return address in the calling routine.
- Register 15 points to the entry point RDCDRT.



[1]Device mask length, in bytes, is equal to the number of UCB pointers in the UCB pointer table.

Figure 4. Internal Table Maintained by IEHVOLMT

Upon returning control to the calling routine, the control card routine returns the following information:

- Register 1 points to the starting address of the item scanned.
- Register 2 contains the length of the item scanned.
- The first byte of the 192-byte work area is labeled SWITCHRD; its bits have the following meanings when set to 1:

Bit	Meaning
0	Syntax error
1	Bypass switch
2	End-of-file
3	Initial entry
4	Command word
5	Keyword
6	Parameter
7	Parameter or keyword delimited by a right parenthesis.

The control card routine contains the following subroutines:

RDCARD
: resets switches and saves registers 3-14.

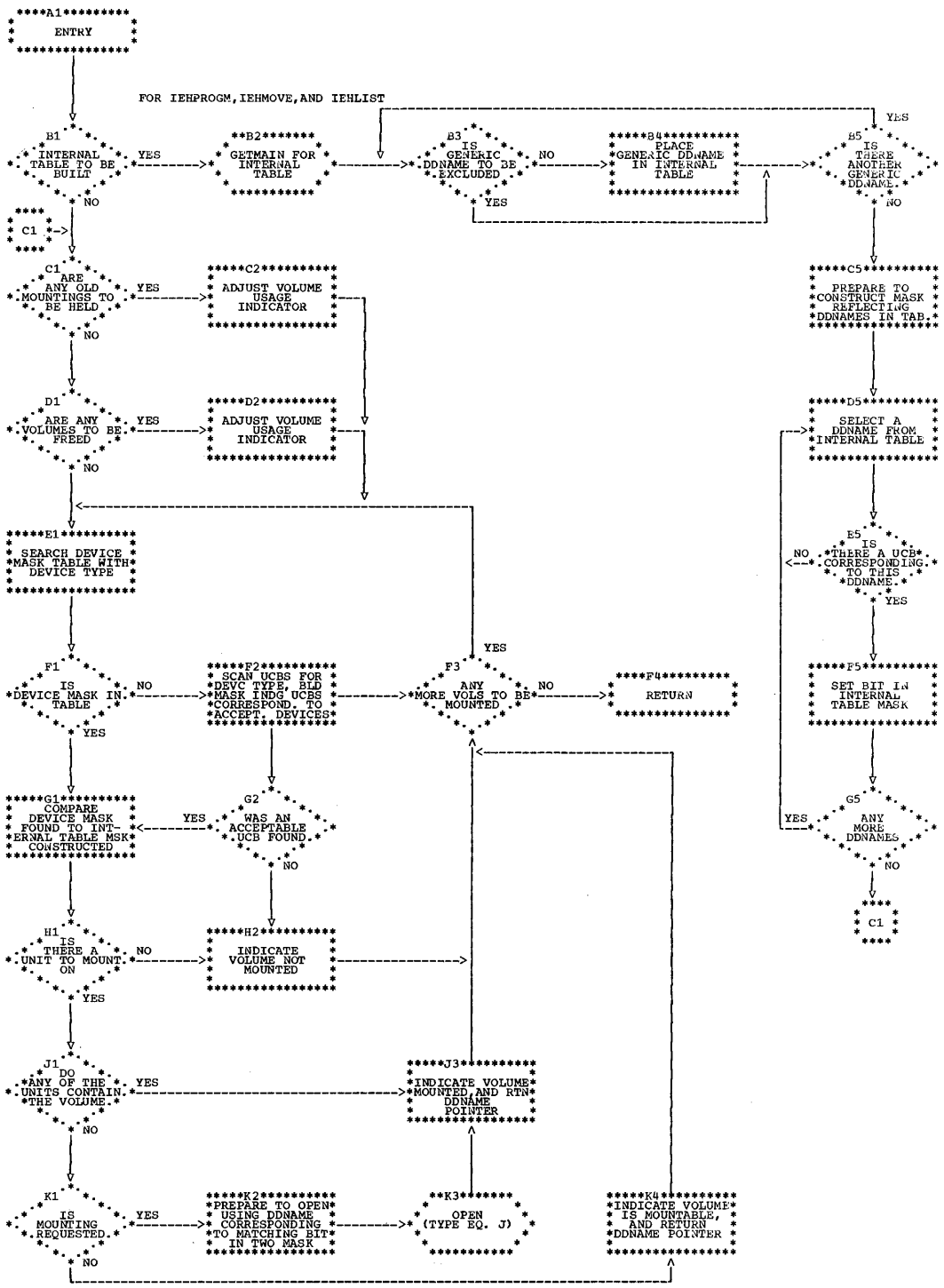
KGTCD
: reads a card into the work area, using QSAM.

KTRT
: saves the "start" address of the scan, and scans for a delimiter.

KPPARQ
: stores the address and length (of the item scanned) in registers 1 and 2.

KINVAL
: is entered when an invalid delimiter is found.

Chart 01. IEHVOLMT - Volume Mounting Logic



Modifying System Control Data (IEHPROGM)

The IEHPROGM program is a convenient interface between the user and data management routines which modify volume and catalog structures. By means of utility control statements, the user may request the IEHPROGM program to:

- Scratch, rename, catalog, or uncatalog a data set.
- Scratch or rename a PDS member.
- Scratch a data set assigned by the operating system.
- Build or delete an index level, index alias, or generation data group.
- Connect or release a control volume.

The general design of the program is shown in Figure 5. For each request listed above, the program issues one or more supervisor calls (SVCs) for data management routines which perform the requested service; the IEHPROGM program interfaces with these routines by building parameter lists for them, invoking them, and analyzing their returns.

Following the return from a data management routine, further processing by the IEHPROGM program may include additional calls to data management routines, as in the case, for instance, of supplying the catalog with index levels needed to catalog a data set by its specified, fully-qualified name.

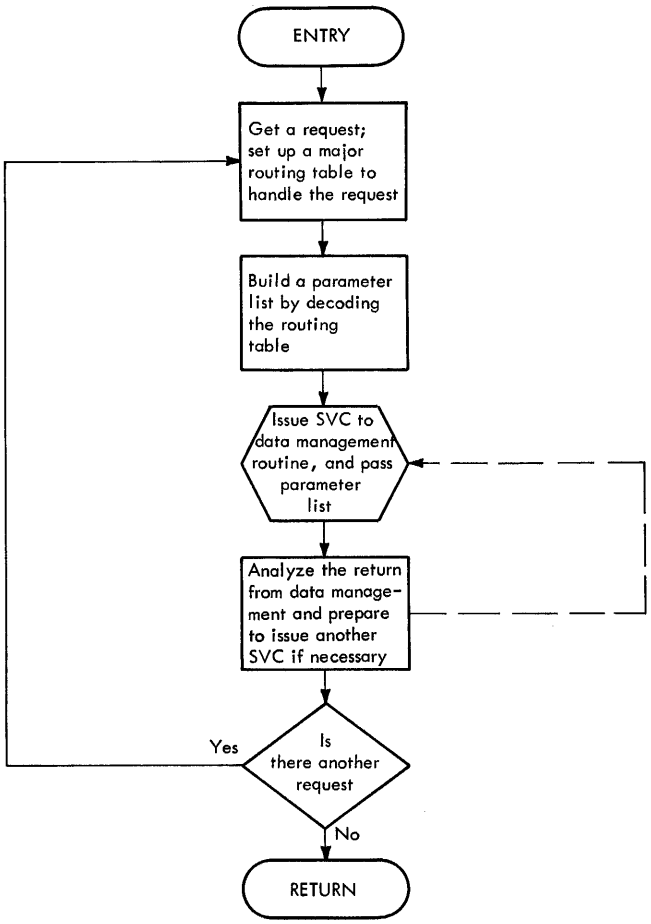


Figure 5. The General Design of the IEHPROGM Program

PROGRAM STRUCTURE

The program consists of seven load modules and a dynamically allocated work area:

- The Root resides in main storage throughout the program's execution. It contains V-type address constants needed by the overlay supervisor.
- The Parameter List Builder initializes the program, obtains and analyzes requests, and builds a parameter list for the appropriate data management routine.
- The Volume Mounter ensures that all volumes needed to service the request are mounted or mountable.
- The SVC Return Analyzer issues the appropriate SVC and analyzes its return. In some instances, additional SVCs may be issued by this module.
- The Auxiliary Parameter Analyzer analyzes auxiliary parameters supplied to the IEHPROGM program by a calling program, and also opens SYSIN and SYSPRINT.
- The Message Writer writes all diagnostic messages and operator instructions issued by the program.
- The Volume Look-up obtains volume information from the device name table when the keyword parameter VOL occurs in a request.
- The Work Area is obtained dynamically by the Parameter List Builder.

The overlay structure of the program is shown in Figure 6; each block represents a control section (CSECT).

The structural flow of the program is shown in Figure 7. The Auxiliary Parameter Analyzer, Message Writer, and Volume Lookup are subordinate load modules; the others are control load modules.

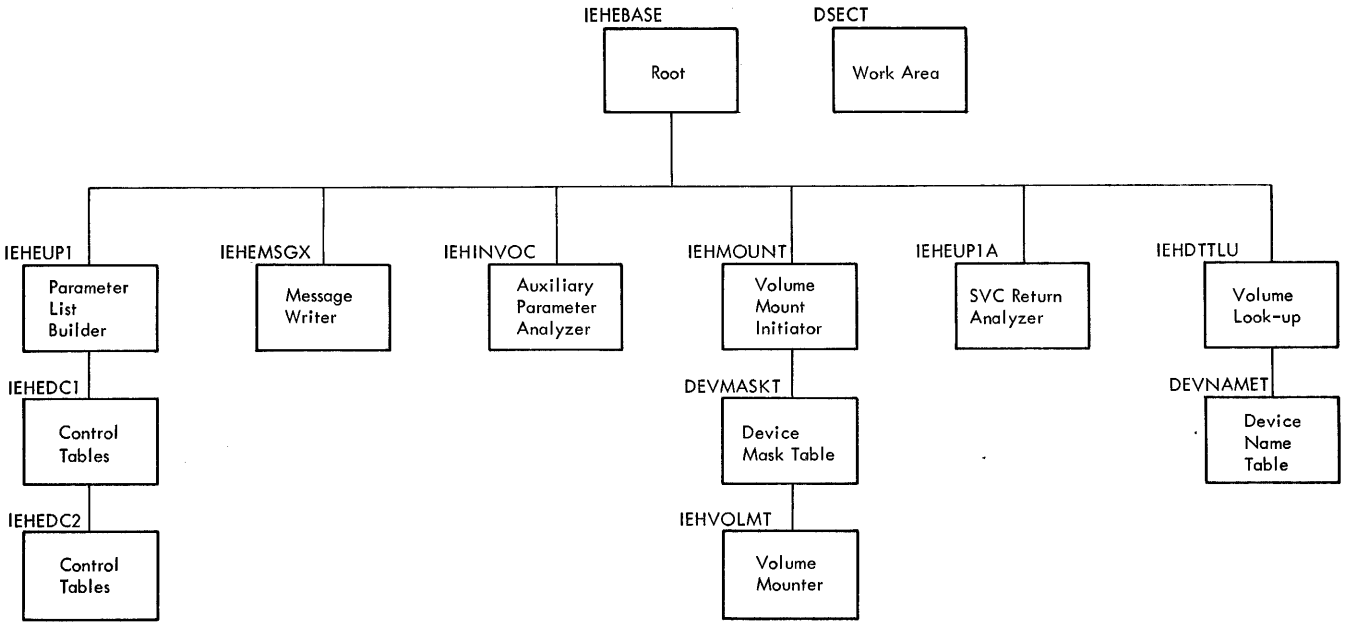


Figure 6. The Overlay Structure of the IEHPROGM Program (Each block represents one control section)

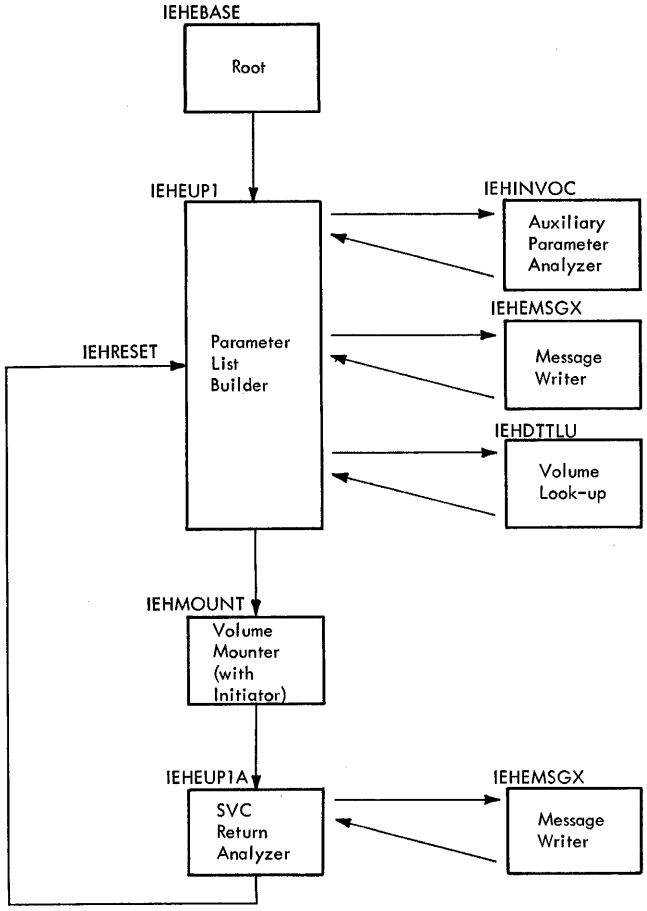


Figure 7. The Structural Flow of IEHPROGM Program (Each block represents one load module)

The linkage procedure used to execute a subordinate load module and shown in Figure 8 is as follows:

1. Module A loads the address (coded as a V-type address constant) of an entry point, B1, to module B in register 15; A branch-and-link instruction is then issued by module A, resulting in the following action:
   a. The address of the next sequential instruction is loaded into register 14, and
   b. A branch is made to the overlay supervisor.
2. The overlay supervisor then causes module B to be loaded and gives control to entry point B1. The return address in register 14 is now meaningless, since module B has overlaid module A.
3. Module B returns control (indirectly) to module A by loading the address of an entry point to module A in register 15 and branching to it, resulting in a branch to the overlay supervisor.
4. The overlay supervisor then causes module A to be loaded and gives control to entry point A1. Since module A has now overlaid module B, the return address in register 14 is re-established.
5. The instruction at entry point A1 gives control to the instruction at the return address.

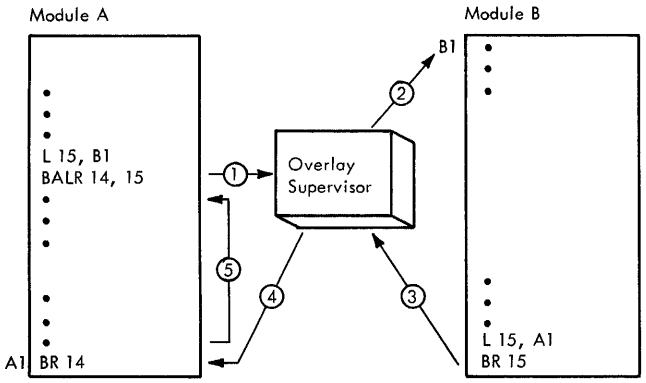


Figure 8. Linkage Procedure Used by the IEHPROGM Program to Invoke a Subordinate Load Module

CONTROL LOAD MODULES

The Root, the Parameter List Builder, the Volume Mounter, and the SVC Return Analyzer are control load modules.

The Root (IEHEBASE)

The Root consists of one CSECT, IEHEBASE. It contains V-type address constants needed by the overlay supervisor, and a branch instruction to the Parameter List Builder.

The Parameter List Builder (IEHEUP1, IEHEDC1, IEHEDC2)

The Parameter List Builder contains three CSECTS: IEHEUP1, IEHEDC1, and IEHEDC2.

IEHEUP1
builds the parameter list for the initial SVC to the appropriate data management routine. It contains seven routines: COMMENCE, IEHRESET, FNDECODE, FDLD, KODECODE, IEHESCAN, and IEHETLU. The parameter list (Figure 11) is built at location IEHEMAC1 in the work area.

COMMENCE
initializes the program by establishing addressability and obtaining a work area of 4416 bytes in main storage.

IEHRESET
initializes for a new request by resetting switches.

FNDECODE
determines the operation (e.g. SCRATCH) requested, and stores the address of its routing table at location FINUSE in the work area. The use and format of the routing tables are discussed under "Program Flow."

FDLD
decodes the routing table in use. Each routing table indicates a sequence of operations to be performed; FDLD effects these operations by decoding the routing table.

KODECODE
causes keyword parameters to be scanned from a utility control statement, and successively directs control to subroutines which move parameter data to the parameter list for the data management SVC. The list is at location IEHMAC1 in the work area.

IEHESCAN
scans a control statement for an operation or an operand.

IEHETLU
performs a table look-up for the address of a routine or a routing table: if the search argument used is an operation, a routing table address is retrieved; if the search argument used is an operand, a routine address is retrieved.

IEHEDC1
contains seven tables: TABLEN3, TABLEN4, TABLEN5, TABLEN6, TABLEN7, CATALOG, and UNCATLG.

Each of the tables TABLENn consists of a variable number of entries: the first n bytes of each entry is an operation or keyword operand, and the last four bytes of each entry is the address of a routing table (for an operation) or a routine (for a keyword operand). As an example, TABLEN6 contains as typical entries

```
C'RENAME'    AL4(RENAME)
C'DSNAME'    AL4(DSNAME)
```

where RENAME is the symbolic location of the RENAME routing table, and DSNAME is the symbolic location of the routine given control when the data set name is scanned.

The tables CATALOG and UNCATLG are major routing tables for the catalog and uncatalog operations.

IEHEDC2
contains the major routing tables DELETEX, CONNECT, RELEASE, BUILDA, DELETEA, SCRATCH, BLDX, and RENAME. The use and format of all routing tables are discussed under "Program Flow."

The Volume Mounter (IEHMOUNT, IEHVOLMT, DEVMASKT)

This segment ensures that all volumes needed by the data management routine are mounted or mountable. Three CSECTs are present: IEHMOUNT, IEHVOLMT, and DEVMASKT.

IEHMOUNT
is entered from IEHEUP1 when the parameter list for the initial data management SVC has been built. IEHMOUNT then calls IEHVOLMT, if necessary, to ensure that a needed volume is already mounted or is mountable. IEHVOLMT is called in the following cases:

1. For a SCRATCH or RENAME request, the volume ID from the control statement is passed to IEHVOLMT, together with an indication that the volume is not to be mounted. (The scratch and rename data management routines themselves perform volume mounting.) The call to IEHVOLMT serves as a check that the volume is mountable.
2. For any other request, if CVOL is specified, IEHVOLMT is passed the volume ID, together with an indication that the volume is to be mounted. IEHVOLMT is not called otherwise.

For cases (1) and (2) above, IEHVOLMT normally returns a pointer to a ddname associated with a channel-unit on which the volume is mounted or can be mounted. IEHMOUNT then inserts the ddname into the data control block (DCB) needed to perform I/O operations on the volume. Control is then given to IEHEUP1A.

If the volume is not mountable, as indicated by the return from IEHVOLMT, IEHMOUNT aborts the request, giving control to IEHEUP1 to honor the next request.

IEHVOLMT and DEVMASKT are discussed under the heading "Volume Mounting and Device Allocation."

The SVC Return Analyzer (IEHEUP1A)

This segment issues, and analyzes the returns of, all data management SVCs used by IEHPROGM to perform a requested operation. The SVC Return Analyzer segment consists of a single control section, IEHEUP1A, which contains ten routines: LATAB, FDLD, OPENVTOC, GETADSCB, SVCRET, SVC26RET, INDEX8, NEEDINDX, SCANIT, and VTOCRET.

LATAB
stores the address of the routing table in use at location FINUSE in the work area.

FDLD
decodes the routing table, directing control to the routines indicated by the table. (This is the same FDLD present in IEHEUP1, the Parameter List Builder.)

OPENVTOC
constructs a DCB for the VTOC to be opened for a Scratch VTOC request, and then opens the VTOC.

GETADSCB
reads the VTOC and scratches the following DSCBs for a Scratch VTOC request:
1. If the SYS keyword was specified, each system-assigned data set (a data set having a name beginning with the 36 characters AAAAAAAA.AAAAAAAA.AAAAAAAA.AAAAAAAA.) is scratched.
2. If the SYS keyword was not specified, each Format 1 DSCB is scratched.

The VTOC is closed when an EOF is detected in reading it.

SVCRET
interprets the returns from the scratch (SVC 29) and rename (SVC 30) routines. The return is used as an indexing factor on the branch table BRANTAB, giving control to an appropriate diagnostic routine.

SVC26RET
interprets returns from the catalog and index (SVC 26) routines. SVC26RET uses BRANTAB in the same manner as SVCRET; the difference between the two routines is that SVC26RET also interprets the return from the locate routine (locate is used by both catalog and index).

INDEX8
gives control to SVC26RET to interpret an error return from the Locate routine.

NEEDINDX
is entered the first time SVC26RET detects that an index level supplied in a utility control statement was not found in the catalog. NEEDINDX passes the index name to SCANIT.

SCANIT
constructs a parameter list for an SVC to the index routine each time a return shows that an index level was absent.

VTOCRET
effects the writing of diagnostic messages following a return from the scratch routine on a SCRATCH VTOC request.

SUBORDINATE LOAD MODULES

The subordinate load modules of the program are the Auxiliary Parameter Analyzer, the Message Writer, and the Volume Look-up.

The Auxiliary Parameter Analyzer (IEHINVOC)

The Auxiliary Parameter Analyzer (IEHINVOC) analyzes any auxiliary parameters passed to the IEHPROGM program by a calling program and moves the DCBs for the data sets SYSIN and SYSPRINT (or their substitutes) to the work area and opens them.

The Message Writer (IEHEMSGX)

The Message Writer (IEHEMSGX) writes all messages issued by the program. Messages are written on the SYSPRINT data set unless a calling program specifies otherwise; console messages are written using the job management WTO routine.

Input to the message writer consists of a full word in register 0:

Byte 0 is unused.

Byte 1 Bits 0-5 are unused.

Bit 6 is set to one if the message to be written is to be placed at the next available location in the output buffer; otherwise it is set to zero.

Bit 7 is set to zero if the message is to be written during the current execution of the message writer, and to one otherwise.

Byte 2 contains the relative position in the buffer of the message.

Byte 3 contains the message number.

Volume Look-up (IEHDTTLU, DEVNAMET)

This segment obtains volume information for use by IEHVOLMT and data management. It is called only when information specified by the VOL keyword is encountered by IEHESCAN. The segment consists of two CSECTs: IEHDTTLU, a table look-up, and DEVNAMET, the device name table maintained at the installation. At the time IEHDTTLU is entered, register 2 points to a 28-byte field in the work area consisting of the following subfields:

1. (Bytes 0-7) contain the value supplied to the keyword operand VOL (left-justified and padded with blanks), as scanned from a utility control statement.
2. (Bytes 8-11) are initially blank.
3. (Bytes 12-15) contain the return address (coded as a V-type address constant).
4. (bytes 16-27) are initially blank.

IEHDTTLU then stores registers 3-5 in subfield 4, and performs a table look-up in DEVNAMET, using subfield 1 as a search argument. The table value of the argument is moved to subfield 2, and control is returned to the location specified by subfield 3.

Subfield 2, the table value of the search argument, is of the following format:

Byte 8 contains a bit configuration used by the I/O Supervisor.

Byte 9 contains a device option code.

Byte 10 contains a device class code.

Byte 11 contains a device type code.

PROGRAM FLOW

The logical flow of the program proceeds in three phases:

- Phase 1 (Chart 02), during which a major routing table is established for the operation to be performed.
- Phase 2 (Chart 03), during which the major routing table is decoded, causing the parameter list to be built for the SVC to a data management routine, and causing appropriate volumes to be mounted. The parameter list (Figure 11) is built at location IEHEMAC1 in the work area.
- Phase 3 (Chart 04), during which the initial SVC is issued, its return analyzed, and any additional SVCs issued in order to complete the request.

Phase 1

The program receives control of the CPU when the keyword operand PGM=IEHPROGM is encountered in an EXEC statement, or when the program is invoked by a calling pro-

gram. The Root segment immediately gives control to the Parameter List Builder segment. The registers are saved and the work area is obtained. Any auxiliary parameters are analyzed, and SYSIN and SYSPRINT are opened. A new request is initialized for, and then obtained (using BSAM).

FNDECODE then directs control to IEHESCAN to scan the operation name (e.g., SCRATCH) from the utility control statement image. IEHETLU then uses the name thus obtained to look up the address of the major routing table corresponding to the given operation. FNDECODE then places this address at location FINUSE in the work area, for use in Phase 2.

Communication between FNDECODE, IEHESCAN, and IEHETLU is effected through the use of the communication table IEHECHAR (in the work area), shown in Figure 9. IEHECHAR is also used in Phase 2 in scanning keyword operands.

Phase 2

This phase of the program decodes the major routing table established during Phase 1. Decoding of the routing table results in the scanning of keyword operands, the volume look-up, the building of the parameter list for the data management routine to be used, and the mounting of volumes.

Major routing tables appear in CSECTs IEHEDC1 and IEHEDC2 in the Parameter List Builder segment. There is one major routing table for each operation of the IEHPROGM program; the symbolic name of the routing table is the same as the name of the operation it supports. Each major routing table consists of a variable number of entries; the first byte of each entry contains a routine code, and the remaining bytes of the entry contain information useful to the routine. Figure 10 shows the format of the routing table for the Catalog operation.

As indicated on Chart 03, FDLD decodes the successive entries of the major routing table in use, directing control to the indicated routines.

Note: It is possible for the sequence of operations indicated by a routing table to be interrupted, possibly even broken. For example, if a syntax error on a control statement is encountered, the address of the major routing table will be replaced by the address of a routing table which will effect the writing of appropriate messages. FDLD would then decode this table, causing IEHEMSGX to write the selected messages. The last entry in tables of this nature will then either cause the address of the original major routing table to be restored, or will cause the request to be aborted, depending on the severity of the situation.

The scanning of keyword operands takes place when a routine code of hexadecimal 01 is encountered. KODECODE directs control to IEHESCAN to scan a keyword or keyword value. If VOL is present as a keyword, the value supplied to it in the control statement is passed to IEHDTTLU, the Volume Look-up, and the device information retrieved from the look-up is saved for the

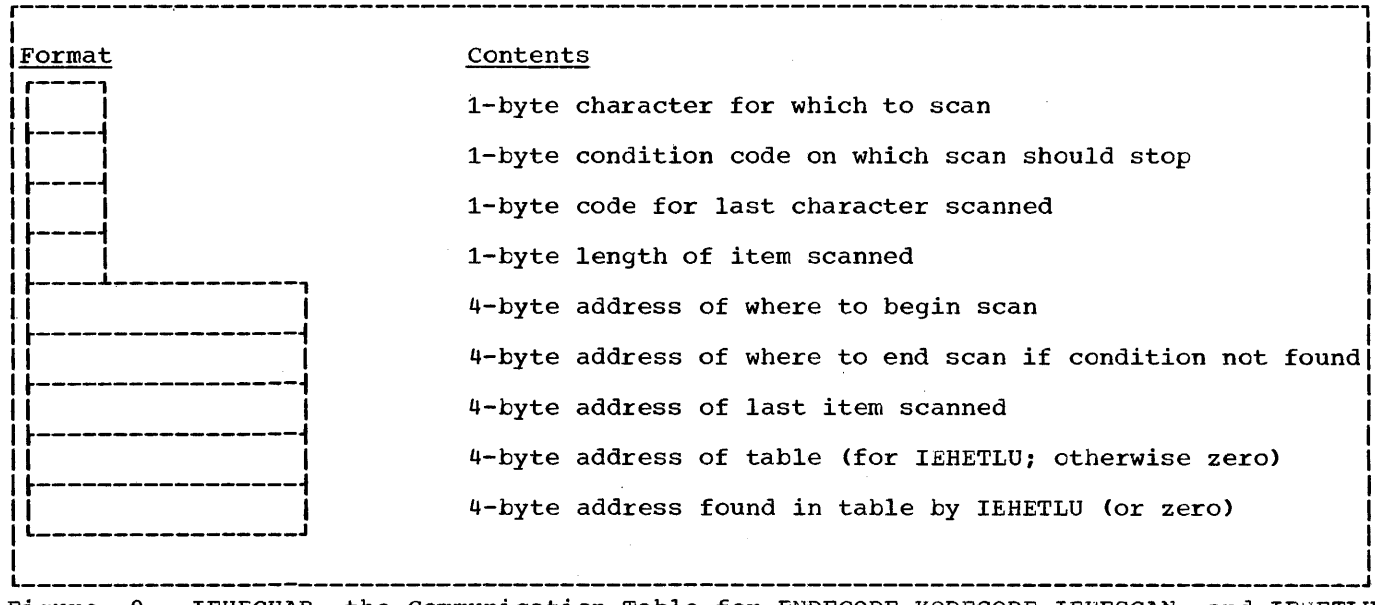


Figure 9. IEHECHAR, the Communication Table for FNDECODE,KODECODE,IEHESCAN, and IEHETLU

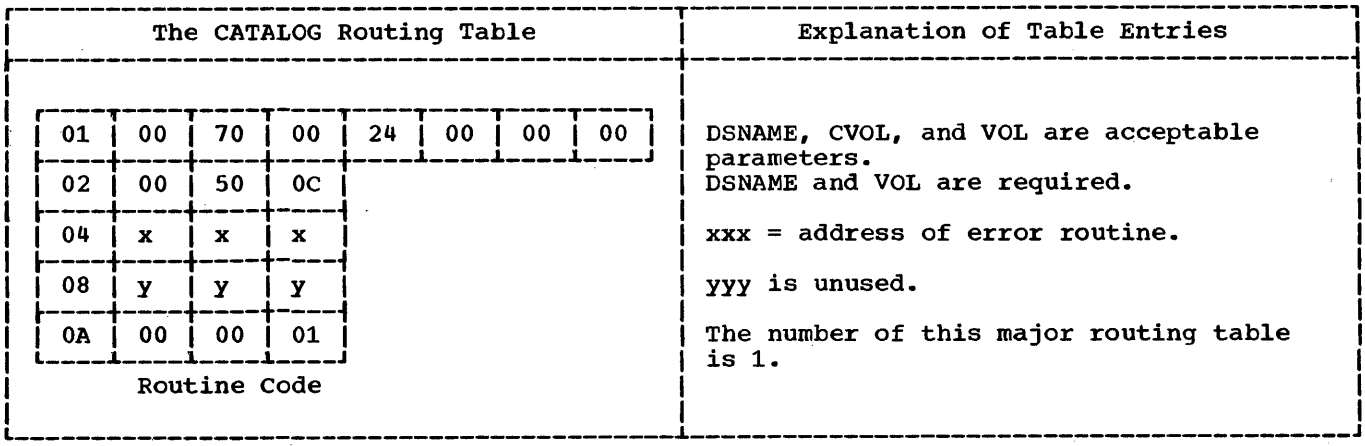


Figure 10. The CATALOG Routing Table

Volume Mounter and data management routines. When a value supplied to another keyword is detected, a look-up is performed by IEHETLU, using the keyword (e.g., DSNAME), to retrieve the address of a keyword routine, which is then given control. (The symbolic address of a keyword routine is identical to the keyword name. For example, the symbolic address of the DSNAME routine is DSNAME.) The keyword routine then enters the keyword value information into the parameter list for the data management routine. Control is then given back to IEHESCAN to scan the next keyword parameter, and the cycle continues until an EOF condition is detected in reading SYSIN (or its substitute).

When the control statement has been scanned, control is given back to FDLD to decode the next entry in the major routing table. Successive entries in the major routing table may test for the presence of minimum allowable parameters (TESTDUP), establish a temporary routing table to write messages (LINKSAVE), restore the major routing table (DCRETURN), or read and log the remaining cards (READALL). If an error has been found in the utility control statement, the request is aborted, and control is given to IEHRESET to honor the next request.

If no error has been found, the last entry of the major routing table will have been decoded. The last entry of each major routing table causes FDLD to give control to GETAVOL, which places the number of the major routing table at location FINUSE in the work area, and then gives control to the Volume Mounter. If any required volume is not mounted or mountable, control is given to IEHRESET; otherwise, control is given to the SVC Return Analyzer (IEHEUPIA), and Phase 3 is entered.

At the completion of Phase 2, the parameter list for the initial SVC to a data management routine has been built at location IEHEMAC1 in the work area. The formats of the various parameter lists for the data management routines are shown in Figure 11.

Phase 3

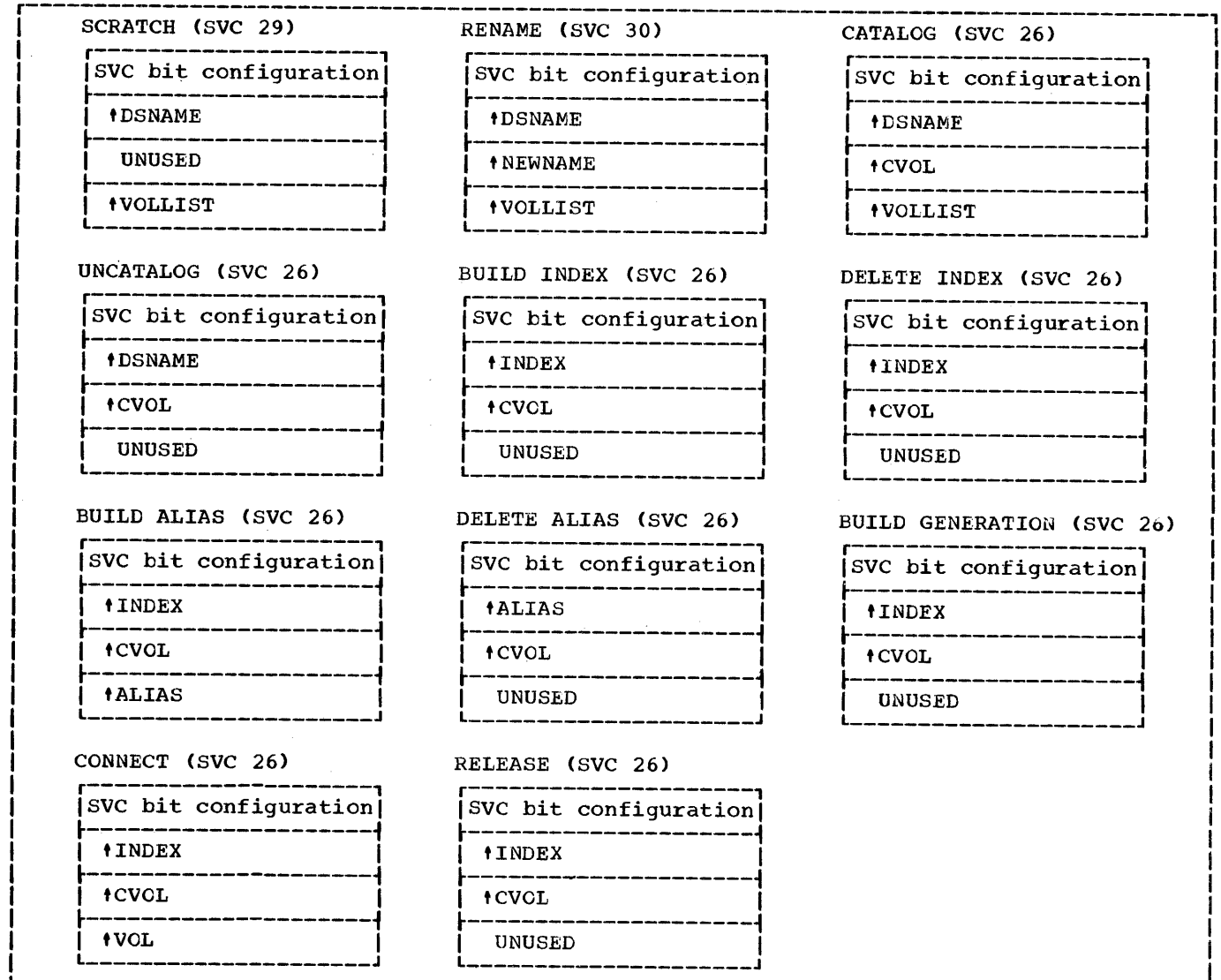
Phase 3 issues, and analyzes the returns of, all SVCs to data management routines used to accomplish IEHPROGM functions. At the time Phase 3 is entered, the parameter list for the initial SVC has been built, and all volumes are mounted or mountable.

Using the routing table number placed at FINUSE (in the work area) during Phase 2 by routine GETAVOL, routine LATAB replaces the routing table number with the address of a carry-over routing table, to be decoded by FDLD. If the current request is a SCRATCH VTOC request, the carry-over routing table will cause itself to be replaced with the routing table VTOCDCS. Otherwise, the carry-over routing table will simply cause the SVC to be issued, by means of the entry

```
X'00'        AL3(SVC instruction)
```

FDLD then decodes the carry-over routing table, establishing the VTOCDCS routing table only if the request is to scratch a VTOC.

SVCRET and SVC26RET decode the return from the data management routine used, directing control through the branch table BRANTAB to diagnostic routines (NEEDINDX, SCANIT, INDEX8), or to message-effecting routines.



Notes:

- At time of SVC, register 1 contains the address of the parameter list.
- Each parameter list is constructed at IEHEMAC1 in DSECT, the work area.
- The addresses in the above parameter lists point to the following items:

DSNAME	44-byte data set name
NEWNAME	44-byte new name of data set
INDEX	44-byte index name
ALIAS	8-byte alias
VOL	6-byte volume ID
CVOL	6-byte volume ID
VOLLIST	2-byte count followed by a variable number of fields of the following format:

4-byte table value from DEVNAMET
6-byte volume ID
2-byte sequence number (for taped volumes)

Figure 11. Parameter Lists Built by IEHPROGM for Data Management Routines

Control is directed in the following way:

1. The return code (in register 15) is used by routine SVCRET or SVC26RET (depending on the SVC) as an indexing factor to retrieve a code number from the current entry of the active routing table. (The current entry is the entry following that entry designating the SVC.)

2. The code number thus retrieved is used as an indexing factor to give control to the appropriate IEHPROGM diagnostic routine to treat the type of return. Control is given to a diagnostic routine by means of a branch table, BRANTAB.

Example: Figure 12 shows the return-indexing entry of the CATALOG routing table. This entry is at location CATALOG+28, immediately following the SVC entry. Following the return of control to the program from the catalog routine (SVC 26), routine SVC26RET would use the contents of register 15 to retrieve a code number from this return-indexing entry. If the return code from the catalog routine were 28, for example, a code number of eight would be retrieved. Routine SVC26RET would then give control (via the branch table BRANTAB) to the diagnostic routine indicated by a code number of eight.

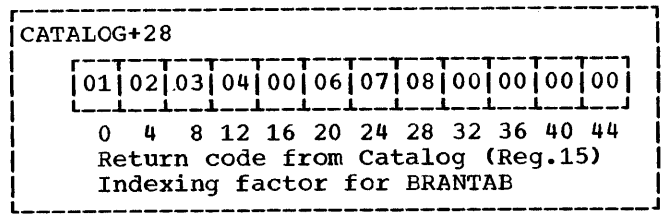


Figure 12. The Return-Indexing Entry (for the Catalog SVC) of the Catalog Operation

Chart 02. IEHPROGM Phase 1 - Modifying System Control Data

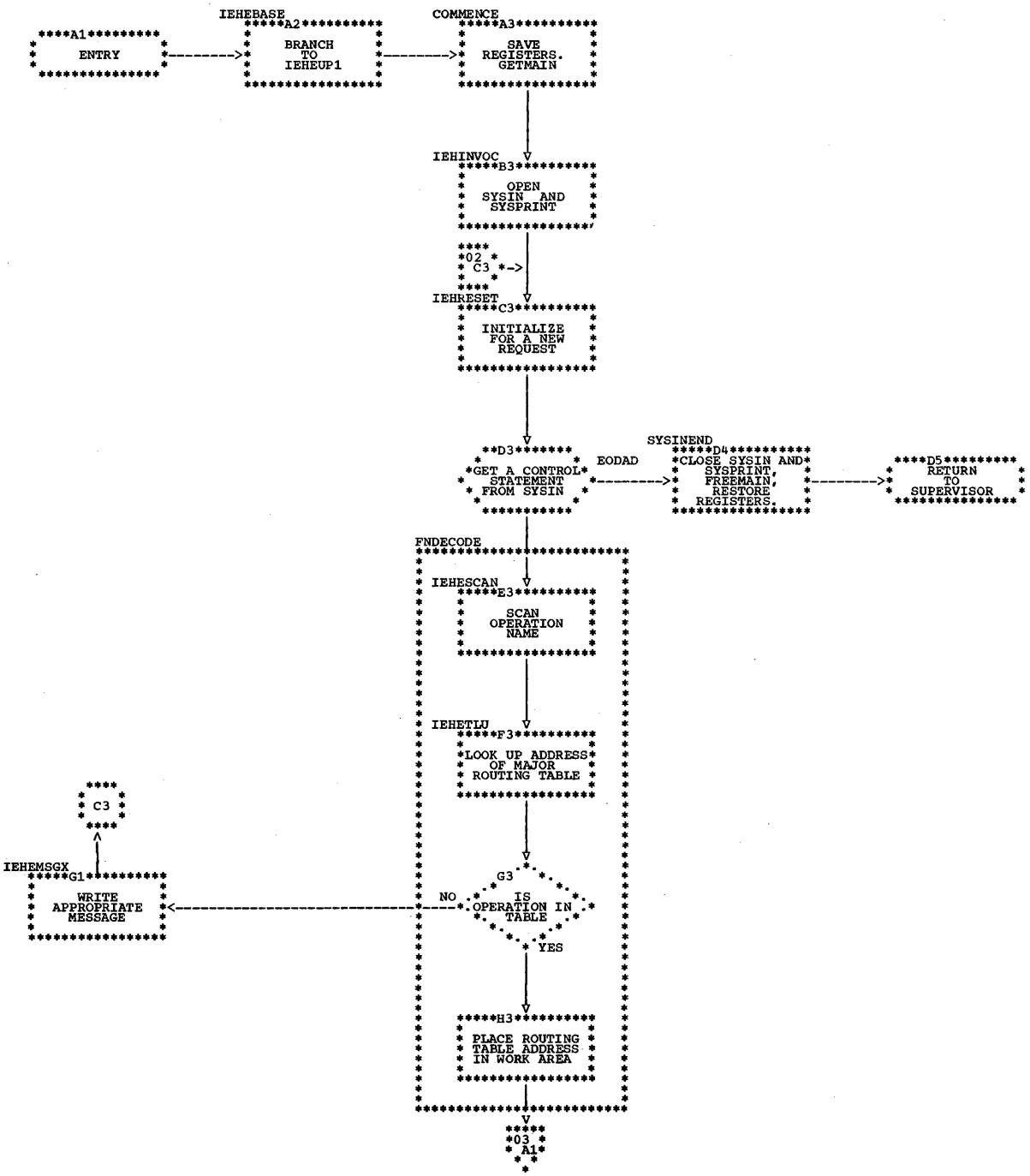


Chart 03. IEHPROGM Phase 2 - Modifying System Control Data

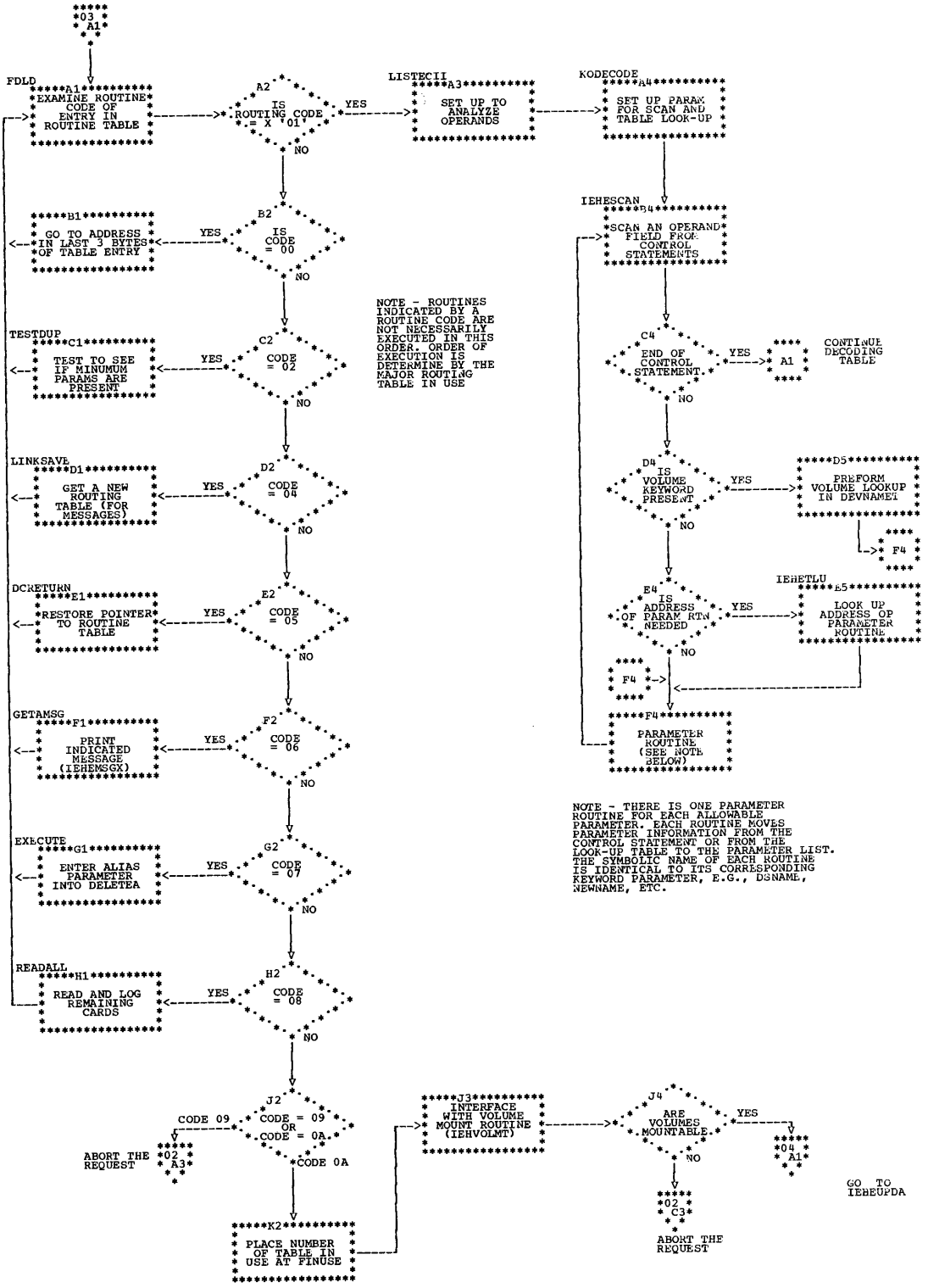
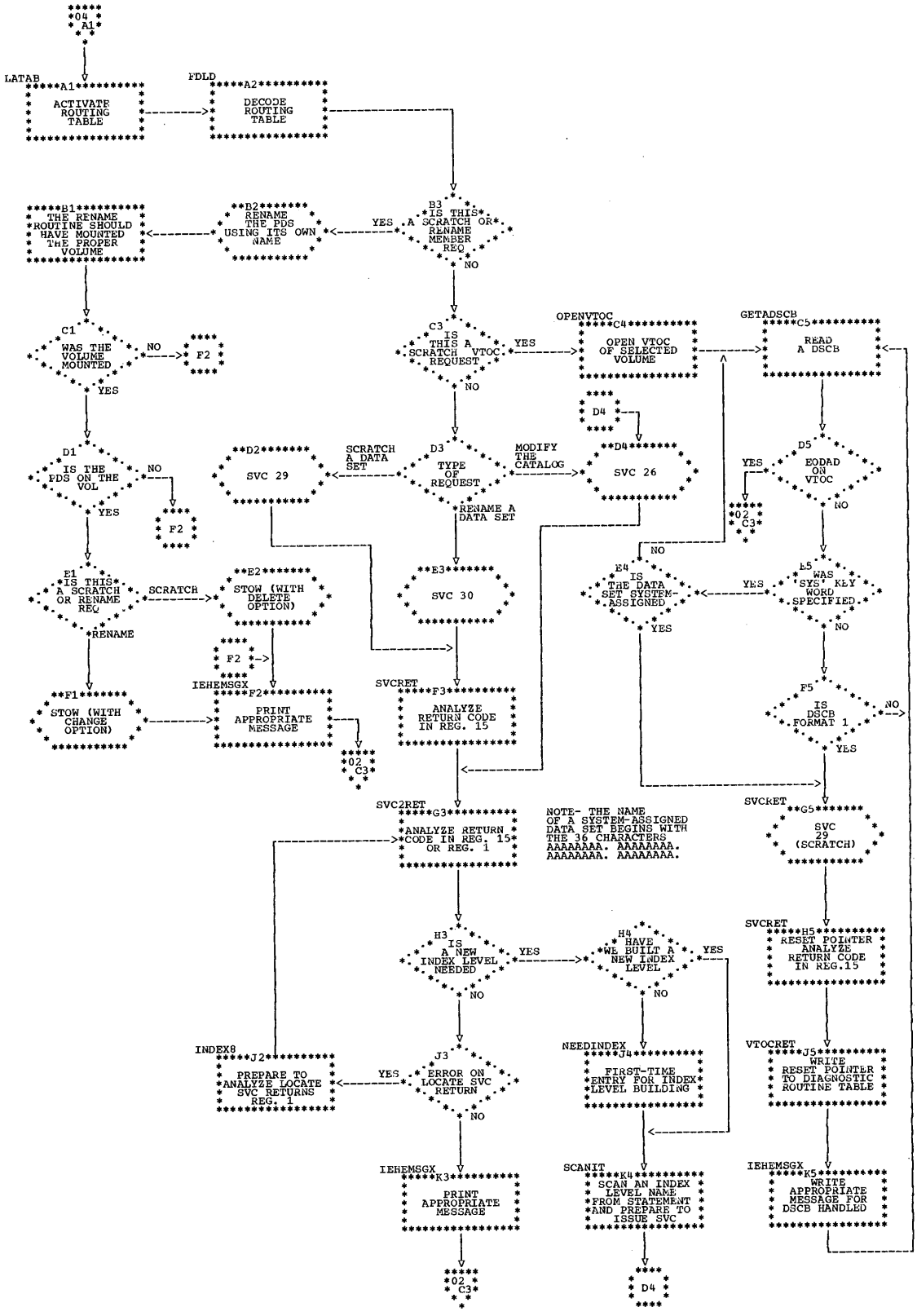


Chart 04. IEHPROGM Phase 3 - Modifying System Control Data



Moving and Copying Data (IEHMOVE)

The IEHMOVE utility program reproduces one or more data sets. The move operation relocates a collection of data and scratches the source data; the copy operation produces a replica of the source data, and leaves the source data intact. Throughout this discussion, the word "copy" will refer to both the MOVE operation and the COPY operation.

The program is serially reusable. It copies the following collections of data:

- A data set
- A volume
- A group of data sets related by a catalog
- A catalog

Depending on the compatibility of the source and receiving volumes, the movability of the source data set, and the allocation of space on the receiving volume, an attempt to copy a data set may result in an "unloaded" version of the data set. This version of the data set is in a format recognizable to the IEHMOVE program, but is not directly usable by other programs. An attempt to copy an unloaded data set onto a volume that would have originally supported a successful operation results in the "loading" of the unloaded data set, that is, the reconstruction of the original data set.

If a user has requested the processing of input/output header/trailer labels, this program will handle the direct moving or copying of such labels as they exist on the data sets to be moved or copied.

OVERALL FLOW

Figure 13 shows the design of the IEHMOVE program; each of the smaller blocks represents a load module, while each of the larger blocks represents a grouping of load modules by function:

- Program Set-up
- Request Set-up
- Message Writing
- Data Set Group (DSGROUP) Set-up
- Data Set and Volume Set-up
- Partitioned Data Set (PDS) Subroutines
- Copying, Unloading, and Loading
- Data Set Wrap-up
- DSGROUP Wrap-up

Charts 05 through 10 show the logical flow of the program as follows:

Overall Flow	Chart 05
DSGROUP Logic	Chart 06
VOLUME Logic	Chart 07
PDS Logic	Chart 08
DSNAME Logic	Chart 09
CATALOG Logic	Chart 10

Control is passed between loads almost always by means of Transfer Control (XCTL), with the following exceptions:

1. The stem, IEHMOVE, links to IEHMVXSE. The corresponding return to the stem is issued at the conclusion of the program by IEHMVESK.

2. The message writer, IEHMVESU, is always linked to.

3. IEHMVESR, a PDS subroutine which retrieves directory entries from a work data set, is always linked to.

PROGRAM STRUCTURE

Program Set-up (IEHMOVE, IEHMVXSE, IEHMVXSF)

The function of initializing the program for a job is performed by three loads: IEHMOVE, IEHMVXSE, and IEHMVXSF.

IEHMOVE
: is the stem, which is present during the entire execution of the program. It obtains main storage for a work area (IEHMVV) to be used by the rest of the program.

IEHMVXSE
: allocates space for the work data sets and opens them, clears the work area, and sets up an initial call to IEHMVXSF.

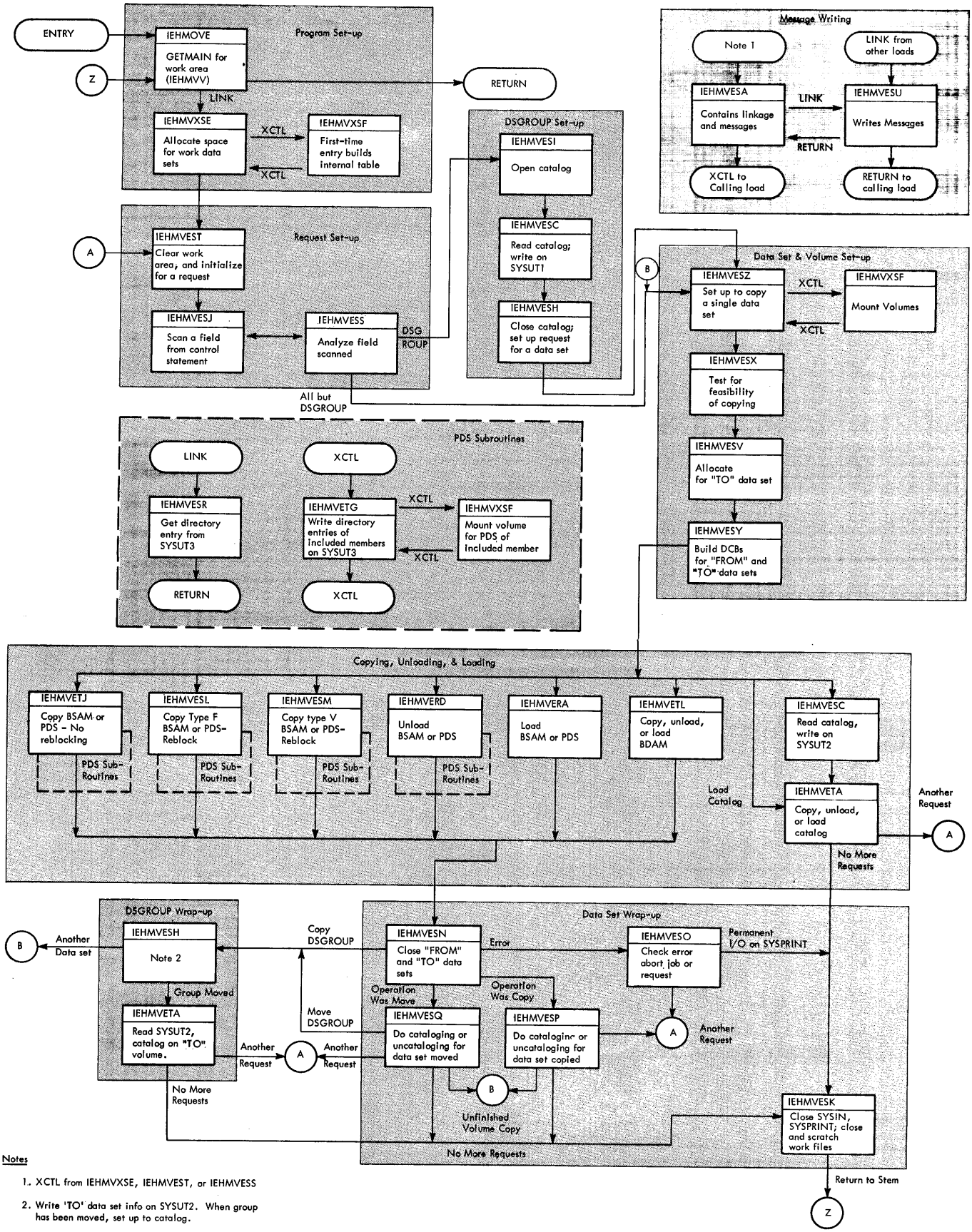
IEHMVXSF
: is the volume mounter, described under the heading, "Device Allocation and Volume Mounting." The first-time entry of this routine builds the internal table used by the volume mounter.

Request Set-up (IEHMVEST, IEHMVESJ, IEHMVESS)

The program is initialized to handle a single request by three loads: IEHMVEST, IEHMVESJ, and IEHMVESS.

IEHMVEST
: initializes the work area for a request and sets up an initial call to IEHMVESJ.

IEHMVESJ
: is the control statement scanner, described under the heading "Control Card Scanner."



Notes

1. XCTL from IEHMVXSE, IEHMVEST, or IEHMVESS
2. Write 'TO' data set info on SYSUT2. When group has been moved, set up to catalog.

Figure 13. The Design of the IEHMCVE Program

IEHMVESS
analyzes the information scanned by IEHMVESJ. For a PDS request, IEHMVESS writes the following information:

- Member names to be included or to replace on SYSUT1 (the format is shown in Figure 14).
- Member names to be excluded or to be replaced on SYSUT2 (the format is shown in Figure 15).

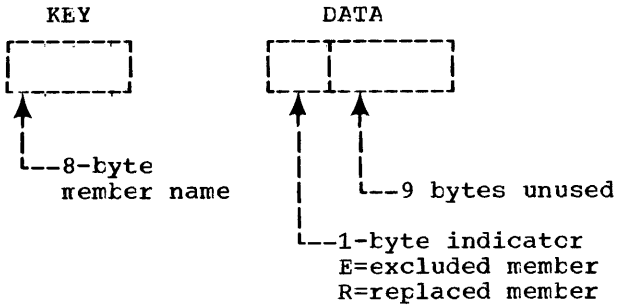


Figure 15. SYSUT2 Record Format (for a PDS Request only)

Message Writing (IEHMVESA, IEHMVESU)

All messages are written by IEHMVESU. Whenever IEHMVESS effects a message, it first interfaces through IEHMVESA, which contains messages and linkage to IEHMVESU.

DSGROUP Set-up (IEHMVESI, IEHMVESC, IEHMVESH)

Preliminary operations needed to copy a group of data sets are performed by three loads: IEHMVESI, IEHMVESC, and IEHMVESH.

IEHMVESI
opens the catalog and sets up a call to IEHMVESC.

IEHMVESC
reads the catalog and writes data set information on SYSUT1 (see Figure 18).

IEHMVESH
closes the catalog and sets up a request to copy a single data set, using data from SYSUT1.

Data Set and Volume Set-up (IEHMVESZ, IEHMVXSF, IEHMVESX, IEHMVESV, IEHMVESY)

Preliminary operations needed to copy a single data set are performed by five loads: IEHMVESZ, IEHMVXSF, IEHMVESX, IEHMVESV, and IEHMVESY.

IEHMVESZ
examines the request and sets up a call to IEHMVXSF. For a VOLUME request, IEHMVESZ reads the VTOC and obtains a DSCB; if a catalog is detected on the volume, its presence is noted, but no request to copy it is set up until the data sets are copied.

If an abnormal termination is indicated as the result of an error in either this module or a called subroutine, this module initiates termination of the program. User label exits are not processed at this time.

IEHMVXSF
is the volume mounter, described under the heading "Device Allocation and Volume Mounting." This execution of the volume mounter effects volume mounting.

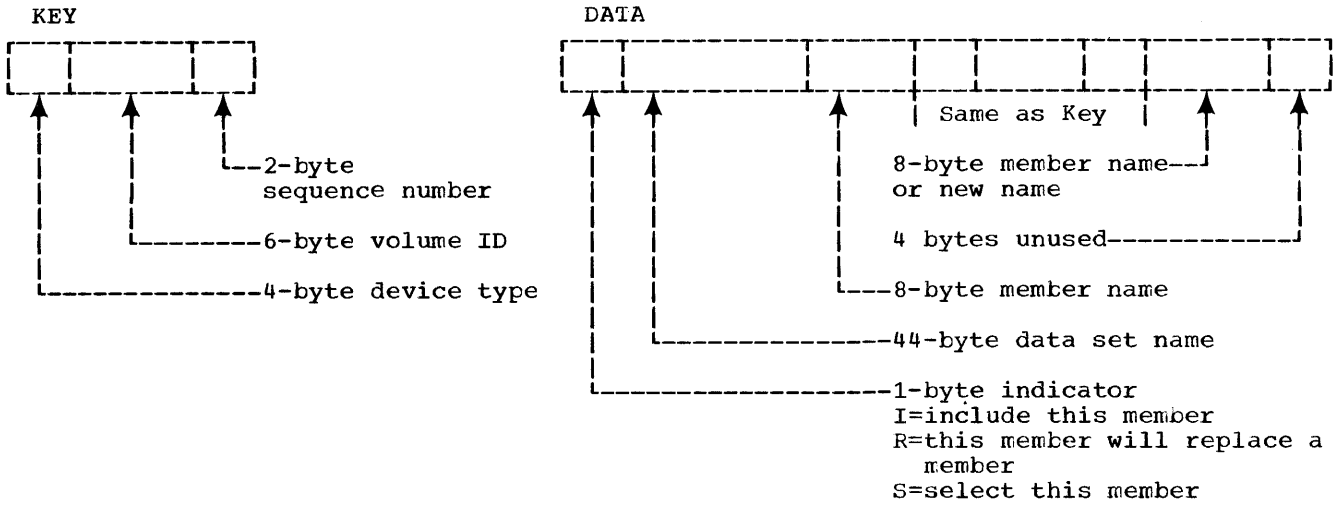


Figure 14. SYSUT1 Record Format (for a PDS request only)

IEHMVESX
performs tests on the data set to be copied. Tests include movability, unload or load, access method type, compatible block size, and compatible receiving device. If a catalog had been detected in IEHMVESZ, space is allocated for it on the 'TO' volume.

When user-label processing has been specified, this module processes the input header labels as it opens the input data set. If storage in which to save the labels is required, it is obtained in this module.

IEHMVESV
causes space to be allocated for the 'TO' data set. If user labels exist in the 'TO' data set and user-label processing has been requested, an additional track will be allocated for the user labels. The DS1EXT1 field of the DSCB is modified for this purpose. (For a preallocated 'TO' data set, (i.e., one that has been allocated before the execution of the IEHMOVE program) the user must provide a user-label track to permit the labels to be moved.) If the 'FROM' data set is a PDS, the directory entries of members to be copied are written on SYSUT3. During abnormal terminations that are handled by this module, no user-label exits are processed. If a user-label track has not been allocated, the message text in this load module informs the user that labels cannot be moved or copied.

IEHMVESY
builds, using DCB and label information specified by FROMDD and TODD (if given in an operation involving 7-track or 9-track unlabeled tape), the DCBs for the 'FROM' and 'TO' data sets and directs control to the appropriate module to copy, unload, or load the data set.

When user-label processing has been specified, this module processes the output of the user header labels that have been saved by the program, as it opens the output DCB.

If this module encounters errors such that an abnormal termination is indicated, the module initiates termination of the program. If user-label processing has been specified, the processing operations are not performed during the abnormal termination procedures.

If variable spanned records are indicated, this module will identify the record format and determine to which module control is to be given for processing each record. If logical copies involving changes in the record format (RECFM), the block size (BLKSIZE), or the logical record length (LRECL) of data set records are attempted, an error is indicated and a message is printed. The program will then attempt to move the rest of the data sets as requested.

This module also writes the first records of an unloaded data set and determines the module that next receives control to perform the actual move/copy operation.

PDS Subroutines (IEHMVESR, IEHMVETG, IEHMVXSF)

If a partitioned data set is being copied or unloaded, IEHMVESR is always used by the copying module, whereas IEHMVETG (which uses IEHMVXSF, the volume mounter) is used if the request specifies PDS. Figure 13 shows which loads use PDS subroutines.

IEHMVESR
is used to obtain a directory entry from the work data set SYSUT3. If the directory entry is from the PDS being copied, it was placed on SYSUT3 by IEHMVESV (Data Set and Volume Set-up); if the directory entry is from an INCLUDE or REPLACE option, it was placed on SYSUT3 by IEHMVETG. The format of a directory entry on SYSUT3 is shown in Figure 16.

IEHMVETG
places directory entries of members from INCLUDE, REPLACE, or SELECT options on SYSUT3. Each execution of IEHMVETG places the directory entries to be included from one PDS. When there are no more members to be included in the copy, IEHMVETG gives control to IEHMVESN. IEHMVXSF, the volume mounter, is used as needed. The logic of IEHMVXSF is described under the heading "Device Allocation and Volume Mounting."

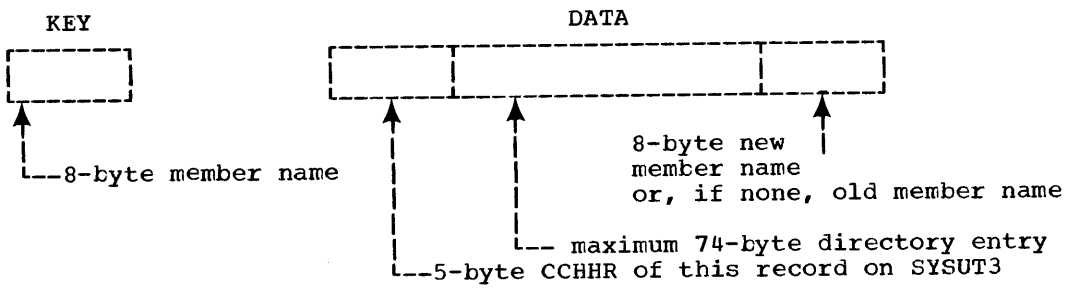


Figure 16. SYSUT3 Record Format

Copying, Unloading, and Loading

The load modules used to copy, unload, and load data are grouped according to the type of data set and format condition as shown in Figure 17. The modules are described below:

IEHMVETA
copies, unloads, or loads a catalog. If the catalog is to be loaded, it is in the format shown in Figure 18. The entries are then cataloged on the receiving volume. If the catalog is to be unloaded or copied, IEHMVESC first writes catalog entries on SYSUT1 as shown in Figure 18; IEHMVETA then catalogs them on the receiving volume (for a copy) or else simply writes them in the same format (for an unload).

IEHMVETL
copies, unloads, or loads a BDAM data set. The data set is copied using BDAM read and BSAM write (load mode) routines. If the input data set record format is type U, the block length of each physical record is read and calculated and then passed to BSAM write. This calculation is unnecessary for types F and V and is bypassed.

If user-label processing has been specified, this module obtains any necessary storage in which to save the labels. This is done when either the end of a data set has been reached or a switch to another volume is to be made. The saved labels will be passed to the data set wrap-up routines.

IEHMVETJ
copies a BSAM data set or a PDS. A BSAM data set is copied using BSAM in a simple read-write loop. When either the end of a data set or the end of a volume has been reached in reading, storage will be obtained, if necessary, for saving any labels for which processing has been requested. The saved labels will be passed to the data set wrap-up routines. For a PDS, at the time IEHMVETJ receives control, the directory entries of members (barring any EXCLUDEs or REPLACEs) of the 'FROM' PDS are on SYSUT3, where they were written by IEHMVESV (Data Set and Volume Set-up). The 'FROM' PDS is then copied as follows:

1. IEHMVETJ directs control to IEHMVESR, which reads from SYSUT3 one directory entry of the PDS.

2. If the entry indicates a note list is present, IEHMVETJ reads the note list. Using BSAM, IEHMVETJ then reads and writes member records up to the note list. The note list is then updated and written with the new TTRs of the members.

3. When all note lists and member records have been written, the directory entry is updated with the new note list addresses and then stowed. This process is repeated for each directory entry on SYSUT3. When the 'FROM' PDS has been copied, control is given to IEHMVETG to write on SYSUT3 the directory entries for all members to be included, selected, or to replace from another PDS. IEHMVETJ then copies these members as outlined. IEHMVXSF (the volume mounter) is given control by IEHMVETG as needed.

IEHMVESL and IEHMVESM
copy partitioned and BSAM data sets if the 'TO' data set has been preallocated (that is, before execution of IEHMOVE) and the 'FROM' and 'TO' DSCBs indicate that reblocking is necessary in order to perform copying.

IEHMVESL
copies (with reblocking) a PDS or BSAM data set having type F record format. Blocks are read (using BSAM) until the output block size is surpassed, and then logical records are sectioned

from the high-order end of the buffer until the output block size is reached. The block is then written, using BSAM. The last block (of a BSAM data set or of a member of a PDS) is written as a truncated block if necessary. For a PDS, any user TTRs are ignored.

If user-label processing has been requested, this module will, when reaching either the end of a data set or the end of a volume, obtain necessary storage in which to save the labels. When the module passes control to the data set wrap-up routines, the saved labels are passed to the routine that receives control.

IEHMVESM
: copies (with reblocking) a PDS or BSAM data set having type V record format. The operation is similar to that of IEHMVESL: blocks are read using BSAM until the maximum output block size is surpassed, and then logical records are sectioned from the high-order end of the buffer until the size of the output buffer is not greater than the maximum block size. As with IEHMVESL, any user TTRs are ignored in a PDS.

  If user-label processing has been requested, this module will, when reaching either the end of a data set or the end of a volume, obtain necessary storage in which to save the labels. When the module passes control to the data set wrap-up routines, the saved labels are passed to the routine that receives control.

IEHMVERD
: unloads a PDS or a BSAM data set. For a BSAM data set, the data set is read one block at a time. After each read, the block is prefixed by three-to-six bytes of control information, and then deblocked into 78-byte sections. Each section is then prefixed with a 2-byte physical sequence number. The resultant 80-byte blocks are then written, or, if the receiving device permits, are reblocked in groups of ten to be written out as 800-byte blocks. The last block written is padded with blanks. For a PDS, the directory entry of a member is first read into the buffer. Each note list is read and followed by member records which precede it in the PDS. Aliases are read last. The buffer is sectioned and control information inserted. The process is repeated for each directory entry and its member blocks. Directory entries are read from SYSUT3 by IEHMVESR. The options INCLUDE, REPLACE and SELECT are ignored in unloading; the option EXCLUDE is not ignored.

  If user-label processing has been requested, this module will, when reaching either the end of a data set or the end of a volume, obtain necessary storage in which to save the labels. When the module passes control to the data set wrap-up routines, the saved labels are passed to the routine that receives control.

Type of Data Set	Format Condition	Load Modules Used
Catalog	Normal	IEHMVETA with IEHMVESC
Catalog	Previously unloaded	IEHMVETA
BDAM	Any	IEHMVETL
PDS or BSAM	Normal, copiable no reblocking	IEHMVETJ with IEHMVETG (PDS only) IEHMVESR IEHMVXSF
PDS or BSAM	Type F, copiable with reblocking	IEHMVESL with IEHMVETG (PDS only) IEHMVESR (PDS only) IEHMVXSF
PDS or BSAM	Type V, copiable with reblocking	IEHMVESM with IEHMVETG (PDS only) IEHMVESR IEHMVXSR
PDS or BSAM	Normal, uncopiable must be unloaded	IEHMVERD with IEHMVESR IEHMVXSF
PDS or BSAM	Previously unloaded	IEHMVERA

Figure 17. Load Module Groupings for Copying, Unloading, and Loading

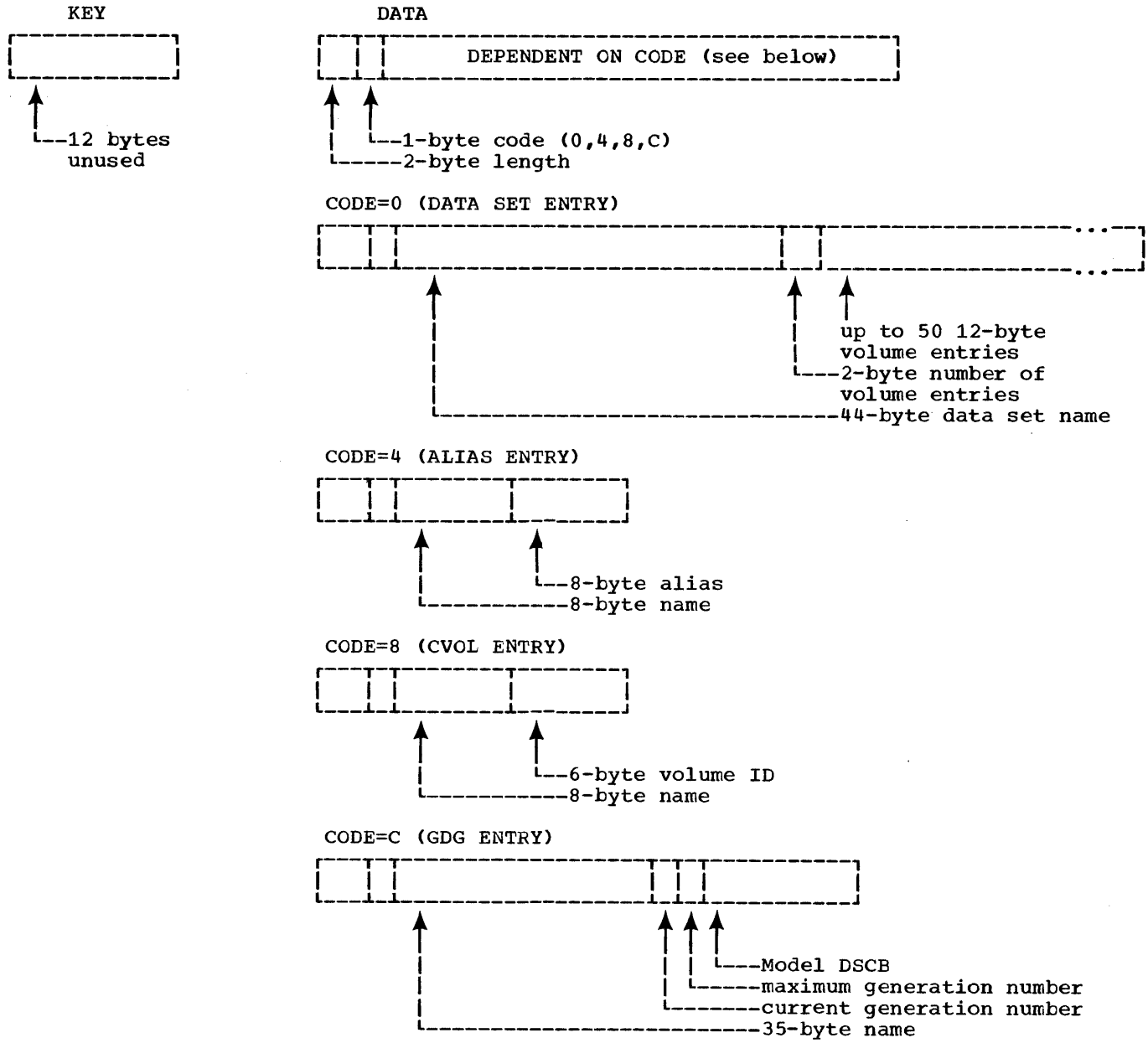


Figure 18. SYSUT1 and SYSUT2 Record Formats for DSGROUP; SYSUT1 Record Formats for CATALOG

IEHMVERA

loads a PDS or BSAM data set. For a BSAM data set, a block is read, the control information is removed and analyzed, and successive blocks are read until a block from the original data set is reconstructed. The block is then written, and the process is repeated until the original data set is loaded. For a PDS, the process is similar: the directory entry is first reconstructed, but is not stowed until member blocks have been written and note lists updated and written. At the time the data set was unloaded, an entry was written, followed by member blocks (in unloaded format).

If user-label processing has been requested, this module will, when reaching either the end of a data set or the end of a volume, obtain necessary storage in which to save the labels. When the module passes control to the data set wrap-up routines, the saved labels are passed to the routine that receives control.

DSGROUP Wrap-up (IEHMVESH, IEHMVETA)

After each data set of a DSGROUP has been copied, control is given to IEHMVESN of the Data Set Wrap-up portion of the program. If scratching of the 'FROM' data set is necessary (for a MOVE DSGROUP, for example), control is given to IEHMVESQ, which scratches the 'FROM' data set, and then gives control to IEHMVESH. If scratching is not necessary, control goes directly from IEHMVESN to IEHMVESH.

IEHMVESH
'TO' data set writes 'FROM' data set information on SYSUT2 in the same format in which the information was originally written on SYSUT1 (see Figure 18). This information is written so that the catalog can be updated as needed. If there is another data set in the group to be copied, IEHMVESH gives control to IEHMVESZ to set up the next copy; if all the data sets have been copied, IEHMVESH sets up a request to catalog the updated data set information on SYSUT2 and gives control to IEHMVETA.

IEHMVETA
reads SYSUT2 and catalogs the information. The process is the same as that followed by IEHMVETA in copying a catalog.

Data Set Wrap-up (IEHMVESN, IEHMVESO, IEHMVESP, IEHMVESQ, IEHMVESK)

Load modules in this group perform terminal operations following the copying, unloading, or loading of a data set processed for a PDS, DATA SET, or VOLUME request. In addition, when all requests have been serviced, control is given to IEHMVESK.

IEHMVESN
completes the moving or copying of a data set and closes the 'TO' and 'FROM' data sets.

If user labels have been specified, and if output trailer labels have been saved in storage, these labels are written out and the storage area is released. If a user-label track has not been allocated, the message text in this load module informs the user that labels cannot be moved or copied.

IEHMVESO
is entered following an unsuccessful copying, unloading, or loading operation, or following a test ((Data Set and Volume Set-up) indicating a request could not be honored. IEHMVESO prints a diagnostic message and scratches the 'TO' data set.

IEHMVESP
performs terminal operations following a COPY request, including any specified or implied cataloging, uncataloging, and scratching.

IEHMVESQ
performs terminal operations following a MOVE request, including any specified or implied cataloging, recataloging, and scratching.

IEHMVESK
is entered when all requests have been serviced, or when a permanent I/O error has been detected on the printer. IEHMVESK closes SYSIN and SYSPRINT, closes and scratches the work data sets, frees main storage, and returns control to the stem, IEHMOVE. During abnormal termination handled by this module, user-label exits are not processed.

COMMUNICATION AREA (IEHMVV)

The communication area for the program is defined at assembly time by the macro instruction IEHMVV, which is internal to the IEHMOVE program. Register 12 contains a pointer to the communication area whenever a request for a module is issued.

The macro instruction IEHMVV generates a dummy section (also IEHMVV) containing work areas and control data for all object modules of the program. Main storage for the dummy section IEHMVV is obtained dynamically by the stem.

The communication area consists of the following parts:

- A work area of 512 bytes (IEHMV00).
- The addresses of the beginning and end of an 800-byte work area (IEHMVV10).
- A table of switches controlling the flow of the program (IEHMVV20).
- A control table containing the return codes of the control statement scan routine (IEHMVESJ).
- A table of control data for volume lists and include-exclude-replace lists (IEHMVV21-IEHMVV26).
- A table of addresses of the FROM data set's DCB, DSCB, DECB and ddname (IEHMVV30) and the TO data set's DCB, DECB and ddname (IEHMVV31). Each address is stored in a fullword.

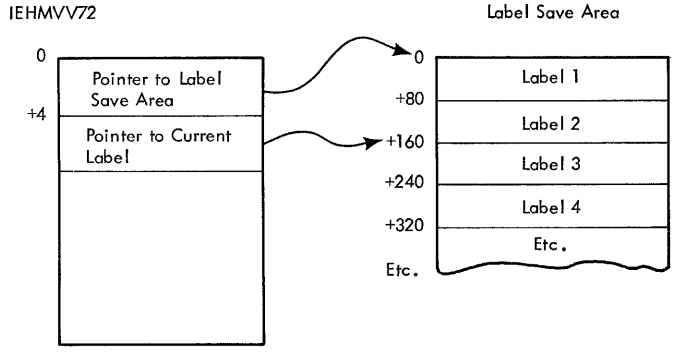
- A table of the addresses of the DCBs and DECBs of SYSPRINT (IEHMVV33), SYSIN (IEHMVV32), and SYSLIB (IEHMVV34).

- A table of work data set control data (IEHMVV39).

- A table of addresses of work areas for loading, unloading, including, replacing, and copying a PDS.

- The DCB exit list (for user-label processing) defined by the macro instruction IEHDCBXL. This list is found in the 40-byte section IEHMVV70 of the communication area. Included in the list are symbolic names for the input and output header-label processing subroutines, the input and output trailer-label processing subroutines, the DCB exit, and the OPENJ JFCB exit.

- An area containing pointers both to the storage area (the label save area) obtained for user labels and to the current label being processed. These pointers are in the 20-byte section IEHMVV72. For the first label being processed, both pointers will indicate the same address. Figure 19 indicates these relationships.



Example of pointers when third label is being processed

Figure 19. Label Save Area Pointers

IEHMOVE WORK DATA SET RECORD FORMATS

The program uses three work data sets: SYSUT1, SYSUT2, and SYSUT3. How a work data set is used depends on the function being performed by the program. The following table (Figure 20) shows where record formats may be found. A blank entry indicates that the work data set is not used for the indicated function. The entry 16* indicates that SYSUT3 is used for any partitioned data sets found within a group of data sets or a volume.

Function	SYSUT1	SYSUT2	SYSUT3
Single Data Set (not a PDS)			
Single Data Set (PDS)	Fig. 14	Fig. 15	Fig. 16
Volume			Fig. 16*
DSGroup	Fig. 18	Fig. 18	Fig. 16*
Catalog	Fig. 18		

Figure 20. Where to Find Record Formats

The device on which the work data sets reside is allocated by job management and is associated with the ddname SYSUT1. The spaces occupied by the work data sets specified by the names SYSUT1, SYSUT2, and SYSUT3 are dynamically allocated.

Obtaining Space for a Work Data Set

Space for a work data set (e.g., SYSUT3) is obtained from DADSM in the following way:

1. The first time space is requested, it is requested for the data set **SYSUT3.

2. If the return from DADSM indicates that a DSCB for the requested data set space already exists on the VTOC, the name previously submitted to DADSM is qualified by the index name consisting of the single character * and the modified request is submitted to DADSM.

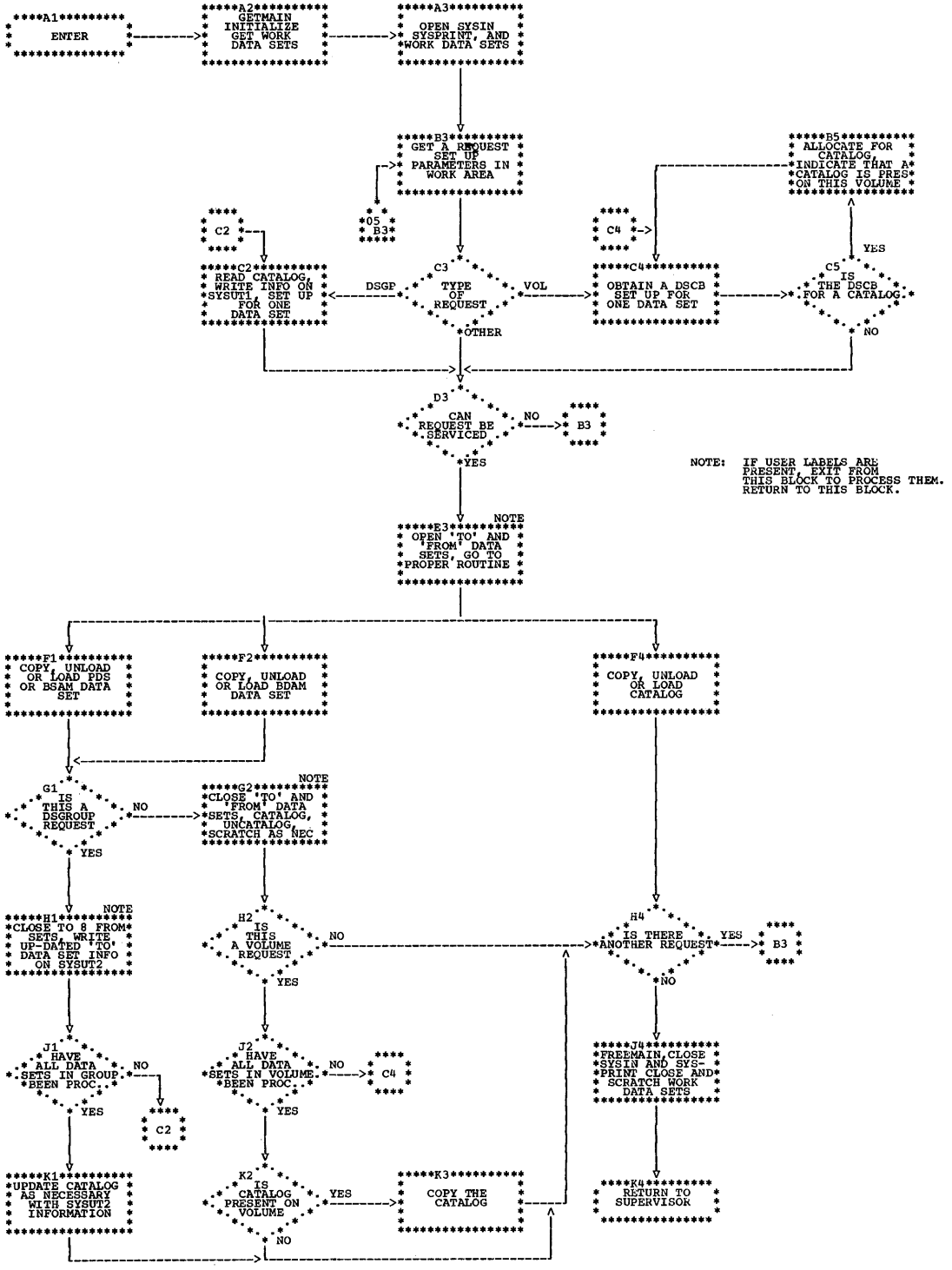
Step 2 is repeated until space is allocated or until 44 bytes have been used with no success. Thus, the first request for space for SYSUT3 either results in space being allocated for the 8-byte name **SYSUT3 or an indication that a data set of the name **SYSUT3 already resides on the subject volume. If the latter, space is requested for the data set **SYSUT3.*. The third request, if necessary, would specify the name **SYSUT3.*.*. A count of the number of times the name has been qualified is maintained in the communication area, IEHMVV.

After space has been allocated for the work data sets, they are opened in the order: SYSUT1, SYSUT2, SYSUT3.

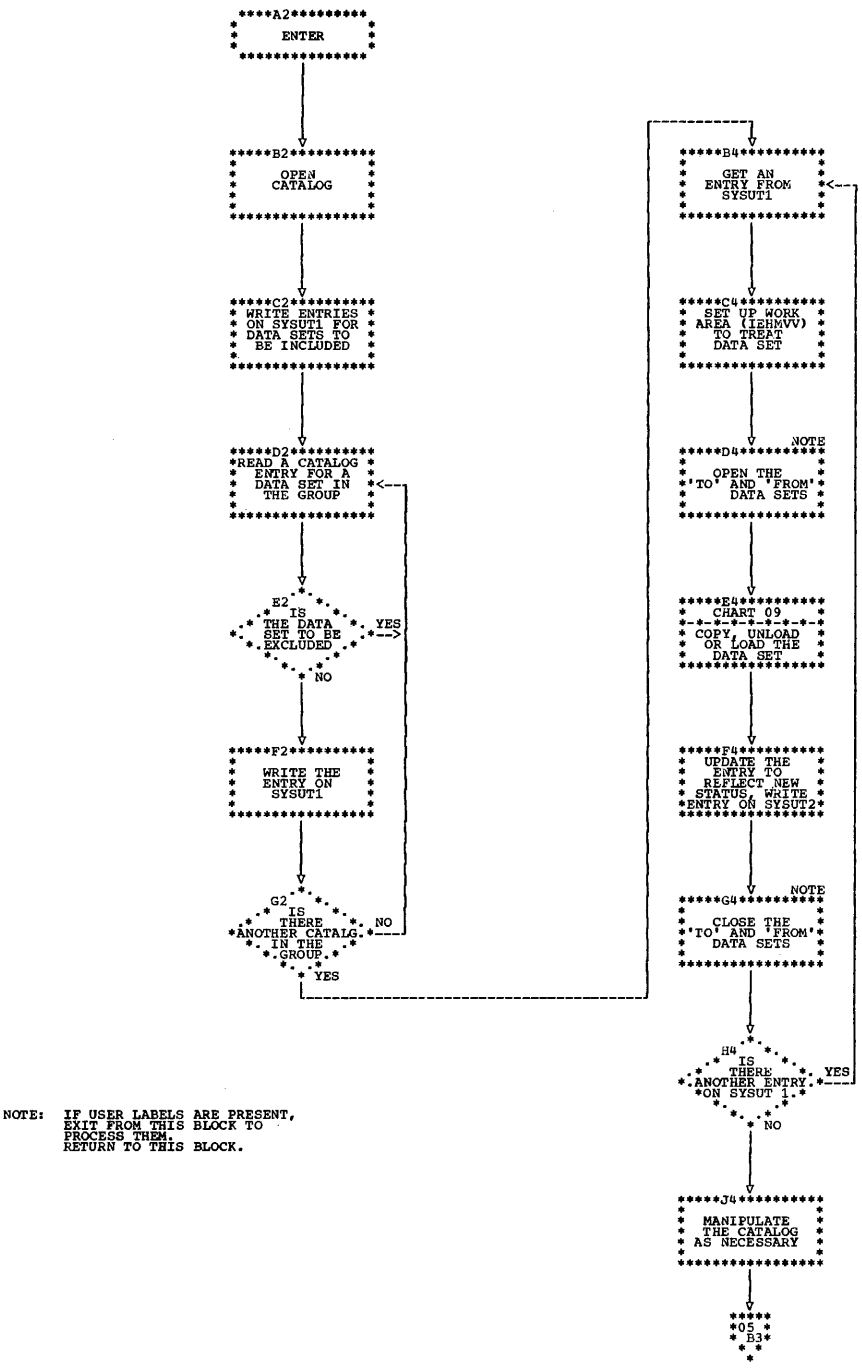
Releasing Space Used by a Work Data Set

The work data sets are not closed until final wrap-up. At this time, SYSUT3 is first closed and scratched, then SYSUT2 is renamed to SYSUT3 and closed and scratched, and then SYSUT1 is renamed to SYSUT3 and closed and scratched.

Chart 05. IEHMOVE Overall Logic



• Chart 06. IEHMOVE DSGROUP Logic



NOTE: IF USER LABELS ARE PRESENT, EXIT FROM THIS BLOCK TO PROCESS THEM. RETURN TO THIS BLOCK.

• Chart 07. IEHMOVE VOLUME Logic

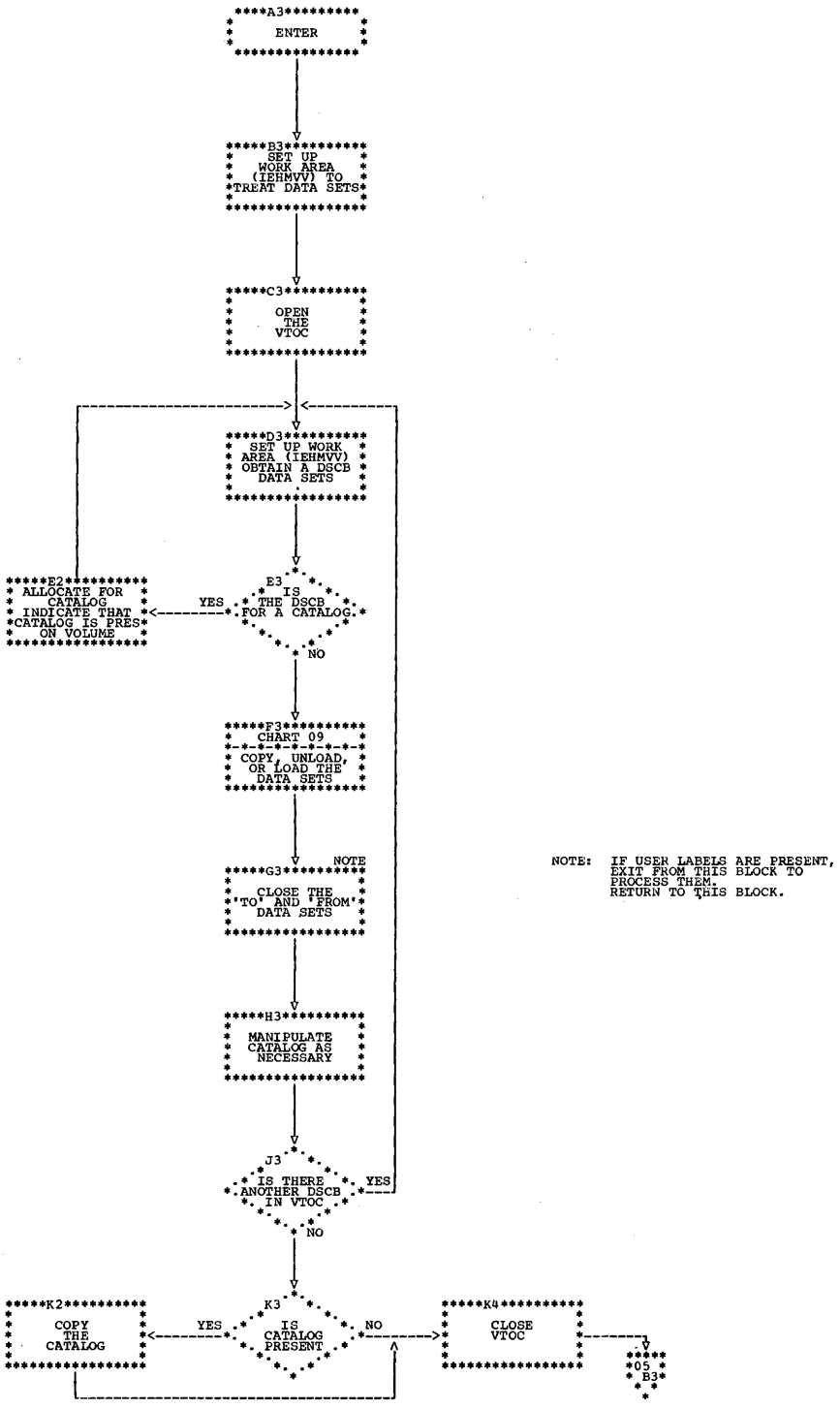
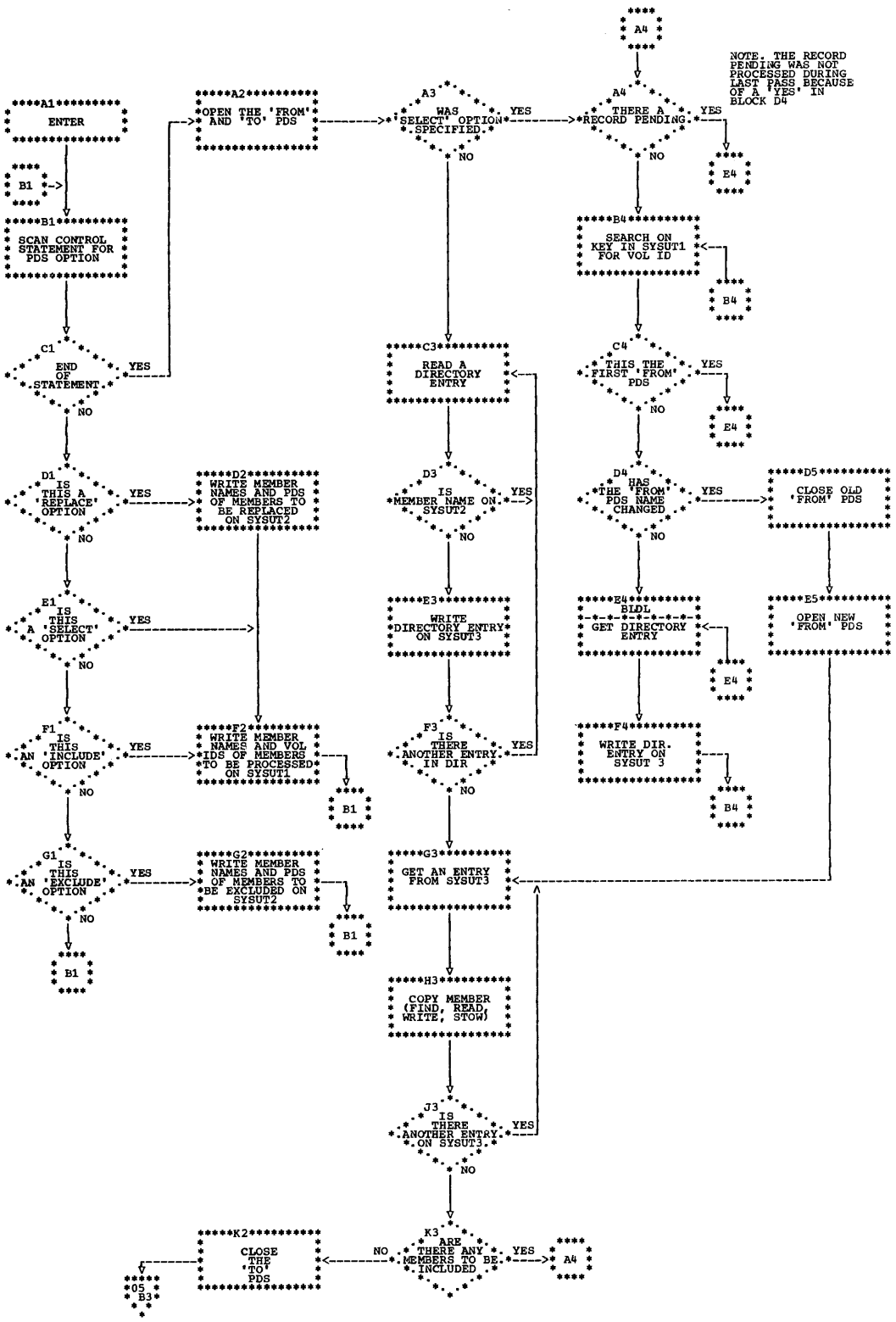


Chart 08. IEHMOVE PDS Logic



• Chart 09. IEHMOVE DSNAME Logic

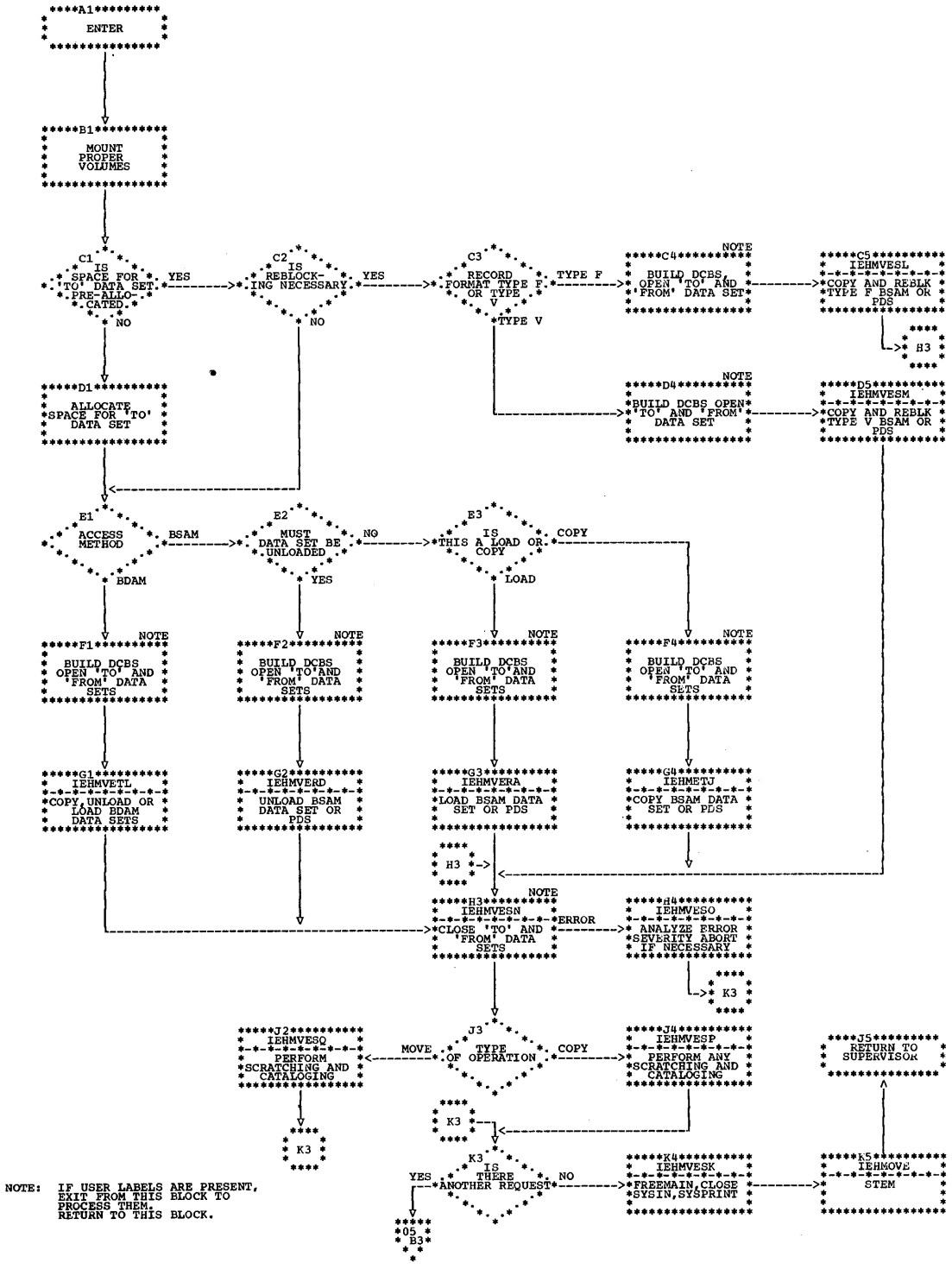
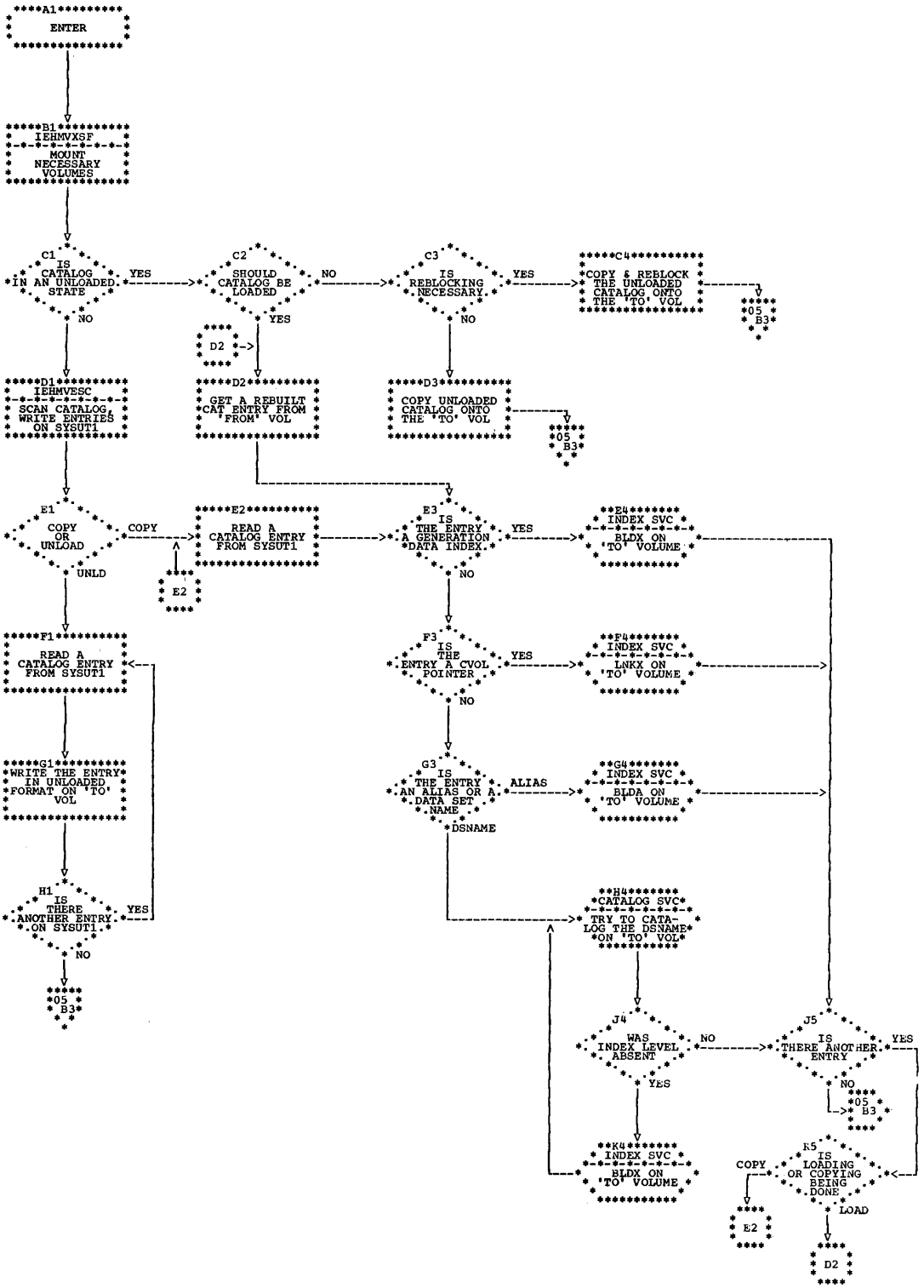


Chart 10. IEHMOVE CATALOG Logic



Listing System Control Data (IEHLIST)

The IEHLIST utility program performs three functions:

- It prints a catalog or partial catalog.
- It prints a volume table of contents (VTOC).
- It prints the directories of up to ten partitioned data sets (PDSs).

The program is serially reusable, but not reenterable.

PROGRAM STRUCTURE

The overlay structure of the program is shown in Figure 21. The program consists of the following control sections (CSECTs):

- IEHROOT performs basic program initialization. It is resident in main storage throughout the program's execution, unlike the other CSECTS. IEHRCOT contains V-type address constants needed by the overlay supervisor.
- IEHMSG contains only messages.
- IEHPSEG analyzes requests.
- RDCDRT scans control statements.
- DEVNAMET is the device name table.
- IEHINSEG interprets parameters supplied by a calling program.
- IEHQSEG scans and prints cataloged data.
- IEHRSEG scans and prints VTOC data.
- IEHSSEG scans and prints PDS directory data.
- IEHVOLMT mounts necessary volumes. It is described under the heading "Device Allocation and Volume Mounting."
- DEVMASKT is a device mask table used by IEHVOLMT.

Chart 11 shows the logical flow of control through the program. Figure 22 shows the structural flow of the program, including the successive phases of the contents of main storage during the program's execution. The logic of each control section (CSECT) of the program is described in the following paragraphs.

IEHROOT
: contains miscellaneous routines needed in main storage throughout the program's execution (PERRPR, WORKERR, PTERM, LINEPR, DOPOINT, and PBEGIN), together with several communication areas (CARDIN, PRINTOUT, WORKIN, and RONTAB).

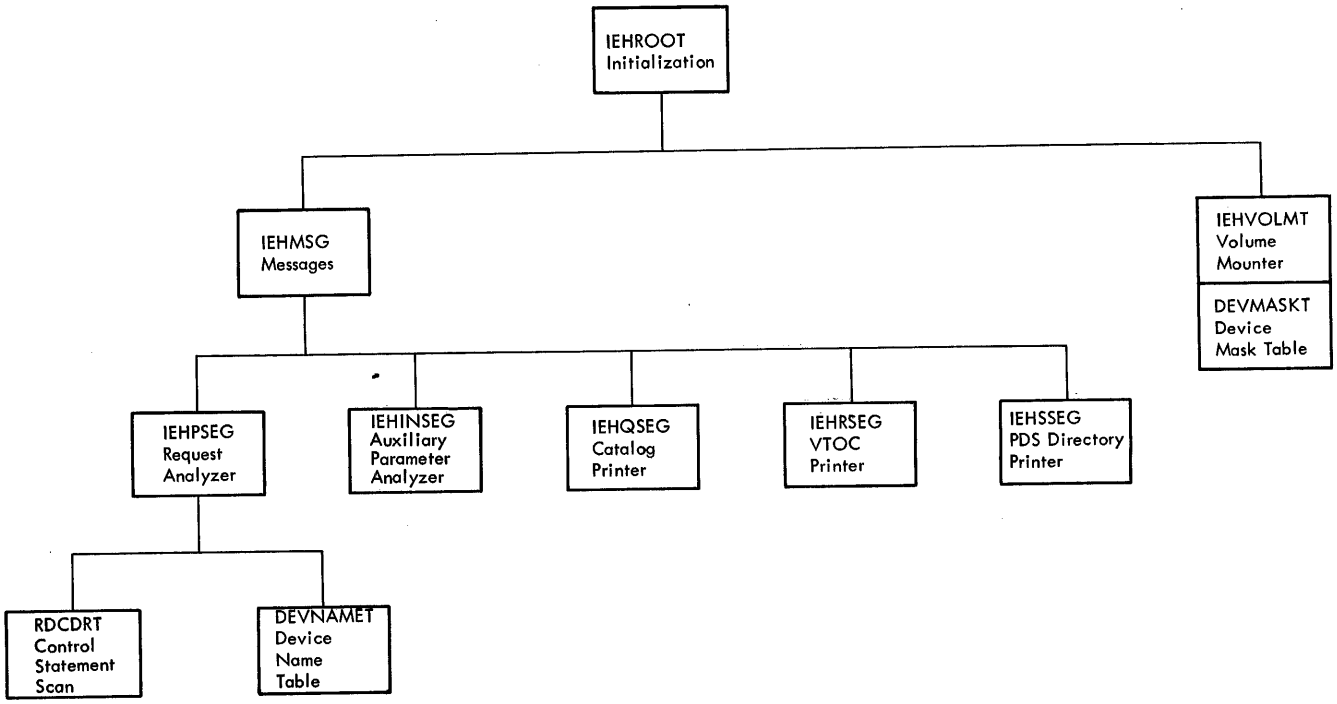


Figure 21. The Overlay Structure of the IEHLIST Program

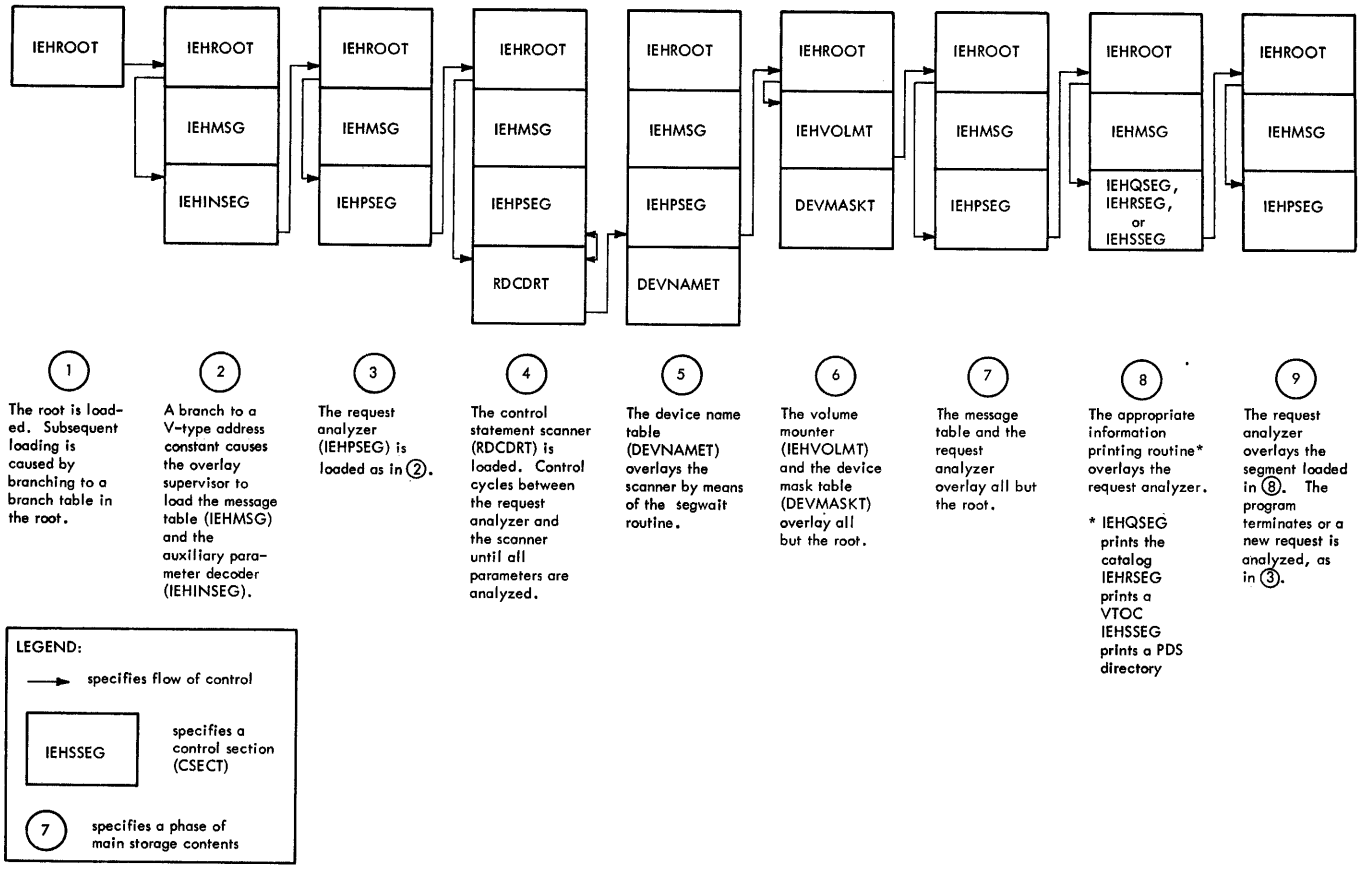


Figure 22. The Structural Flow of the IEHLIST Program

PERRPR
causes any invalid control statement to be printed, and gives control to PTERM or PBEGIN, depending on whether there are any more control statements.

WORKERR
treats all SYNAD exits.

PTERM
receives control when all job requests have been serviced or aborted, and ends the job.

LINEPR
prints all program output.

DOPOINT
issues all POINT supervisor calls used by the program.

PBEGIN
directs control to the segments by means of a branch table of V-type address constants.

The following areas are located in the root:

CARDIN
is the DCB for reading control statements.

PRINTOUT
is the DCB for printing the catalog, VTOC, or PDS.

WORKIN
is the DCB for reading the catalog, VTOC, or PDS directory.

RONTAB
is the parameter list for the volume mounting routine.

IEHPSEG
analyzes requests and directs control to the appropriate routine to print the requested information. It contains the following subroutines:

PBEGIN
directs control to IEHINSEG to interpret calling program parameters, and to open SYSIN and SYSPRINT.

PON
directs control to RDCDRT to obtain control card information.

PKEY
is given control when a keyword is returned by RDCDRT.

PPARAM
is given control when a parameter is returned by RDCDRT. When the parameter supplied to the (optional) VOL keyword is returned, the parameter is used as a search argument in the device name table. The value retrieved is used by PWORKIN.

PWORKIN
constructs the calling sequence for IEHVOLMT.

PHEAD
prints a header and gives control to the appropriate routine to print catalog, VTOC, or PDS directory information.

PTERM
receives control following the printing of catalog, VTOC, or PDS directory information. If there is another request, PTERM directs control to PON; otherwise, PTERM closes SYSIN and SYSPRINT, and returns control to the supervisor.

RDCDRT
scans utility control statements. It is described under the heading "Control Card Scanner."

IEHINSEG
interprets auxiliary parameters supplied by a calling program (these parameters are described under the heading "Auxiliary Parameters"), and also opens SYSIN and SYSPRINT or their substitutes, as specified in the calling sequence to the program.

IEHQSEG
prints the catalog. It gains control indirectly from the request analyzer, IEHPSEG, by means of a branch to a branch table in the root. IEHQSEG contains the following routines:

QCHECK1
scans the catalog for general information and prints it. Actual printing is done via a branch-and-link to LINEPR in the root.

QHEAD
prints a catalog header after the general information is printed. Actual printing is done via a branch-and-link to LINEPR.

QALL
scans high-level node points in the catalog and passes them to QLOCATE. QALL is used only in the case of an entire catalog printout.

QLOCATE
scans from a node point to successive index levels until a data set pointer is found. A fully qualified data set name is placed at location INDXNAME for routine LPRDATA.

LPRDATA
prints information pertinent to a data set.

QCATREAD
performs all the reading of a catalog for the catalog function of the program.

IEHRSEG
prints a VTOC. Control is gained indirectly from IEHPSEG by means of a branch to a branch table in the root. IEHRSEG contains the following routines:

RPARTIAL
treats requests for partial VTOC printouts. Successive DSCBs are printed by linking to RPRDSCB.

RENTIRE
treats requests for entire VTOC printouts and differs from RPARTIAL in that the VTOC must be opened.

REODAD
calculates and prints volume space information for an entire VTOC printout.

RPRDSCB
prints a DSCB that has been read by RPARTIAL or RENTIRE.

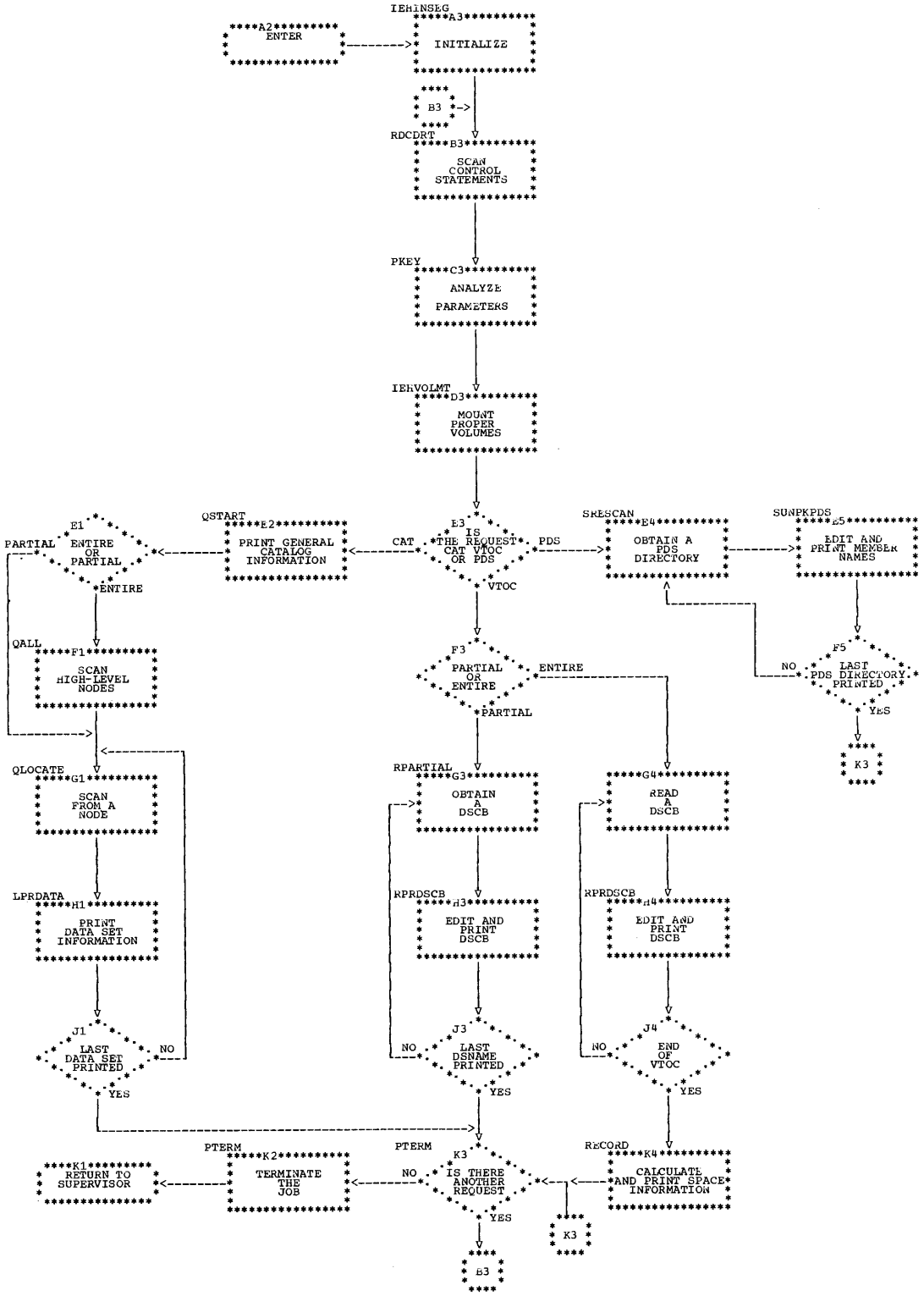
RREAD
reads the VTOC.

IEHSSEG
prints PDS directories. Control is gained indirectly from IEHPSEG by means of a branch to a branch table in the root. IEHSSEG contains the following routines:

SSTART
obtains the directory of a given PDS.

SRESCAN
prints the member names of the directory. Actual printing is done by linking to LINEPR.

Chart 11. IEHLIST - Listing System Control Data

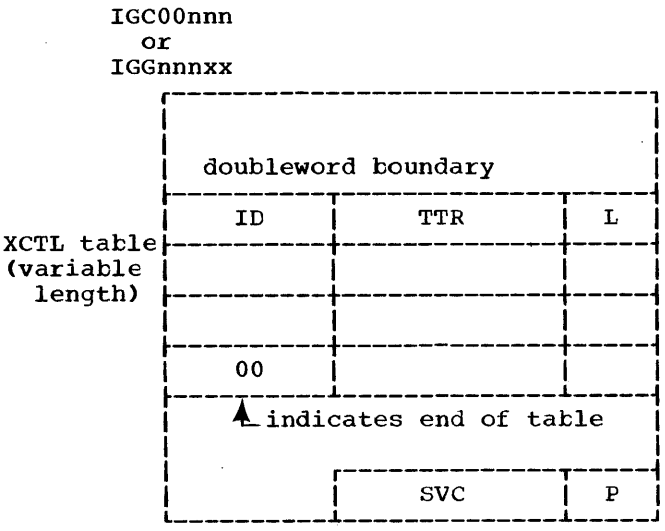


Updating XCTL Tables for OPEN, CLOSE, and EOV (IEHIOSUP)

The IEHIOSUP program updates the XCTL tables embedded within various load modules of the I/O support functions OPEN, CLOSE, and EOV. The program is executed as a result of job control statements in the job stream at the time of system generation. The program is not serially reusable. It consists of one load module, IEHIOSUP.

The name of the load module for the first phase of each of the I/O support functions listed above is of the form IGC00nnn, where nnn is the decimal SVC code for the data management function. The names of subsequent load modules are of the form IGGnnnxx, where nnn is the decimal SVC code for the data management function, and xx is a load module identifier. If the seventh character of the load module name is alphabetic, the load module contains no XCTL table.

An XCTL table is always present in the first type of load module, but not always present in the second. If present, the table may be embedded anywhere within the load module (see Figure 23). The last byte of the load module is a relative pointer (in double words) to the table.



ID = 2-byte entry identifier of a subsequent load
TTR = 3-byte relative track address of the subsequent load
L = 1-byte length (double-words) of the subsequent load
SVC = 3-byte decimal SVC number of support function
P = 1-byte relative pointer (double-words) to XCTL table

Figure 23. Embedded XCTL Table Format

Each entry within an XCTL table consists of the identifier of a subsequent load module, the location of the load module (TTR), and the length (in double words) of the load module.

PROGRAM FLOW

The flow of the IEHIOSUP program is shown in Chart 12.

Finding the Load Module

Load modules of the first type (IGC00nnn) are updated first. If a load module of this type is not found, an appropriate message is printed and the program is aborted. Load modules of the second type are processed only after successful processing of the first type; during this processing, the program ends normally if either all load module XCTL tables are updated or the end of the directory is reached in searching for a load module entry.

Entries for load modules are sought for in order of increasing binary value (in accordance with the organization of the directory) by reading a directory record and comparing the record key to the name of the desired load module. When the record key compares higher than or equal to the load module name, the entry is sought for (sequentially) in the record. If the load module is of the first type (IGC00nnn) and no entry is found for it in the record, the program aborts. Load module names whose seventh character is alphabetic are ignored, since the load modules they name have no XCTL tables.

When the entry for the load module is found in a directory record, the location (TTR) of the load module is extracted from the entry and converted to an absolute address (MBBCCHHR). The conversion is effected via the execution of the program IECPCNVT, which is passed the TTR to be converted and the address of the appropriate DEB. The address of the IECPCNVT program is found in the Communications Vector Table at absolute (decimal) location 44.

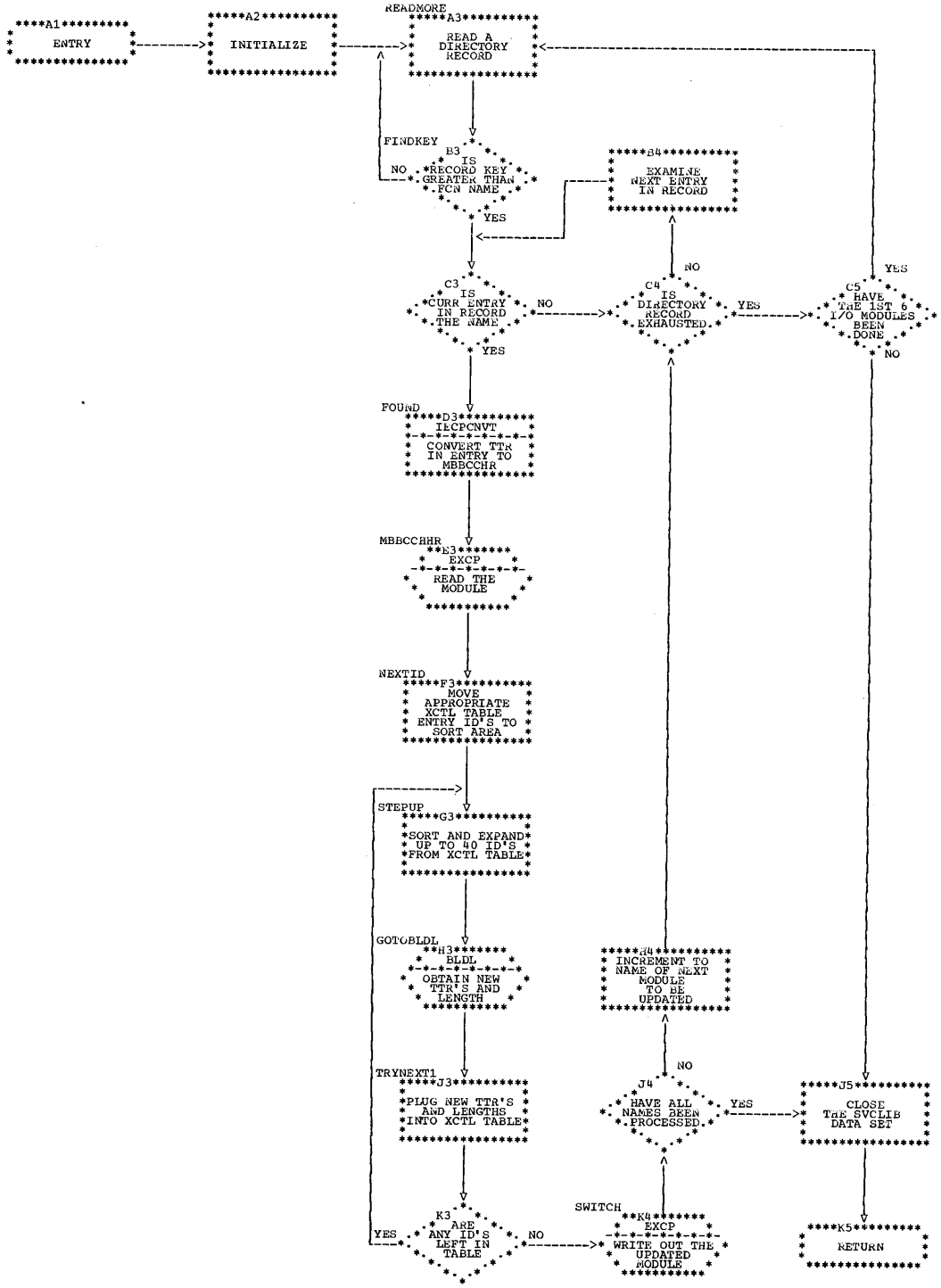
Updating the XCTL Table

The absolute address of the load module desired is then used to read the load module into the buffer BUFFER. Reading of the load module is done via the EXCP macro instruction; the channel program is at location CCWREAD, and the DCB is at location EXCPDCB. When the load module has been read into main storage, the address of its last byte is determined using the count field of the CCW and the residual count of

the CSW and is used to calculate the address of the beginning of the XCTL table within the load module. Up to 40 entry identifiers are then moved from the XCTL table and sorted in the area SORTAREA. If more than 40 entries are in the XCTL table, a switch (SWITCH + 1) is set. After the entry IDs are sorted, each is expanded to its full 8-byte form (i.e., IGGnnnxx). The sorted, expanded IDs are then passed to the BLDL macro instruction, which returns in BLDLAREA the new entry values (TTR and length) for each ID. The values are then moved to XCTL table. Any remaining entry IDs in the table are sorted, expanded, and passed to BLDL, 40 at a time, and updated in the same manner.

The entire load module containing the updated XCTL table is then written at its original location. If there are no more load modules to be processed, the SVCLIB data set is closed and the program terminates. Otherwise, control cycles as indicated in Chart 12 until all load modules are processed or an error condition is returned by BLDL or EXCP. Such an error condition results in abnormal termination of the program.

Chart 12. IEHIOSUP - Updating I/O Support XCTL Tables



Initializing the SYS1.LOGREC Data Set (IFCDIP00)

The IFCDIP00 program is executed during system generation to initialize the SYS1.LOGREC data set (a data set used by systems environment recording modules to record CPU, channel, and I/O device errors).

This program is executed as a result of job control statements provided by the GENERATE macro instruction during the SYSGEN process. Input to the program (as specified in the EXEC statement) consists of the (decimal) number[1] of unit control blocks (UCBs) in the system, and the system resident device type code (for an explanation of the code, see "SYS1.LOGREC Record Format").

The output of this program at normal completion is of three types:

- The initialized data set SYS1.LOGREC. (See the section "SYS1.LOGREC Record Format.")
- Information to be used as parameters for executing the environment recording edit and print (EREP) program.
- Information to be used for recording CPU, channel, and I/O device errors by the systems environment recording modules.

PROGRAM FLOW

Chart 13 shows the flow of the program, which consists of one load module (IFCDIP00). The data set SYS1.LOGREC to be written consists of three subsets:

- A header record, written by this program.
- A variable number of statistical data records (STAT/RECs), written by this program with data fields of zeros.
- A record entry area beginning on the first track following the STAT/RECs, not written by this program.

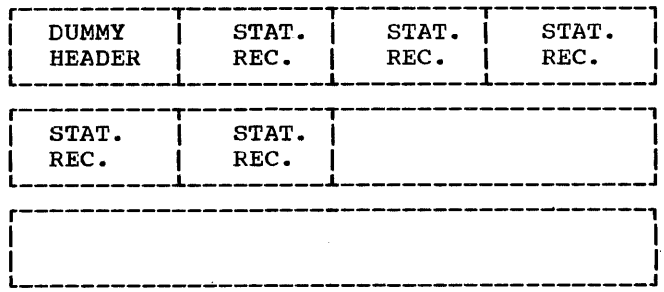
SYS1.LOGREC records are written using BSAM WRITE. Diagnostic messages are written using the WTO macro instruction.

The program is executed in two passes: the first pass (see Figure 24) initializes the program, writes a dummy header record, and writes as many statistical data records as there are UCBs for the system; the second pass (see Figure 24) uses the data obtained in the first pass to write a genuine header record over the dummy, and then writes as many statistical data records (over and following those written during the first pass) as are necessary to fill out the track occupied by the last statistical data record written during the first pass.

---

[1]This number is equal to the number of uniquely addressable I/O devices in the system.

SYS1.LOGREC After First Pass of IFCD1P00



SYS1.LOGREC After Second Pass of IFCD1P00

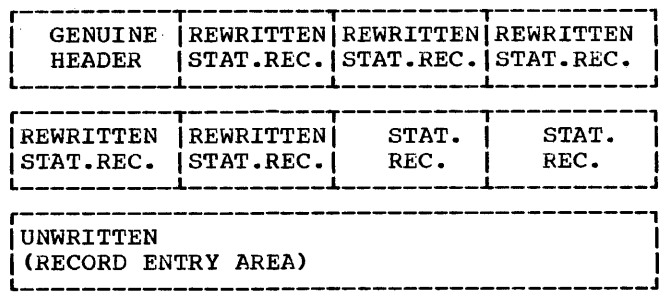


Figure 24. SYS1.LOGREC After First and Second Passes of IFCD1P00

First Pass

Program initialization consists of saving registers and analyzing the input from the EXEC statement. The dummy header is then initialized and written. The location (in TTR format) of the dummy header is saved for the second pass. The first pass statistical data records are then written, each of which consists of a 2-byte key (ascending sequence) and a 38-byte data field of zeros. The location of the last statistical data record written during the first pass is saved for the second pass, where it will be used to compute information necessary to complete the genuine header record. The program then enters the second pass.

Second Pass

A switch (PASS) is set, indicating that the program has entered the second pass. This switch will be interrogated following the rewriting of the statistical data records

in this pass. First, however, data necessary to the genuine header record is computed. A description of the fields of the header record may be found under "SYS1.LOGREC Record Format." The values supplied to these fields are computed using the track number obtained by the NOTE routine following the writing of the last statistical data record written during the first pass.

The genuine header is then written. Following this, the original statistical data records are rewritten. The switch PASS is interrogated, and indicates that the second pass has been entered. The track containing the last statistical data record is then padded with additional statistical data records. SYS1.LOGREC is closed, and IFCD1P00 returns control to the supervisor.

SYS1.LOGREC RECORD FORMAT

The SYS1.LOGREC data set consists of three subsets:

- A header record, written by this program.
- A variable number of statistical data records (STAT/RECs), written by this program and initialized to zero.
- A record entry area (RE), not written by this program.

Header Record

The header record is a 38-byte data field preceded by a 2-byte key of hexadecimal FFFF. The header record contains the following fields:

1. A four-byte field containing the address (CCHH) of the first track in the SYS1.LOGREC extent.
2. A four-byte field containing the address (CCHH) of the last track in the SYS1.LOGREC extent.
3. A one-byte constant containing the highest address of a track on a cylinder of the system resident device.
4. A seven-byte field containing the address and ID (BBCCHHR) of the first track of the RE area. The ID is set to zero.
5. A two-byte field containing the number of remaining bytes on the last RE track written. This field is initially identical to field 6.
6. A two-byte constant equal to the track byte capacity for the system resident device.
7. A seven-byte field containing the address and ID (BBCCHHR) of the last track written in the RE area. This is initially identical to field 4.
8. A two-byte field containing the number of UCBs in the system.
9. A two-byte field containing the number of tracks occupied by the SYS1.LOGREC data set.
10. A one-byte code for the type of system resident device:

DEVICE	CODE
2311	X'01'
2301	X'02'
2302	X'04'

11. A five-byte expansion field.
12. A one-byte field of hexadecimal FF used to detect a previous overrun condition caused by a machine check or channel inboard failure while writing the header record.

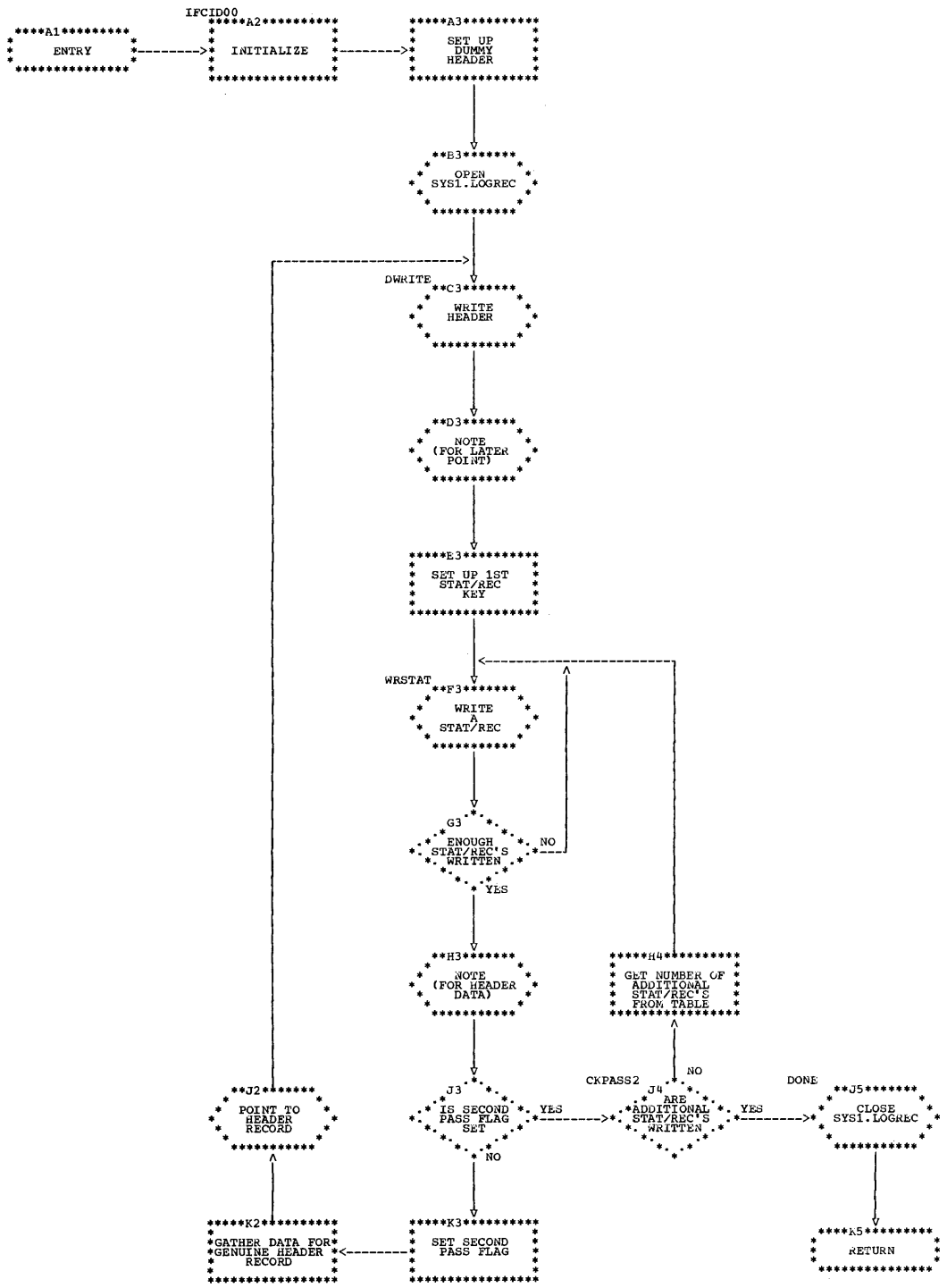
Statistical Data Records

This program writes each statistical data record with a 2-byte key field and a 38-byte data field of zeros.

Record Entry Area

This area begins on the first available track following the last track on which a statistical data record is written. Nothing is written in this area by this program. The address of this track is written by this program in field 4 of the header record.

Chart 13. IFCDIP00 - Initializing the SYS1.LOGREC Data Set



Editing and Printing Environmental Records (IFCEREP0)

The IFCEREP0 ("EREP") program edits and prints records from the SYS1.LOGREC data set. (These records were originally written by systems environment recording programs and provide the error environment of CPU, channel, and device errors.) EREP optionally saves certain SYS1.LOGREC error records on an accumulation (history) data set to provide comprehensive error statistics. The accumulation data set may then be used as input to EREP. Records from SYS1.LOGREC (except SDRs) or from an accumulation data set are printed in summarized form when the summary option is selected.

EREP operates in the problem state and is serially reusable. It consists of the following machine-independent modules:

- IFCEREP0 and IFCEREP1 (Charts 14-18), the control modules.
- IFCMSG00, the message module.
- IFCSDR00 (Chart 19), which edits statistical data records (SDRs).
- IFCOBR00 (Chart 20), which edits I/O outboard records (OBRs).
- IFCOBRSM, which edits the outboard record summary.
- IFCMCH00 (Charts 21 and 22), which performs preliminary editing of channel inboard records and machine check records.

Figure 25 illustrates the communication between modules of the EREP program. Figure 26 lists the machine-dependent modules of the EREP program.

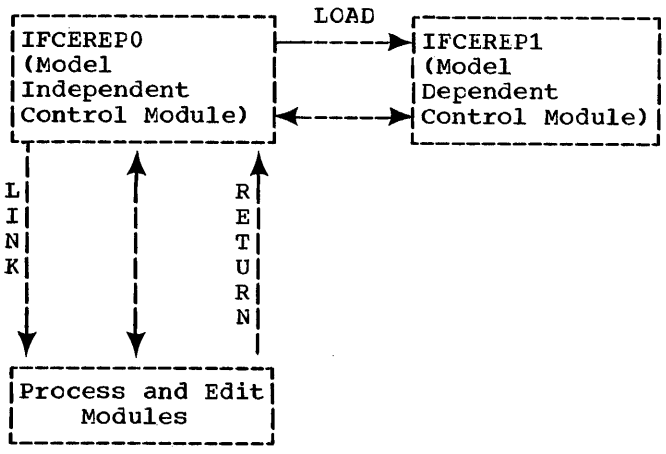


Figure 25. Control Flow Between Modules

All communication between record processing/editing modules and control modules is through IFCEREP0, the model-independent control module. This module then may communicate with IFCEREP1, which is model dependent, to handle summary/process/edit requests.

Overall Flow

The control module first scans and analyzes parameters from the execute statement, and then performs basic initialization, such as loading the second control module and the message module. Module IFCEREP0 also determines the input and output data sets to be used and opens their associated DCBs. When the parameters specify the summary option, the control module obtains via a GETMAIN macro instruction from 1.7 to 4K bytes of storage. (The size of obtained storage depends on the amount of free storage.)

The record-processing path determined by the control module depends on whether the input data set is SYS1.LOGREC or an accumulation data set. When the input is an accumulation data set, program flow also depends on whether a record data summary is requested. (The control modules indicate program flow by setting bits or bit-combinations in four switch bytes.)

SYS1.LOGREC Input

When SYS1.LOGREC is the input data set, EREP processes all records of a type before processing another type. The program reads a record from SYS1.LOGREC and the appropriate editing module is given control by means of the Link routine. When all records of the selected type have been read and the appropriate ones edited and written, the options are checked to see if another type of record is to be processed; if so, all the records of this type are read and the appropriate ones edited and written. For record types other than SDR, optioned summary and accumulation functions are performed before EREP begins processing another record type. (Unlike other record types, OBRs are written into the accumulation data set in blocks of ten. Space for record blocking is reserved in the message module.)

When all records of the selected types have been processed, the SYS1.LOGREC header record is checked by the control module to see if any additional records were stored in SYS1.LOGREC while EREP was processing. If any were stored, the program enters a second pass and the additional records are edited and written regardless of the options selected. No summary of these records is performed.

Accumulation Input

When an accumulation data set is the input, EREP minimizes the number of access cycles by processing more than one record type on a pass if a summary was not requested or if the maximum 4K bytes of storage for a requested summary was obtained from the GETMAIN routine.

If the summary was requested or maximum storage was unavailable, the program first edits and prints OBRs, if requested. Since available space may be insufficient for an OBR summary, more than one summary pass may be necessary. After OBR processing is complete, INB and machine check records are processed in a single pass each.

Control Module Subroutines

The control module and the editing modules make use of the following subroutines, located in the control module, to perform I/O operations:

MACHINE	MODULE ID	MODLIB ID (***)	FUNCTION
Model 40	IFCMC140	IFCEP400	Edits CPU records.
	IFCMC340	IFCEP401	Completes editing of CPU records.
	IFCSUM40	IFCEP104	Summarizes CPU and inboard records.
	IFCMCS40	IFCEP041	Edits the CPU records summary.
	IFCINS40	IFCEP072	Edits the inboard records summary.
Model 50	IFCMC150	IFCEP500	Edits CPU records.
	IFCMC250	IFCEP501	Completes editing of CPU records and edits inboard records.
	IFCSUM50	IFCEP105	Summarizes CPU inboard records.
	IFCHCS50	IFCEP051	Edits the CPU records summary.
	IFCINS50	IFCEP052	Edits the inboard records summary.
Model 65/67 (*)	IFCMC165	IFCEP650	Edits CPU records.
	IFCMC265	IFCEP752	Edits inboard records.
	IFCMC365	IFCEP651	Continues editing CPU records.
	IFCMC465	IFCEP652	Completes editing CPU records.
	IFCSUM65	IFCEP106	Summarizes CPU and inboard records.
	IFCMCS65	IFCEP061	Edits CPU records summary.
	IFCINS65	IFCEP072	Edits inboard records summary.
	IFCASR00(**)	IFCEP655	Edits machine check handler portion of CPU records.
	IFCASR01(**)	IFCEP656	Edits channel check handler portion of inboard records.
Model 75	IFCMC175	IFCEP751	Edits CPU records.
	IFCMC275	IFCEP752	Edits inboard records.
	IFCMC375	IFCEP753	Completes editing CPU records.
	IFCSUM75	IFCEP107	Summarizes CPU and inboard records.
	IFCMCS75	IFCEP071	Edits CPU records summary.
	IFCINS75	IFCEP072	Edits inboard records summary.
Model 91	IFCMC191	IFCEP950	Edits CPU records.
	IFCMC291	IFCEP952	Edits inboard records.
	IFCMC391	IFCEP951	Continues editing CPU records.
	IFCMC491	IFCEP953	Completes editing inboard records.
	IFCSUM91	IFCEP109	Summarizes CPU and inboard records.
	IFCMCS91	IFCEP091	Edits CPU records summary.
	IFCINS91	IFCEP072	Edits inboard records summary.

(*)Except for modules IFCASR00 and IFCASR01, all modules in this group have aliases of IFCxxx67, where xxx represent the fourth, fifth, and sixth characters in the module ID.
(**)These modules occur only in systems having the machine check handler and the channel check handler feature.
(***)This is the module identification before it is link-edited onto the Link Library.

Figure 26. EREP Machine-Dependent Modules

XWRTPRT
: writes edited data, using BSAM, on the specified output device. Records are written in 120-byte blocks from the buffer XPRTBUF, also in the control module.

XRDDISK
: reads, using EXCP, a record from SYS1.LOGREC into the buffer XDADBUF, also in the control module.

XWRTDISK
: writes, using EXCP, a record of zeros on SYS1.LOGREC. The buffer XDADBUF is zeroed out by the editing module. If disk writing is prohibited, this routine returns control immediately.

XWRTOP
: writes, using WTO, messages to the operator.

XACCSUM
: accumulates and summarizes records, if necessary.

Chart 14. IFCEREP0 Initialization and Linkage to Editing Modules

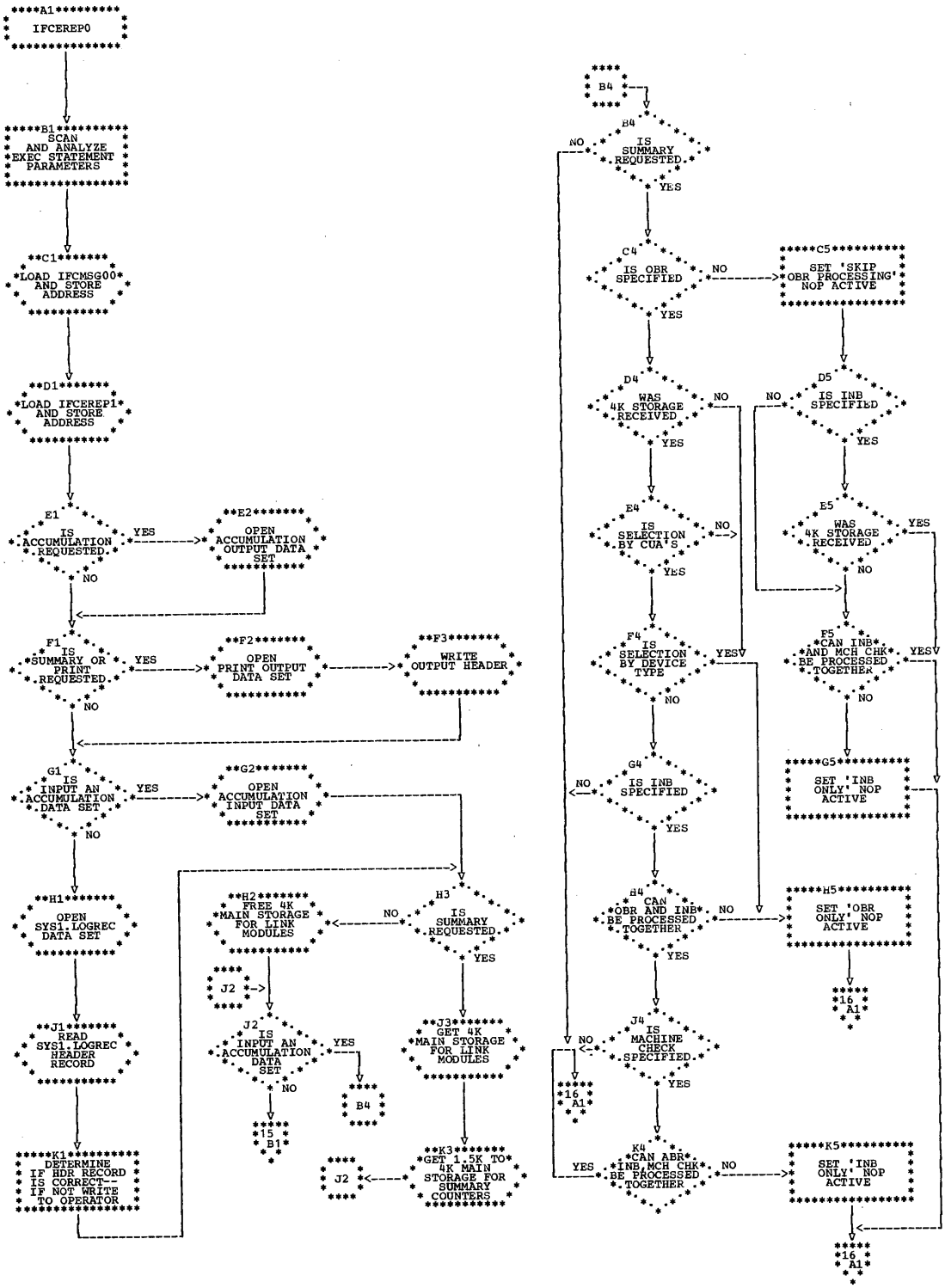


Chart 15. EREP - Input From SYS1.LOGREC Data Set

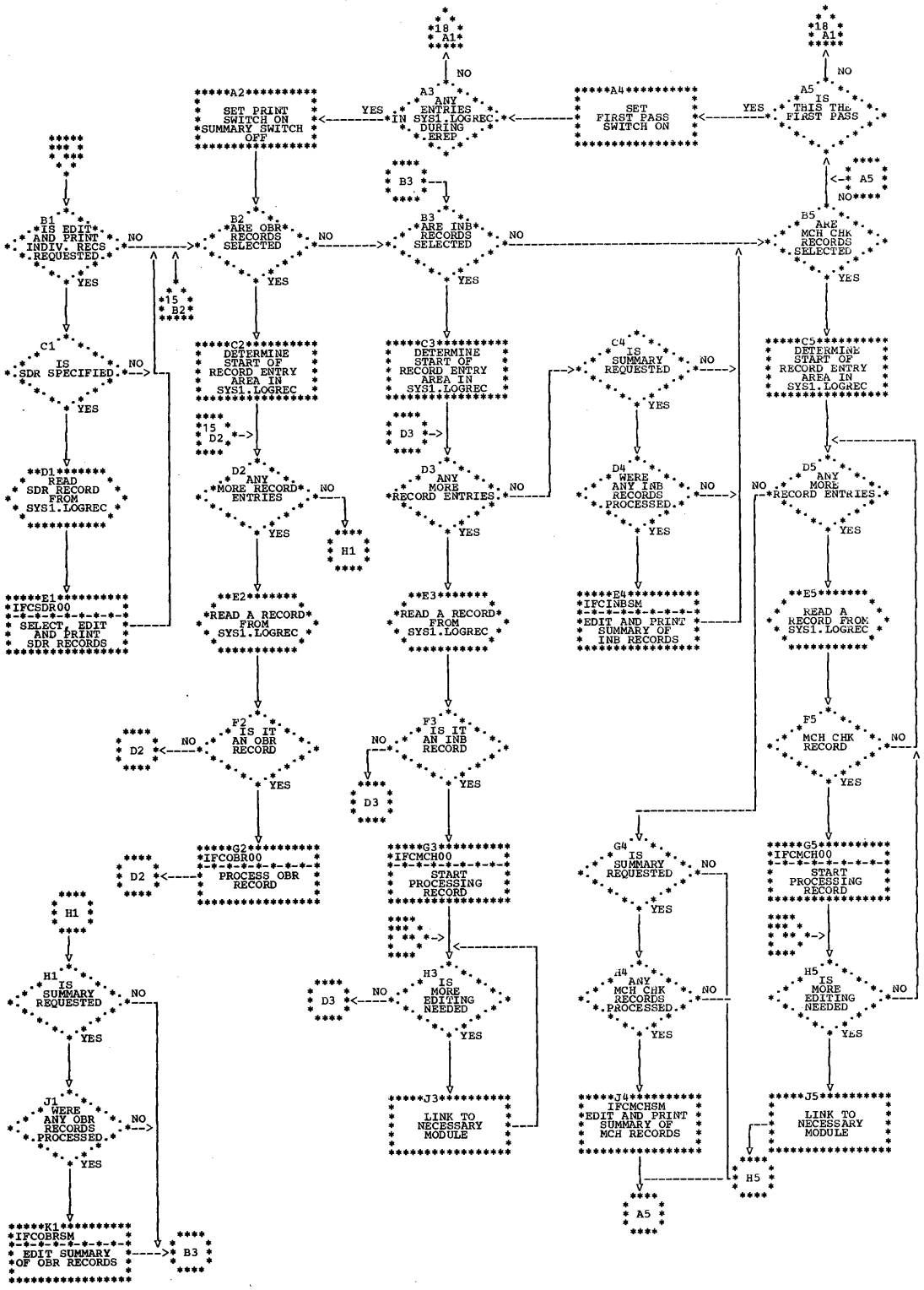


Chart 16. EREP - Input From Accumulation Data Set

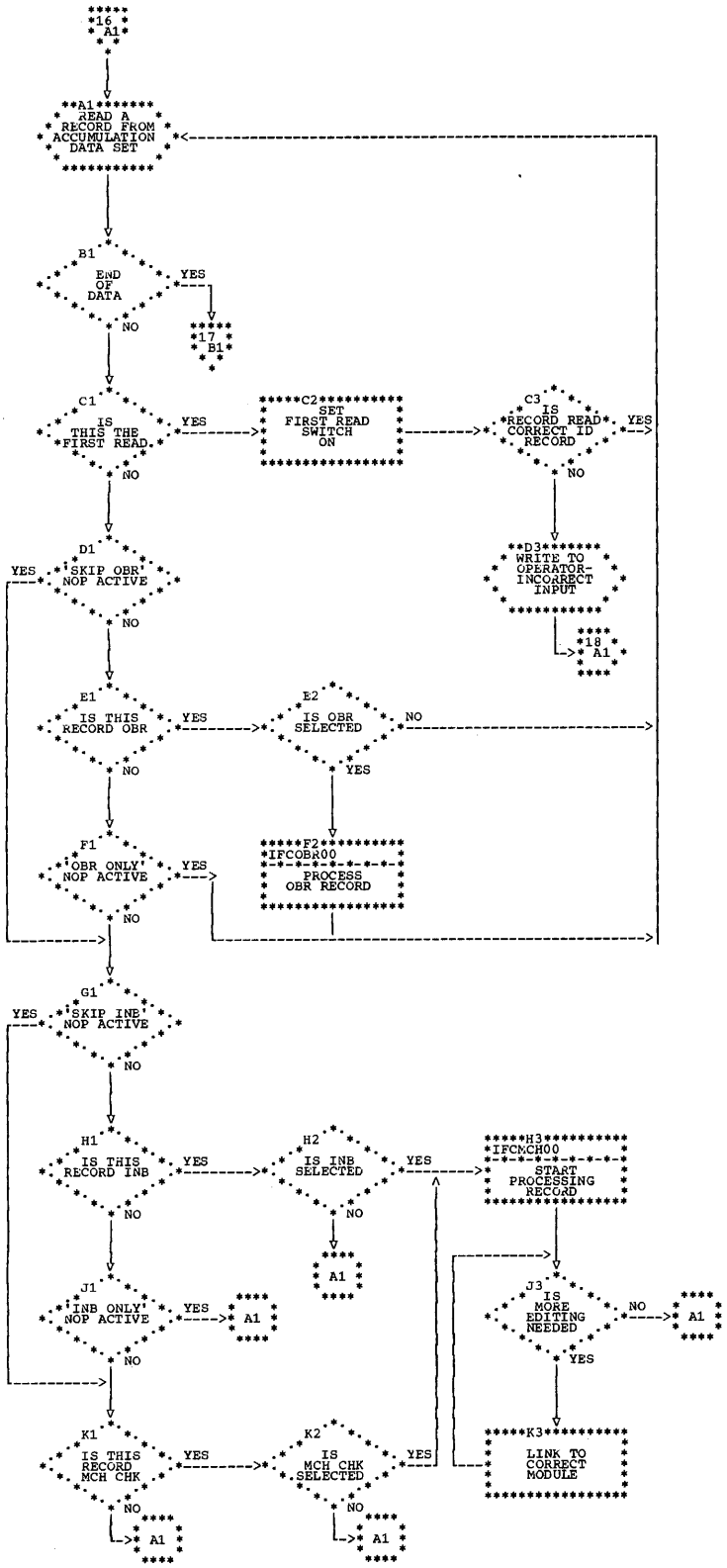


Chart 17. EREP - Accumulation Input - End of Data

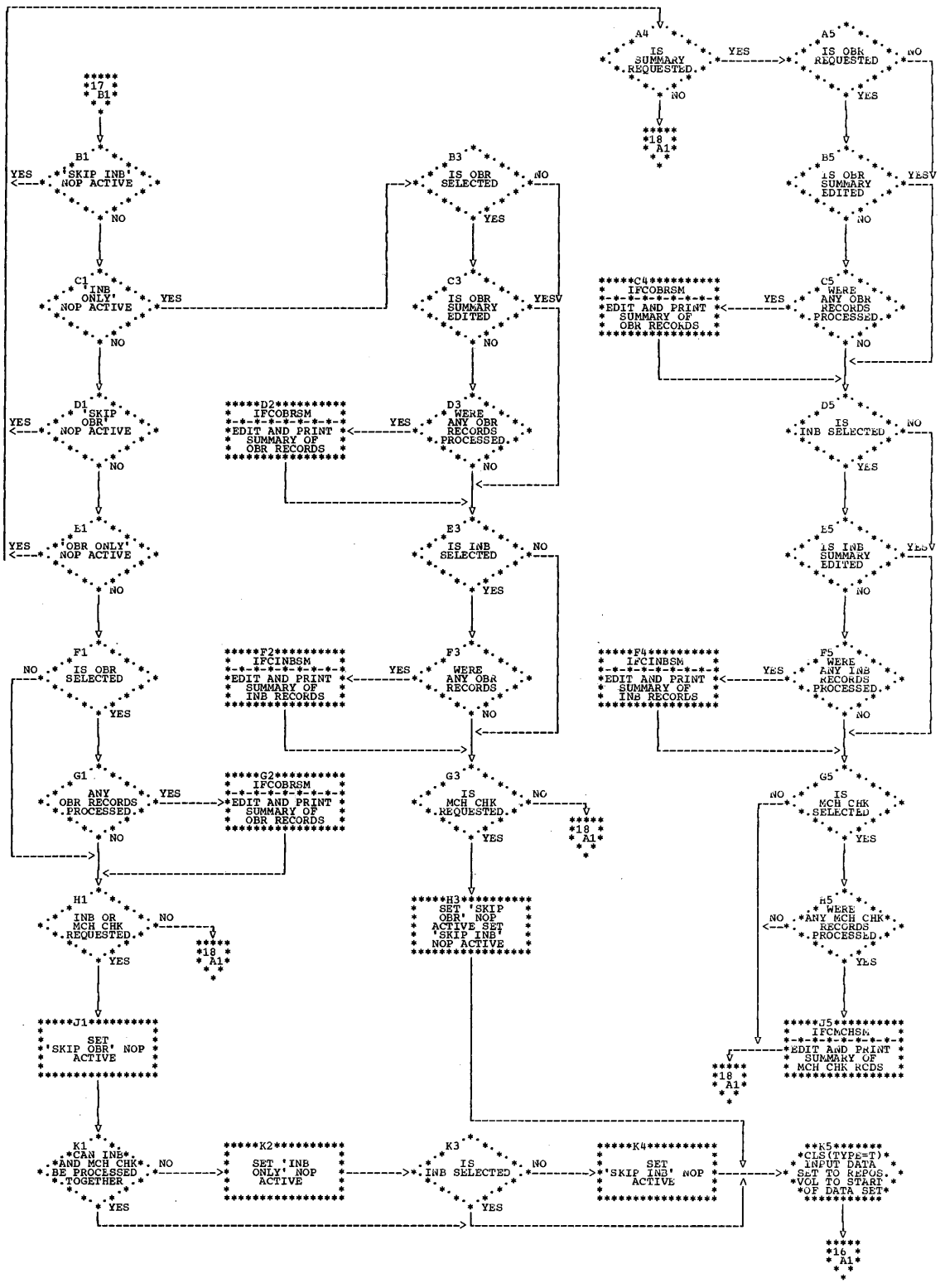


Chart 18. EREP Termination

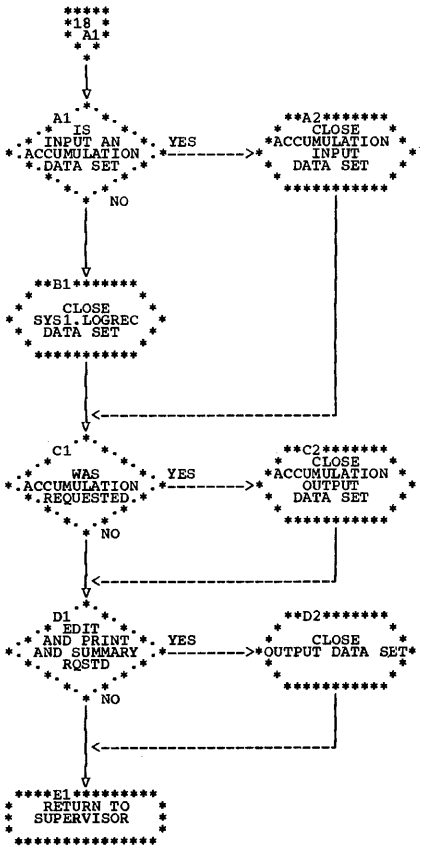


Chart 19. IFCSDR00 - Editing SDRs

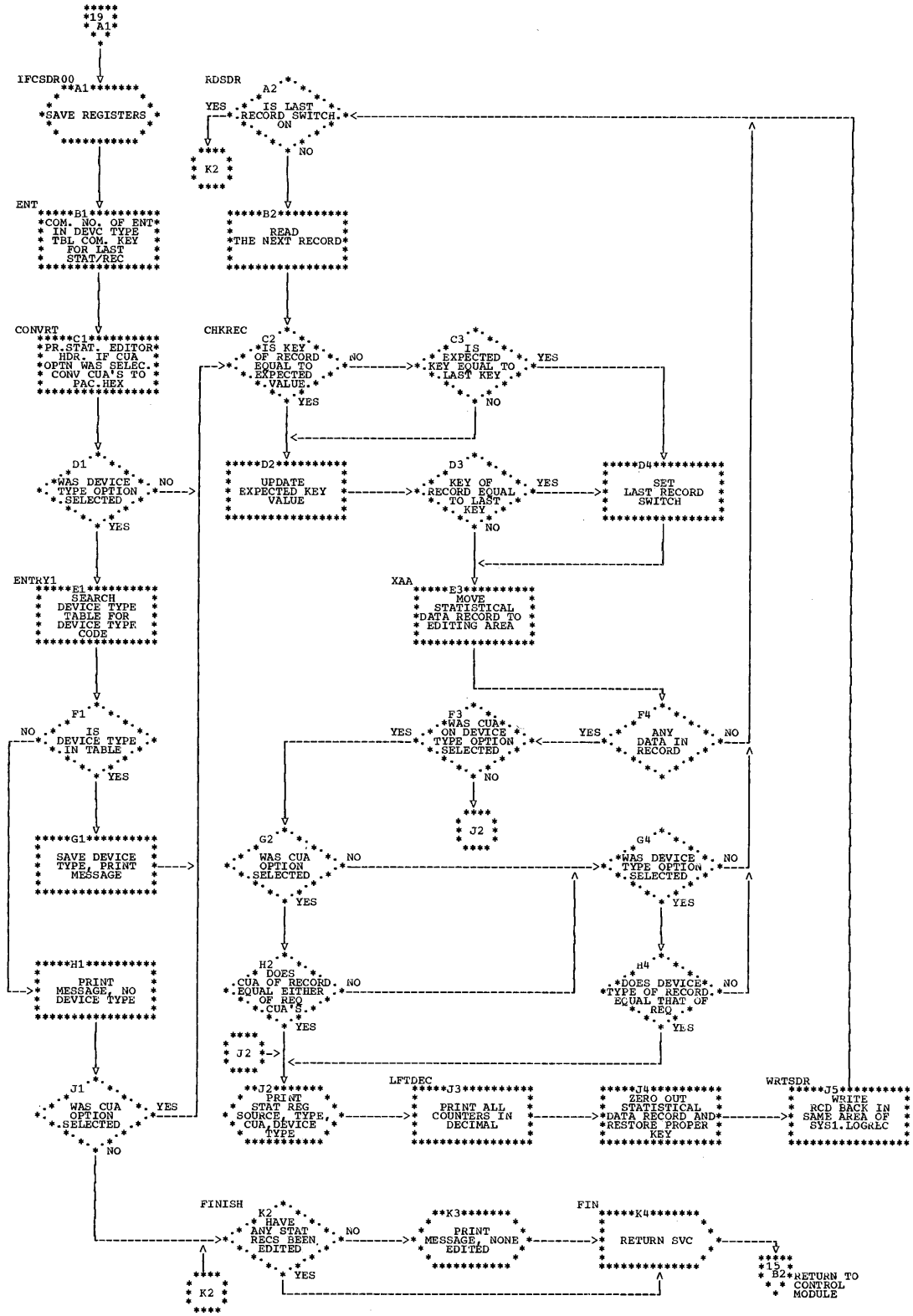


Chart 20. IFCOBR00 - Editing OBRs

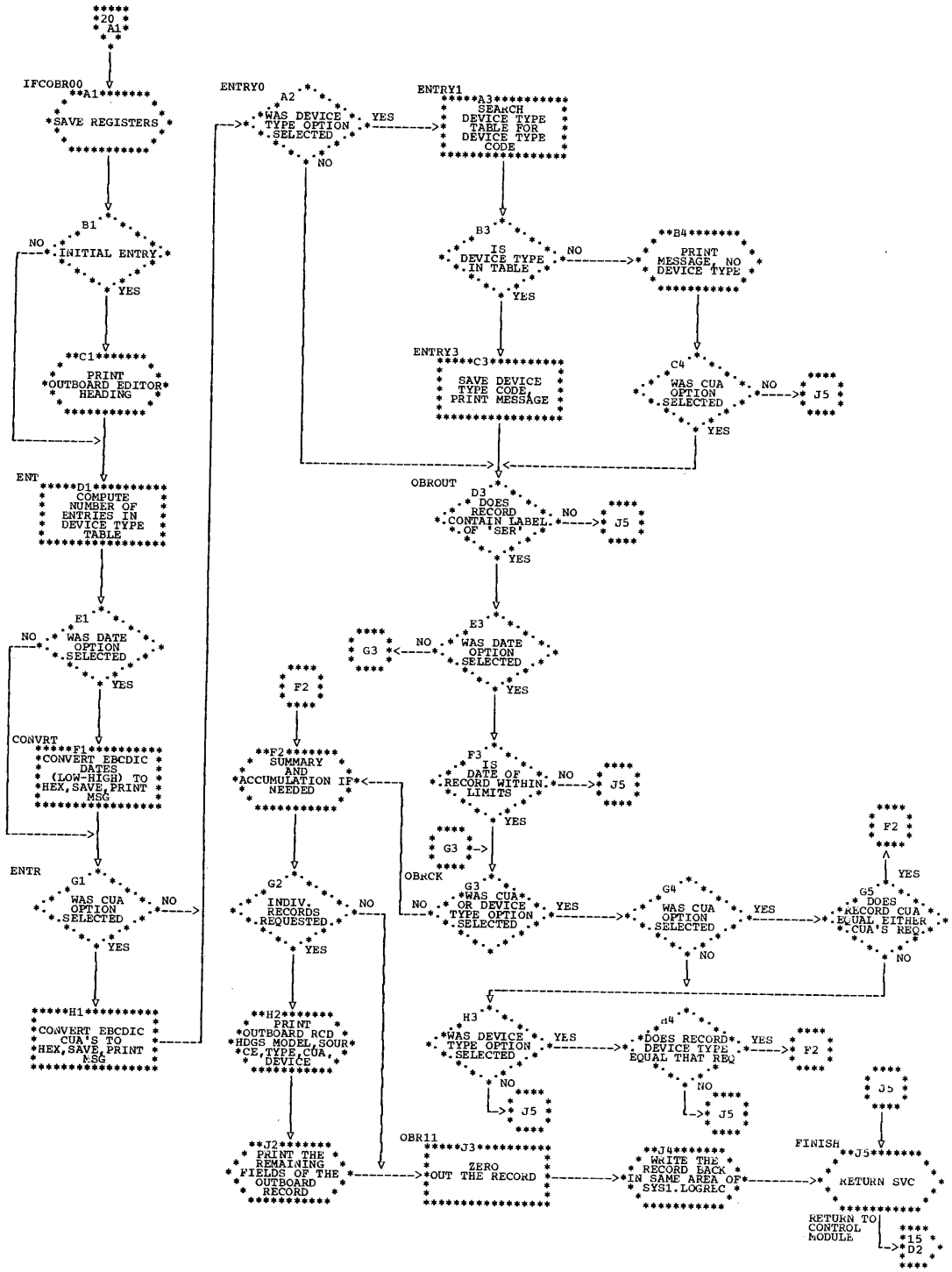


Chart 21. IFCMCH00 - Editing Inboard and CPU Records (Part 1 of 2)

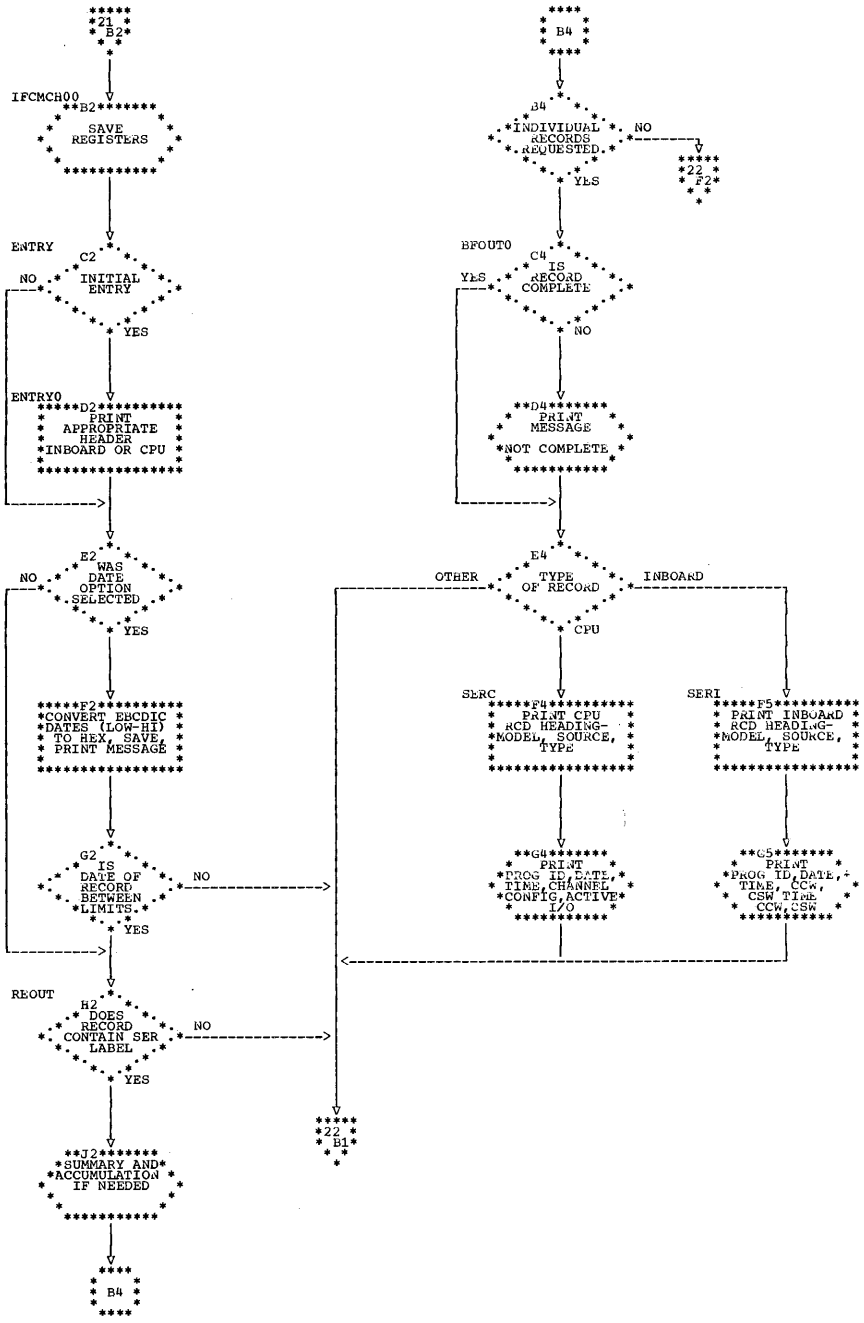
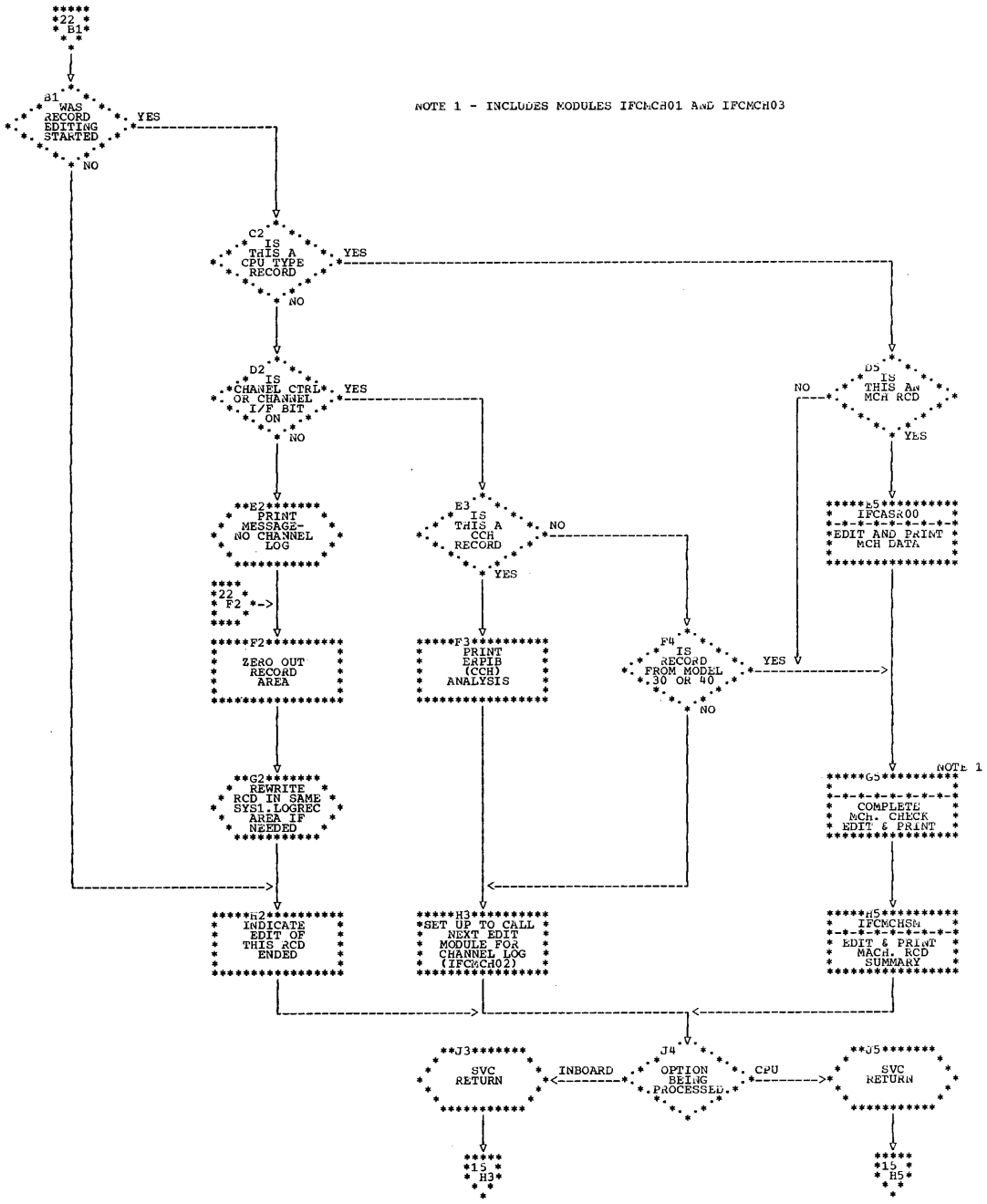


Chart 22. IFCMCH00 - Editing Inboard and CPU Records (Part 2 of 2)



Loading the 2821 Generator Storage (IEHUCSLD)

The IEHUCSLD program reads records that contain user-specified character images, requests the operator to change the print chain or train, loads the images into 2821 generator storage, and prints the images so that the operator can verify the operation. Options allow the user to specify folding or non-folding mode, permit him to use non-standard ddnames and to bypass the verification procedure.

The IEHUCSLD program may be executed as an independent job step or it may be entered via the LINK or ATTACH macro instruction. In either case the user may specify alternate ddnames and bypass verification procedures. Program flow is shown in Chart 23.

PROGRAM FLOW

When IEHUCSLD is given control it examines the parameter list to determine which (if any) option has been specified. If no option has been specified it assumes the VERIFY option.

The next step is to determine whether an alternate ddanme is specified for either the input or printer data set. If an alternate name is specified, IEHUCSLD moves the specified name to the DCB; otherwise it moves the standard names.

The program then initializes the printer DCB for use with the EXCP macro instruction, and opens the input and printer DCBs. It checks to see that both DCBs are properly open, then inspects the printer UCB to insure that the universal character set feature is available.

If either DCB is not properly open, or if the universal character set feature is not available on the requested printer, the ddname specification (or other information in the DD statement) is incorrect. In either case, IEHUCSLD closes both DCBs and returns with a return code of 8.

If both DCBs are properly open and the universal character set feature is available, the IEHUCSLD program copies the printer unit name from the UCB into the operator message and print line texts, and prepares to read the four control records.

IEHUCSLD uses the Read routine four times to bring the control records into main storage. When the first record has been read, there is some initial processing done before the normal processing takes place.

The initial processing includes a check for an asterisk in position 1 and a comparison of the two type ID fields. The type ID is then copied into the operator message and print line texts, the mode option field is inspected, and the printer CCW is initialized (to folding or non-folding mode) accordingly.

The normal processing is done for all four records. The images field is moved to an internal buffer, the record is sequence checked and its format is verified. Then, unless four records have been read, a branch is executed to the expansion of the READ macro instruction.

If it finds an error in a control record, IEHUCSLD uses the WTO macro instruction to issue message IEH503I, the control card error message. It closes the DCBs, loads return code 8, and returns.

When IEHUCSLD has processed all four records, it closes the input DCB and checks for the LOADONLY option. If the LOADONLY option is specified, the program branches to the EXCP macro expansion; otherwise it requests the operator to change the print chain or train. It issues message IEH500A and waits for the operator to reply with the type ID or 'SKIP'.

If the operator replies 'SKIP', the IEHUCSLD program issues the no action message, IEH506I, closes the printer DCB and returns with code 0.

If the reply specifies the type ID requested, IEHUCSLD uses the EXCP macro instruction to load the character images into 2821 generator storage, and waits for completion of the channel program.

When completion of the channel program is posted in the ECB, the IEHUCSLD program inspects the completion code bits to determine whether a permanent error has occured. If so, and the error is a parity error, IEHUCSLD closes and reopens the printer DCB and retries the channel program.

If the error is a permanent error, but not a parity error, the program closes the printer DCB and returns with code 12.

If the error is not a permanent error, but completion is not normal, or if the retry fails, IEHUCSLD closes the printer DCB and returns with code 12.

If the channel program is successfully completed, the IEHUCSLD program closes the printer DCB and checks for the LOADONLY or NOVERIFY option. If either option is specified, the program writes message IEH502I to the operator to tell him that the images

have been loaded, issues return code 0 and returns.

If neither the LOADONLY or NOVERIFY option is specified, IEHUCSLD opens the printer DCB for BSAM. It skips the printer to the next page and prints a header line that specifies the unit, type ID, and mode (folding or non-folding). Then IEHUCSLD spaces two lines and prints two 120 character lines to display the images it has loaded into the 2821 generator storage.

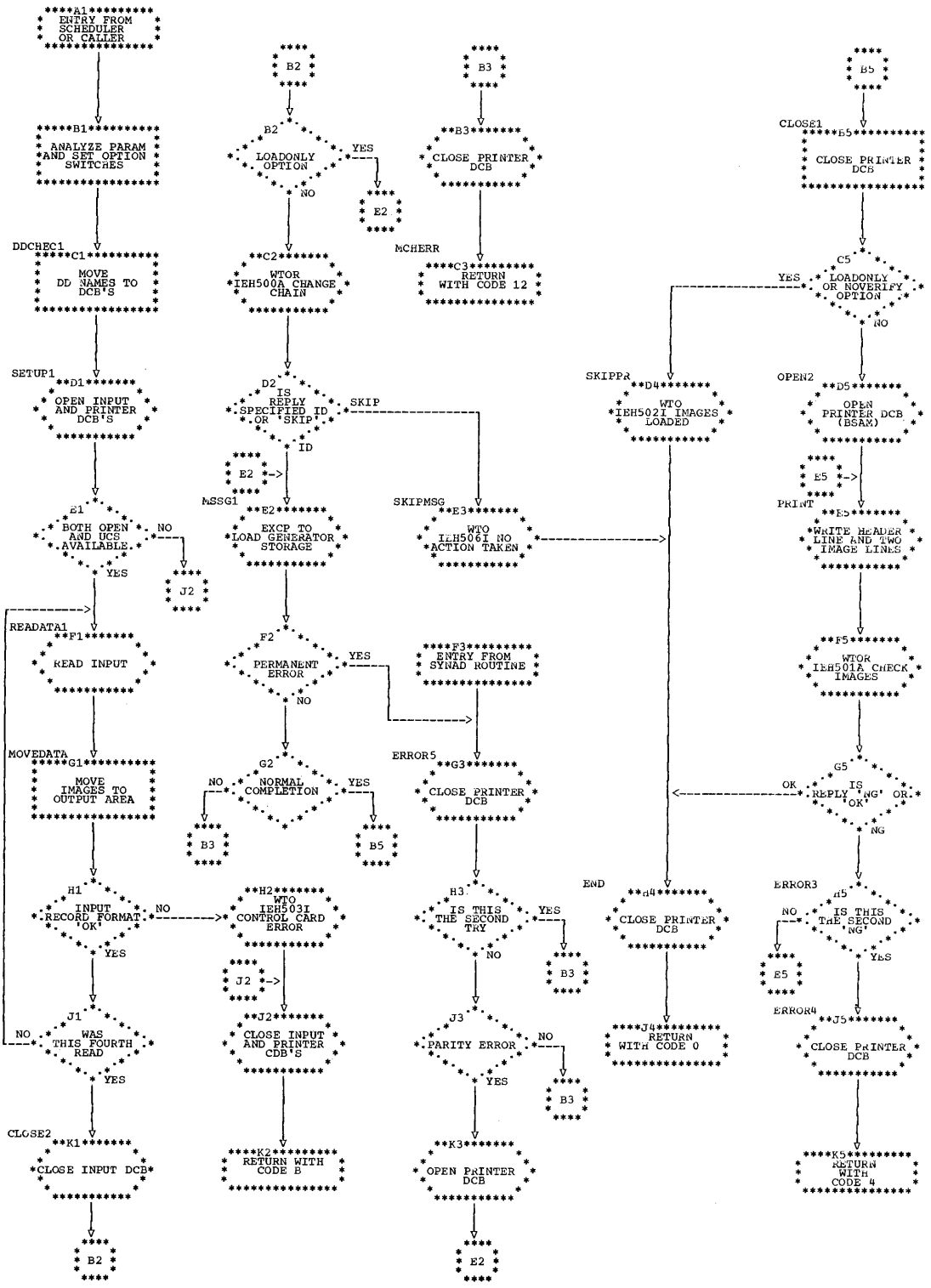
If the header line requires images that were not supplied by the user, and the reset block data check mode is specified in the printer DD statement, the IEHUCSLD program does not space two lines after the header. If the user does not specify reset block data check mode in his printer DD statement, the space will occur; in either case the images that were not supplied will print as blanks.

When the three lines have been printed, IEHUCSLD skips the printer to the next page and tells the operator to check the images, using message IEH501A.

The operator must reply, 'OK' or 'NG'. If the reply is 'NG' the images are printed once more, and the operator is again requested to check the images. A second 'NG' reply causes the program to close the printer DCB and return with code 4.

If the reply is 'OK', IEHUCSLD closes the printer DCB, loads return code 0, and returns.

Chart 23. IEHUCSLD - Loading the 2821 Generator Storage



Writing Tape Labels (IEHINITT)

The IEHINITT program provides the user with a convenient means of writing volume label sets on tapes to conform to Operating System/360 specifications. The program reads control cards, builds a parameter list, calls an SVC routine to write a tape volume label set, and informs the user of the result of the labeling attempt.

Program Flow

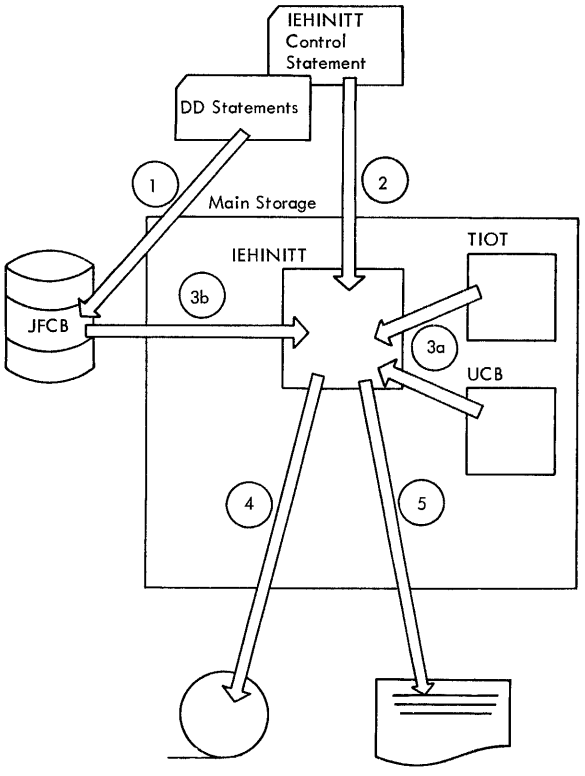
The general flow of the program and its relationship to the operating system are shown in Figure 27. Charts 24 and 25 show more detailed flow. Chart 26 shows the logic of SVC 39, the tape-labeling SVC routine.

Program Structure

The program consists of four modules: IEHINITT, the control module; IGC0003I, the SVC 39 routine; IEHSCAN, the control statement scan routine; and IEHPRNT, the message writer.

The Control Module (IEHINITT): The control module builds two DCBs (SYSIN and SYSOUT) for the tape-labeling operation and moves the DCBs to the work area. It then links to the message writer (IEHPRNT) to write a header, and links to the control statement scan (IEHSCAN) to read a control statement into main storage. IEHPRNT then prints the control statement. Control cycles between IEHINITT and IEHSCAN until the parameters are analyzed or an error is detected. If there are no errors, IEHINITT builds an image of the tape label in main storage, and then builds a parameter list for the tape-labeling SVC by referring to the JFCB, TIOT, and UCBs for DD statement information. The symbolic link needed by the program to gain access to this information is the ddname, supplied in both the DD statement and the utility control statement. IEHINITT then issues the SVC 39, invoking the tape-labeling routine. When control is returned, IEHINITT analyzes the return code and links to IEHPRNT to print the label or an error message. The process of building the parameter list, issuing the SVC, and interpreting the return code is repeated for each tape to be labeled. When the last tape has been labeled, IEHINITT returns control to the supervisor.

The SVC 39 Routine (IGC0003I): The SVC routine writes the specified volume label, a dummy header label (HDR1 followed by 76 EBCDIC zeros), and a tapemark on a desig-



1

Before IEHINITT has gained control, information from the data definition (DD) statements has been entered in the task I/O table (TIOT) and job file control blocks (JFCB)

2

IEHINITT gains control and reads a control statement. The dd name from the control statement indicates which collection of tape drives to use from the TIOT

3

A drive is selected and its relative position in the TIOT is described in the parameter list for the tape-labelling SVC. The parameter list is built by extracting:

a) the device type (dual-density, 7-track, or 9-track) from the UCB
b) the density for dual-density or 7-track from the JFCB

4

The SVC is issued and the tape label is written

5

The return from the SVC is analyzed and the result is logged. If the request being processed shows more tapes to be labeled, go to (3). If another control statement is to be read, go to (2).

Figure 27. Writing Tape Labels

nated drive. By issuing a GETMAIN, the routine obtains 204 + X bytes, where X is the amount by which the volume label exceeds the standard length of 80 bytes. This area is used for building a DCB, DEB, ECB, IOB, and a channel program, and also holds messages and labels. Upon entry to the SVC routine, register one contains the address of a 4-word parameter list:

Word	Bytes	Contents
0	0-1	X'C000'
	2	X'04' to rewind tape X'06' to unload tape
	3	a binary number from 0 to n-1, where n is the number of UCB addresses in the DD entry portion of the TIOT indicating which device to use for volume mounting
1	0-3	a pointer to an 8-byte area containing the ddname corresponding to the ddname in the DD entry portion of the TIOT; the ddname is left-justified and padded with blanks
2	0-3	a pointer to one volume label image to be written on the tape
3	0-1	the binary length of a volume label
	2	the binary number one
	3	command code for the control CCW to set mode

The SVC routine extracts the UCB address from the DD entry portion of the TIOT. This UCB is checked to verify that the tape is not SYSIN or SYSOUT, that the tape is online and not scheduled to go offline, that the tape is not reserved, and that the data management count is zero. If a tape is already mounted on the device and its volume serial number is in the UCB, it is unloaded. After the volume label has been written and verified, the dummy header label and tape mark are written. If the tape is not to be unloaded, its volume serial number is left in the UCB. If a non-standard label was written, the pseudo volume serial number LGL000 is left in the UCB. If an I/O error or a file-protected tape is encountered in the labeling process, the operator is given one attempt to correct the situation. (He may strip off a few feet of tape or add the file protect ring.) When returning control to IEHINITT, the SVC routine issues a FREEMAIN to free the work area, and loads register 15 with one of the following return codes:

Code	Meaning
00	labeling successful
04	operator has cancelled labeling
08	unacceptable parameter list
0C	permanent I/O error

The Control Statement Scan Routine (IEHSCAN): This routine reads, using QSAM, control statements, checks syntax, and returns to IEHINITT an indication of the item scanned. IEHINITT supplies a work area (on a fullword boundary) containing the DCB for the control statement data set, which is opened by IEHINITT before calling IEHSCAN. IEHSCAN inserts the address of the end-of-file routine KEOF in the DCB and the EOF routine for IEHINITT is restored when control is returned to IEHINITT. After scanning a field from the control statement, IEHSCAN returns to IEHINITT the following information:

- Register 1 points to the starting address of the field.
- Register 2 contains the length of the field.
- A setting of a byte, SWITCHRD, in the work area, as follows:

Bit=1	Meaning
0	Syntax error
1	Bypass switch
2	EOF
3	Initial entry
4	Command word
5	Keyword
6	Parameter
7	Not used

Unlike other control statement fields, the owner name field (when enclosed in apostrophes) is moved from the control statement image to the label image by the control statement scan routine. The owner name is considered to begin at the first byte following the first apostrophe; two consecutive apostrophes are considered a single embedded apostrophe and counted as one byte of a maximum of ten for the field. The scan is terminated when the count is exceeded or when a single (i.e., not followed immediately by another) apostrophe is encountered.

The Message Writer (IEHPRNT): This routine writes, using QSAM, page numbers, headings, and messages. Upon entry to the message writer, register 3 contains the address of the message minus one. If a permanent I/O error is detected in writing the message, the one-byte switch SWITCH2 is set to X'01' before control is returned and a code of 4 is returned to IEHINITT in register 15.

Chart 24. IEHINITT (Part 1 of 2)

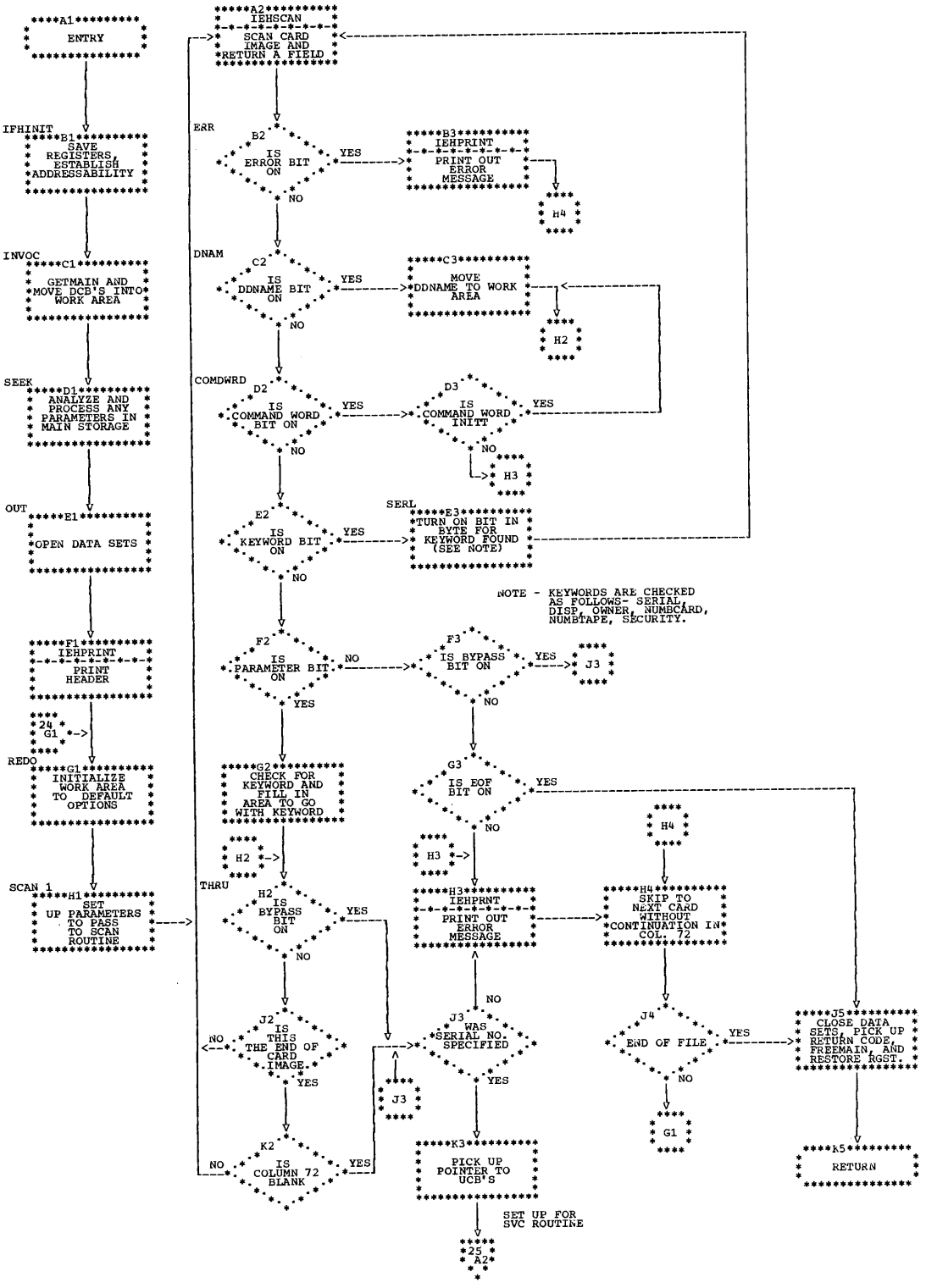


Chart 25. IEHINITT (Part 2 of 2)

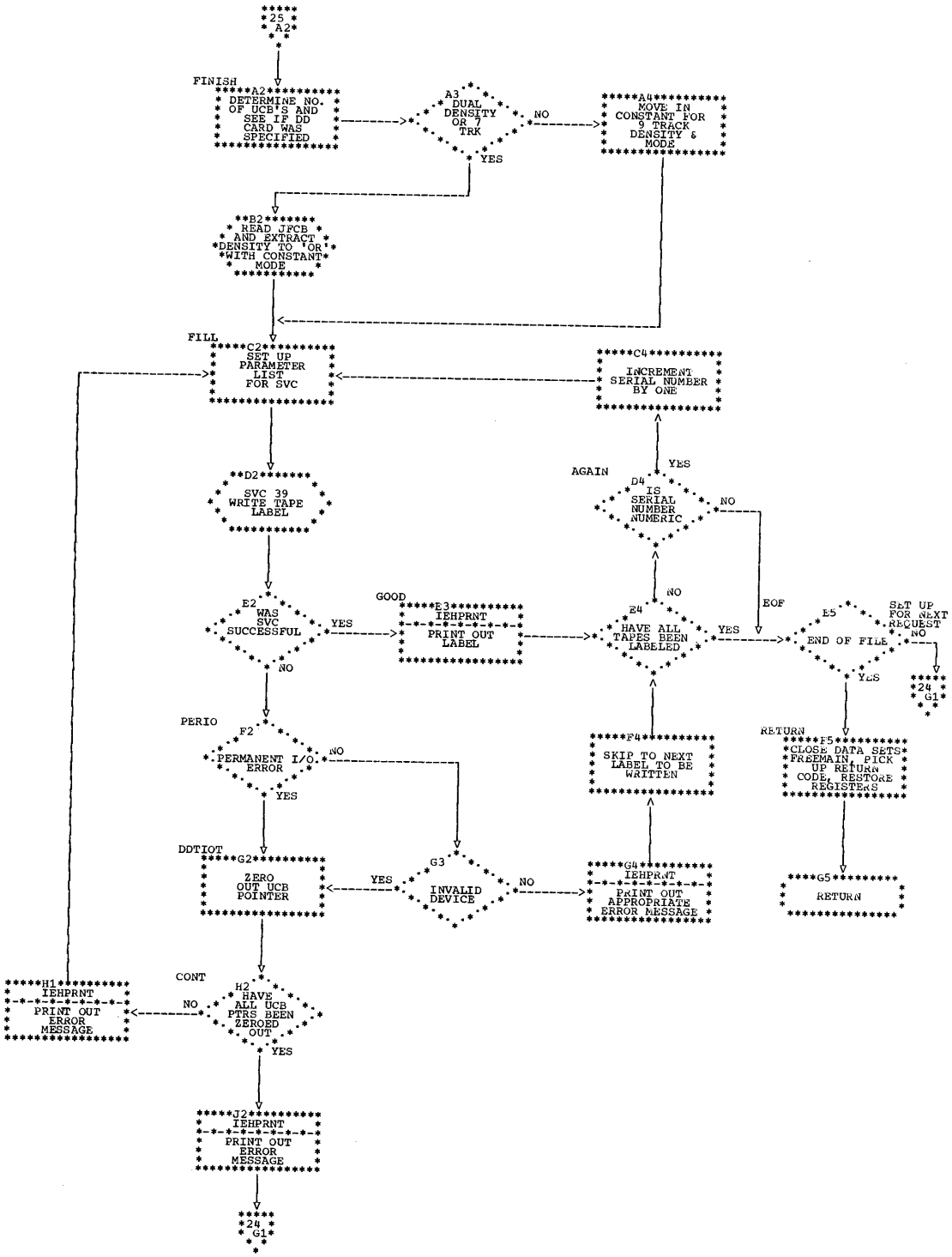
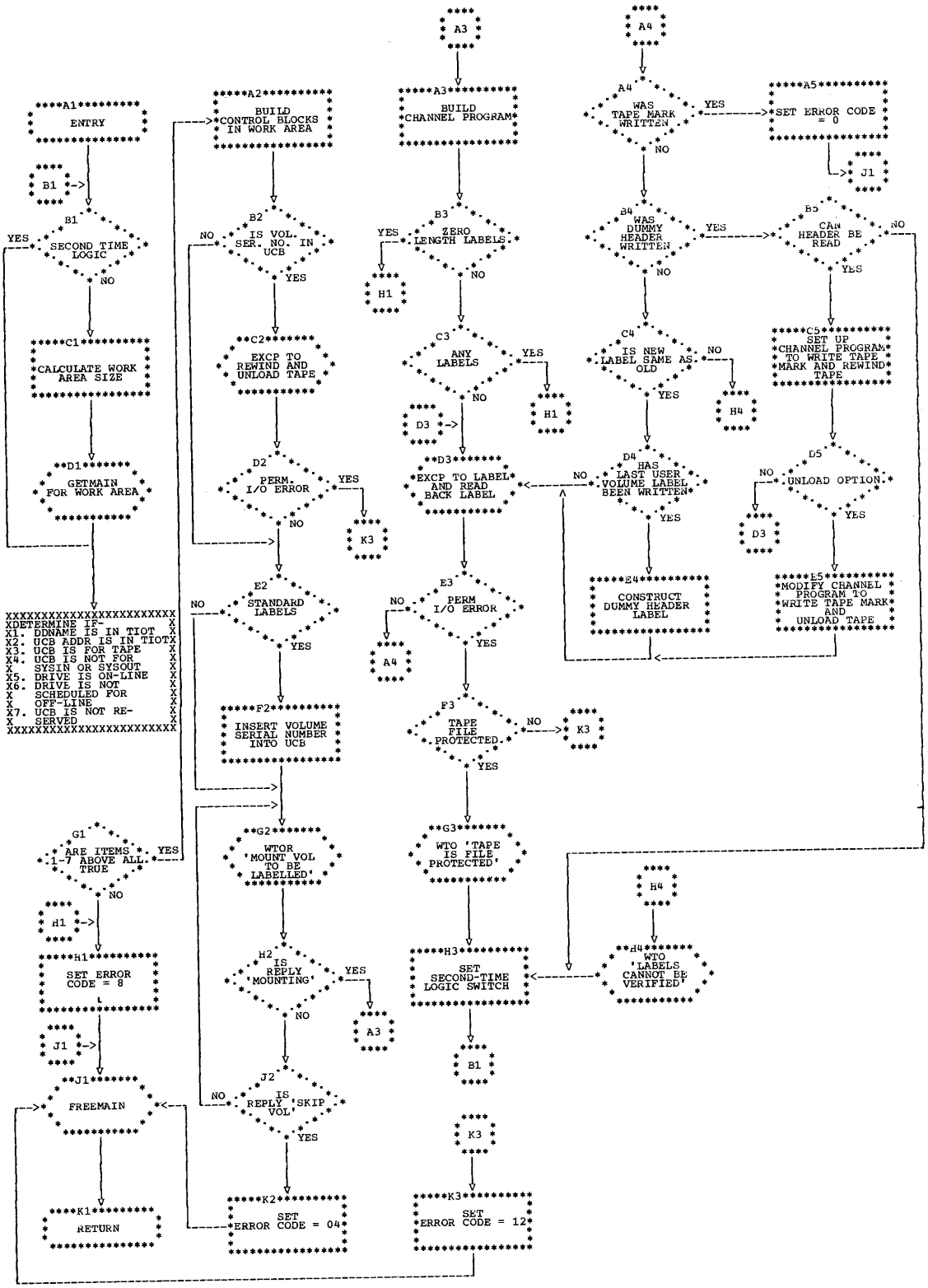


Chart 26. SVC 39 Tape Label Routine



Dumping, Restoring, and Initializing Direct Access Volumes (IEHDASDR)

The IEHDASDR program dumps, restores, and initializes direct access volumes according to parameters specified in control statements. The functions that may be specified are:

- Dump. When the DUMP operation is specified, the IEHDASDR program creates a copy (or copies) of the direct access volume on one or more tape or direct access volumes, or as a system output data set.

- Restore. When the RESTORE operation is specified, the program copies "dumped" data from a tape volume to one or more direct access volumes, thus making one or more copies of the dumped volume.

- Initialize. There are four initializing functions that may be specified:

  1. Specifying ANALYZE causes the program to perform a complete initialization of one or more direct access volumes. The program performs a surface analysis by inspecting each volume for defective tracks, it obtains alternate tracks for all defective tracks, it formats acceptable tracks, and it constructs a volume label, volume table of contents (VTOC), and (optionally) an IPL program for each volume.

  2. Specifying FORMAT causes the program to perform all of the initializing functions (except surface analysis) for one or more volumes.

  3. Specifying LABEL causes the program to write a new volume serial (and optionally an owner name) on a direct access volume.

  4. Specifying GETALT causes the program to assign an alternate for the specified disk or data cell track.

The user specifies the functions to be performed by writing control statements and placing them in the input stream data set. He must also supply DD statements defining the data sets, devices, and volumes required for the program, and may also specify program parameters either in the EXEC statement PARM field or in a parameter area (see the section "Auxiliary Parameters" in this publication).

The IEHDASDR program can perform certain functions concurrently on several volumes of the same type. The user can specify more than one volume in a DUMP, RESTORE, ANALYZE, or FORMAT statement; the program processes the volumes concurrently in the sense that I/O operations are overlapped. This type of concurrent processing is known as "making copies"; in the case of a dump or restore there can be only one input volume, and the output volumes are copies of one another. In the case of an analysis or format, all volumes specified in the control statement are processed the same way, and if a new serial is specified all are given the same volume serial. In either case, the program uses only one set of buffers and internal tables.

The IEHDASDR program can also perform a Dump, Restore, Analyze or Format function concurrently on several volumes which may be of different types. The user specifies the same operation (e.g. DUMP,) on several successive control statements; if enough main storage is available for buffers and internal tables (a set is required for each statement), and if enough I/O devices are available, the volumes will be processed concurrently. Concurrent in this sense (and as it is used in the remainder of this section) means that a processing routine will be reentered to process a different set of volumes when it waits for the completion of certain I/O operations, as well as when its processing of one set of volumes is completed.

The IEHDASDR program may be executed as a job step, or it may be executed as a part of a program performing a job step. The user invokes the program by using IEHDASDR as the program name parameter in an EXEC statement, or by using it in the operand of an LINK or ATTACH macro instruction. The IEHDASDR program, which consists of an initialization routine, a control routine and a set of functional routines, is entered at the Initialization routine (module IEHDASDR). The Initialization routine obtains main storage for the common work area (Figure 28), initializes it according to any parameters passed from the caller, then uses the XCTL macro instruction to pass control to the Control routine (module IEHDASDS). When it has performed the specified functions, the Control routine returns control to the caller.

The Control Routine (IEHDASDS)

The Control routine is entered via an XCTL or ATTACH macro instruction issued in the Initialization routine. The Control rou-

tine uses the Scan routine to read control statements, and based on the specifications in the statements, the Control routine passes control to the appropriate functional routine. (Control flow among the modules of the IEHDASDR program is shown in Chart 27.) When all statements have been processed, the Control routine issues a RETURN macro instruction.

Initialization

When the Control routine (Charts 28 and 29) is entered, it uses the OPEN (type J) macro instruction to open the SYSIN (control) and SYSOUT (message) data sets. It uses the LINK macro instruction to pass control to the Print routine (module IEHDPRINT) which places a header record in the message data set, then uses the LINK macro instruction to pass control to the Scan routine (module IEHDSCAN).

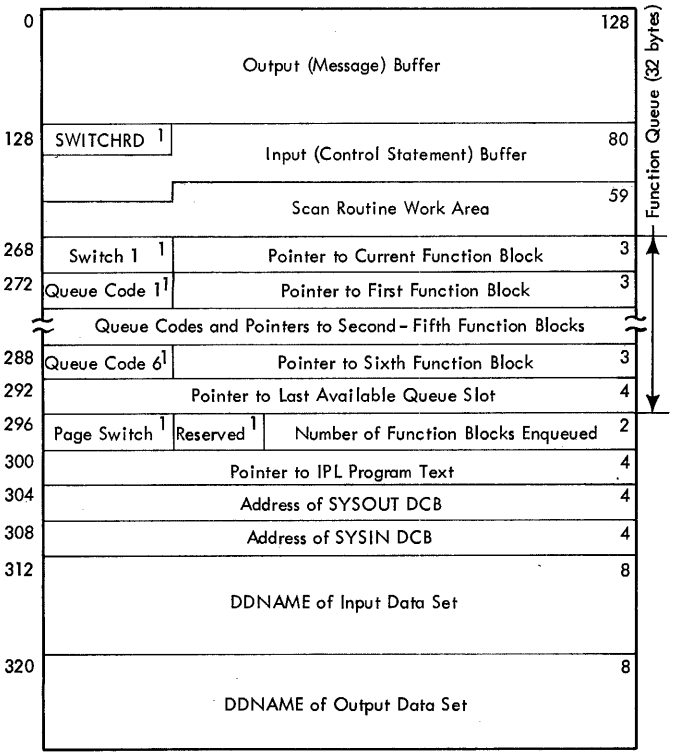


Figure 28. IEHDASDR Common Work Area

Notes: The common work area resides in an area of main storage obtained via a GETMAIN macro instruction in the Initialization routine (module IEHDASDR). Although the names of most fields are self-explanatory, the following fields require further description:

- SWITCHRD indicates the result of scanning a field of a control statement. When set to 1, the bits have the following meanings:

  Bit 0 Syntax Error

  Bit 1 Bypass Switch

  Bit 2 End-of-Data, SYSIN Data Set

  Bit 3 Initial Entry

  Bit 4 Operation Field

  Bit 5 Keyword Field

  Bit 6 Parameter Field

  Bit 7 Reserved

- Switch 1 indicates the status of the function queue. The bits have the following meanings when set to 1:

  Bit 0 Reserved

  Bit 1 Parameter processed

  Bit 2 Multiple parameter possible

  Bit 3 Looking for IPL text

  Bit 4 Reserved

  Bit 5 TODD=cuu

  Bit 6 Concurrent processing

  Bit 7 Looking for operation field

- Queue Code indicates the status of the function block. The bits have the following meanings when set to 1:

  Bit 0 Entry active (this slot not available)

  Bit 1 Processing complete

  Bit 2 Processing includes copies

  Bit 3 Processing interrupted

  Bit 4 Processing started

  Bit 5 Reserved

  Bit 6 Reserved

  Bit 7 No main storage available

Processing and Control

The Control routine uses a scan routine to read and check the syntax of the control statements. Each time the Scan routine is

entered it checks one field; on the return, the Control routine validates the scanned field.

If either the Control routine or the Scan routine encounters an error, the Control routine places a message in the message data set, and starts to scan the next control statement.

If the operation field (which specifies the function to be performed) is valid, the Control routine obtains main storage and constructs a function block (Figure 29). The function block specifies the function to be performed on a set of volumes, specifies the volumes, and contains Control information. If the statement specifies multiple volumes, the Control routine constructs a copy block (Figure 30) for each additional volume. The copy blocks are chained to the function block; they contain specifications for the additional volumes in the set.

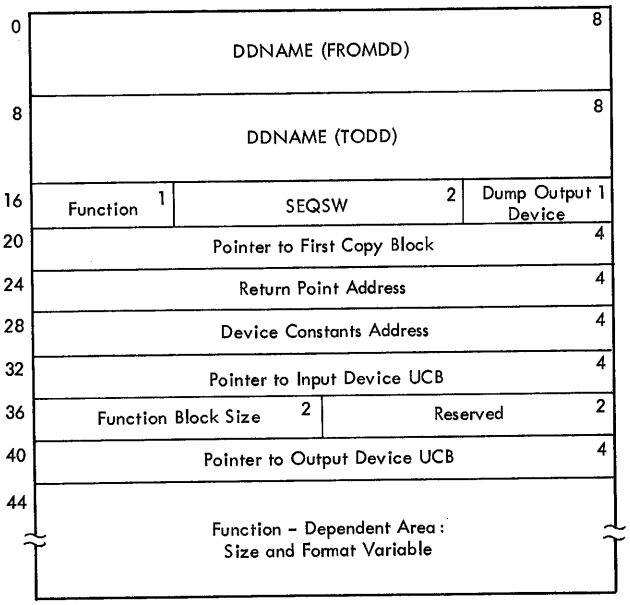


Figure 29. IEHDASDR Function Block

Notes: A function block is created, and enqueued in the function queue, each time the Control routine processes a control statement. The function block, which contains the information necessary to perform the function, is dequeued (and its main storage released) when performance of the function is terminated.

Although the names of most of the fields of the function block are self-explanatory, the following fields require further explanation:

- Function is a 1-byte field containing a code that represents the function to be performed. The codes (in hexadecimal) are:

DUMP	10
RESTORE	20
GETALT	30
LABEL	40
ANALYZE	50
FORMAT	60

- SEQSW is a 2-byte field that indicates which keywords were present in the control statement. If a bit is on, its meaning is as described below:

  Byte 1: Bit 0: FROMDD, TRACK, NEWVOLID
  Bit 1: TODD
  Bit 2: CPYVOLID, EXTENT
  Bit 3: BEGIN, VTOC
  Bit 4: END, IPLDD
  Bit 5: OWNERID
  Bit 6: FLAGTEST
  Bit 7: PASSES

  Byte 2: Bit 0: PURGE
  Bits 1-7: Reserved

- Dump Output is a 1-byte field used during the performance of the DUMP function to indicate the type of output device. The codes (in hexadecimal) are:

Tape	00
System Output	F0
Direct Access	FF

- Return Point Address is a 4-byte field used during concurrent processing to contain the address at which the functional routine is to continue processing.

- Device Constants Address is a 4-byte field that initially contains the address of the control section IEHDCONS. This control section contains information about each type of direct access device, and the field is updated to point to the IEHDCONS entry pertaining to the device type involved in performing the function.

- Function Dependent Area is a field whose format and size depend on the function to be performed. The format used in each case is shown with the description of the way the function is performed.

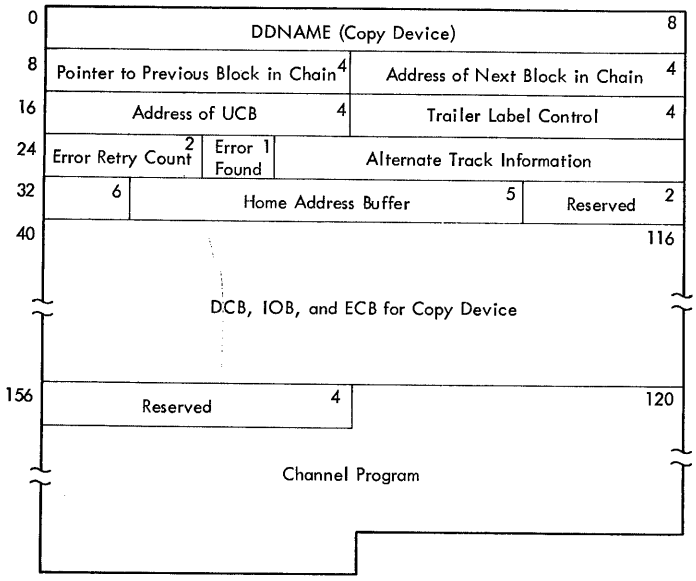


Figure 30. IEHDASDR Copy Block

When it has constructed the function block and any necessary copy blocks, the Control routine enqueues the function block in the function queue by creating a function queue entry for the block. The function queue is a FIFO queue; each entry points to a function block, and the Control routine attempts to initiate performance of functions in the order in which they are enqueued. When performance of a function on a set of volumes has terminated, the Control routine deletes the corresponding entry from the function queue, and pushes all lower priority entries toward the top of the queue.

When it has enqueued the function block corresponding to the first control statement, the Control routine initiates performance of the function. The routine loads the appropriate functional routine, loads registers with pointers to the common work area and the function block, then branches to the functional routine.

Subsequently, when the Control routine initiates performance of a function, it must first determine whether the correct functional routine is loaded. If so, it loads the pointers and branches to the functional routine; if not, the Control routine deletes the old functional routine and loads the new one before branching to it.

Once a functional routine has been entered, it may return to the Control routine under the following circumstances:

- The required main storage is not available.
- An I/O operation has been started but not completed, and concurrent operations can take place.
- Performance of the function has been terminated, either because processing is complete or because an unrecoverable error has been encountered. In the latter case, the functional routine passes a return code greater than zero. The Control routine stores the highest return code and passes it to the user at the end of the run.

The logic and processing performed in the Control routine when a functional routine returns control to it is shown in Figure 31.

Functional Routine Returns to Control Routine																
Main Storage Not Available	Y	Y	Y	Y	Y											
Processing Interrupted						Y	Y	Y								
Processing Complete									Y	Y	Y	Y	Y	Y	Y	Y
Current Entry is at Top of Queue	N	Y	Y	Y	Y				Y	Y	Y	Y	N	N	N	N
Current Entry is Last in Queue		Y	Y	N	N				Y	Y	N	N	Y	Y	N	N
Additional Queue Space is Available							Y									
Next Entry Can be Processed						Y	N	N								
End-of-Data on SYSIN		Y	N	Y	N	Y	Y		Y	N	Y	N	Y	N	Y	N
DO ACTION NUMBER	4	7	9	8	9	1	2	2	7	9	8	9	8	9	8	9

1. Initiate the function specified in the function block corresponding to the next queue entry.

2. Initiate the function specified in the function block corresponding to the entry at the top of the queue.

3. Release the main storage obtained in the Control routine, close the SYSIN and SYSOUT data sets, and return control to the caller.

4. Mark the entry "No Main Storage Available" and do Action 2.

5. Free the main storage occupied by the function block and delete the entry from the function queue.

6. Scan, and enqueue a function block for the next control statement.

7. Do Actions 5 and 3.

8. Do Actions 5 and 2.

9. Do Actions 5, 6, and 2.

Note: The next entry can be processed if the functions are the same, the devices are available, and main storage is available.

Figure 31. IEHDASDR Control Routine Processing at Functional Routine Return

Performing the Dump Function

When the Dump function is specified, the Control routine passes control to the Dump routine (module IEHDDUMP). This routine (see Chart 30) initializes the input device and the output devices, then passes control to the I/O routine (module IEHDEXCP). Module IEHDEXCP performs the I/O operations, except that when the output is a SYSOUT data set, it uses module IEHDACUT as a subroutine to format and write the dumped information.

The dump routine returns control to the Control routine whenever processing is interrupted to await completion of an I/O operation, and when the function is terminated, either because the dump is complete or because an uncorrectable I/O error makes it impossible to continue.

When it is entered, the Dump routine verifies that the input device is a direct access device, then issues a conditional GETMAIN macro instruction to obtain main storage for a buffer and a work area. If enough storage is not available, the routine returns control to the Control routine.

If the Dump routine is able to obtain the required main storage, it constructs an ECB, IOB, and DCB for the input device, and stores them in the function-dependent area of the function block (see Figure 32). It uses the RDJFCB and OPEN (type J) macro instructions to read the JFCB and open the

VTOC data set, then sets the dump extents to correspond to the tracks specified in the function block by converting the track specifications to CCHH format and storing them in the limits record (Figure 33). If no dump extents are specified, the routine stores the CCHH of the first and last tracks on the volumes.

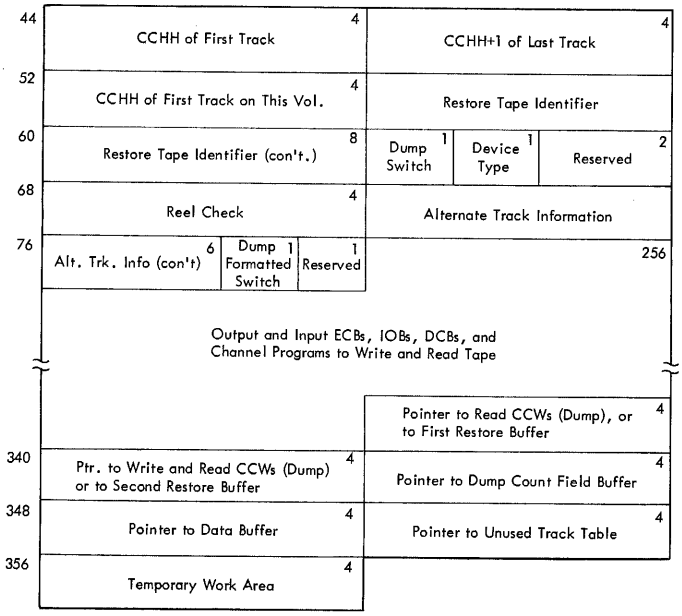
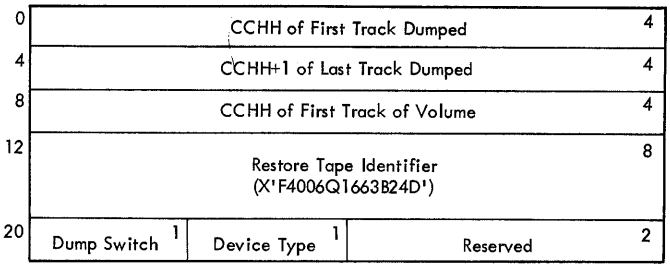


Figure 32. IEHDASDR Function Block -- Dump/Restore Area

Notes: This figure shows the format of the function-dependent area of the function block as it is used in the performance of the DUMP and RESTORE functions. Although most of the field names are self-explanatory, the following fields require further explanation:

- The first seven fields in the area are the 24-byte limits record.
- The reel check field contains the first 4-bytes of the restore tape trailer label; it indicates whether the reel is the last reel required to complete the restore.
- The Alternate Track Information field is extracted from the Format 4 DSCB of the primary output volume and contains two subfields: the first four bytes contain the CCHH of the next alternate track available, and the last two bytes contain the number of alternate tracks available.
- The field containing the pointer to the first RESTORE buffer may also contain X'FF' in the high order byte. If so, it indicates that there are two RESTORE buffers, and the next field points to the second buffer.
- The last 16 bytes of the area are present only for the Dump function.



Notes: 1. Dump Switch settings:
X'F0' = Full Dump
X'00' - Partial Dump

2. Device Type Codes:
0 = 2321
1 = 2311
2 = 2314
3 = 2302
4 = 2303
5 = 2301

Figure 33. 24-Byte Limits Record

If the output is a SYSOUT data set, it is the only output permitted; the Dump routine performs no further initialization, but passes control to the I/O routine. If the output is to tape or direct access, there may be multiple output volumes, and further initialization must be performed for each of the output volumes.

The routine constructs an ECB, IOB, and DCB for the first output volume, and stores them in the function block. If the volume is a tape volume, the routine opens the tape, and uses the EXCP macro instruction to write the limits record. If the volume is a direct access volume, the routine verifies that it is not System Residence, then uses the RDJFCB and OPEN (type J) macro instructions to read the JFCB and open the VTOC data set. The routine then reads the Format 4 DSCB and saves the alternate track information so that it can be placed in the VTOC of the output volume when the dump is complete.

When it has initialized the first output volume, the Dump routine determines whether additional volumes have been specified. If so, it verifies that the next volume is of the same type, then initializes it. The procedure is the same as that used for initializing the first volume, except that the IOB, ECB, and DCB are stored in the copy block associated with the volume. Any other output volumes are then initialized, one at a time.

When all of the output volumes have been initialized, the routine passes control (via a LINK macro instruction) to module IEHDPASS to have the required security checks made. On the return, the Dump routine reads and inspects the Format 5 DSCB from the input device. The routine extracts the available extent information, converts it to CCHH form, and builds a table of unused tracks. The I/O routine uses the table to insure that (unless the output is a SYSOUT data set), only those tracks that are in use (listed in the DSCB as "not available for allocation") will be dumped. When it has built the table, it passes control to the I/O routine.

The function of the I/O routine is to read information from the input volume and (if the output volume is a tape or direct access volume) to write the information out. If the output is a SYSOUT data set, the I/O routine uses module IEHDAOUT as a subroutine to format and write the data.

When the I/O routine has determined that a track is within the specified limits, and that it is either in use or that the output is a SYSOUT data set, it issues the EXCP macro instruction to execute a channel program that reads the data field of record 0, the count, key and data fields of record 1 (if it exists), and the count fields of any additional records on the track. When it has issued the EXCP, the I/O routine returns control to the Dump routine, which in turn returns control to the Control routine. When it is re-entered to continue performing the function, the I/O routine waits for the channel program to be completed.

When the channel program is complete, the I/O routine determines whether the track contains a home address and only one record (R0), a home address and two records (R0 and R1), or a home address and more than two records:

- If the track contains only a home address and record 0, the routine determines the output device type, writes out the contents of the record, and erases the remainder of the track.

- If the track contains a home address, record 0, and record 1, the I/O routine determines the output device type, and writes out the contents of the records.

- If the track contains a home address, record 0, record 1, and additional records, the I/O routine reads the key and data fields of record 2 and the count, key, and data fields of the additional records. It then determines the output device type, and writes out the contents of the records.

If the output is a SYSOUT data set, the I/O routine passes control to module IEHDAOUT, which formats and writes the track contents.

If the output volume is a direct access volume, the I/O routine writes to every (primary) track on the volume. Those tracks on the input volume that are in use are copied onto the output volume; each track corresponding to an unused input volume track is formatted with a home address and record 0. The remainder of the track is cleared.

If the output volume is a tape volume, the I/O routine writes a control record for each track on the input volume. The control record contains the channel program used by the Restore routine to write one track; it is followed by the track image record, which contains the data field of record 0, and all fields of any other records on the track.

At end-of-volume, a trailer record is written following the tapemark. The first 4 bytes of this 24-byte record indicate whether this volume is the last volume of the restore data set.

The I/O routine (module IEHDEXCP) returns control to the Dump routine under two conditions:

If the Dump routine is performing functions concurrently, module IEHDEXCP returns control to it whenever processing is interrupted to wait for the completion of an I/O operation on a track that contains a home address and more than two records. In this case, the Dump routine returns control to the Control routine; when it is re-entered to perform the same function, the Dump routine again passes control to the I/O routine.

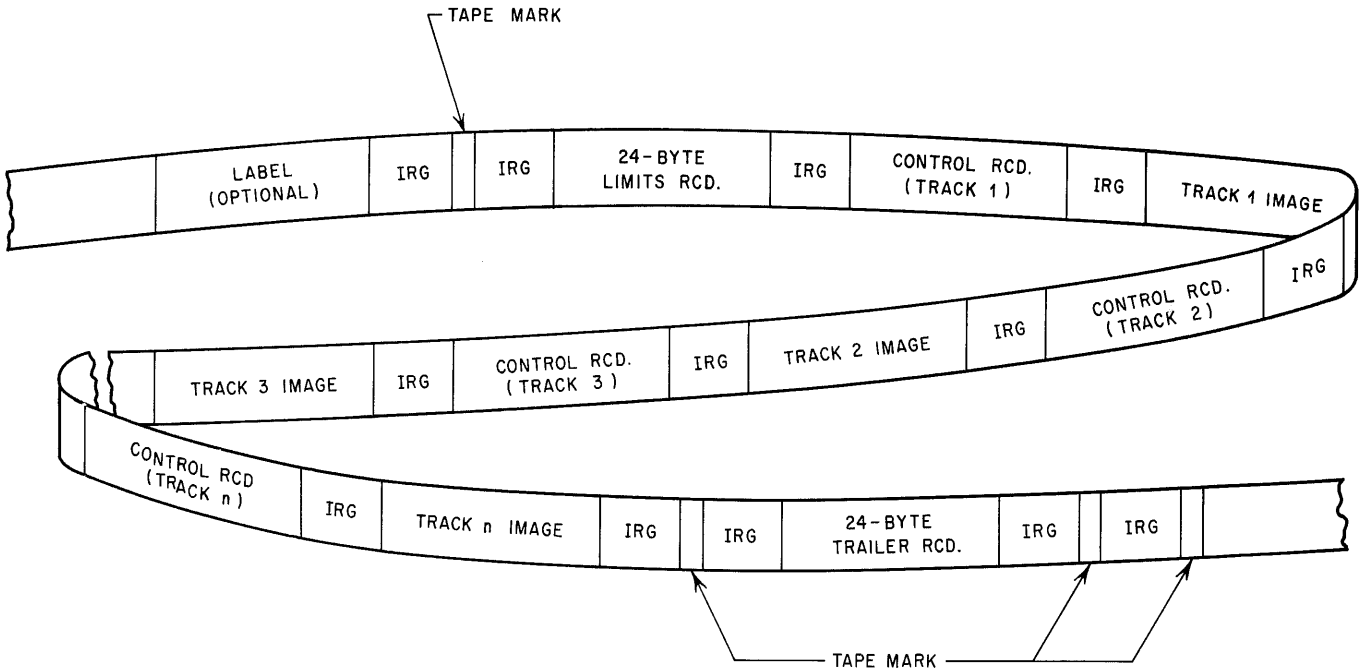
If the I/O routine has terminated its processing, either because the dump is complete or because of an unrecoverable I/O error, it returns control to the Dump routine. In this case the Dump routine closes the input and output data sets, releases the main storage it obtained for buffers, places a completion message in the message data set, and returns control to the Control routine.

Performing the Restore Function

When the restore function is specified, the Control routine passes control to the Restore routine (module IEHDREST), which is shown in Chart 32. The input to the Restore routine is a restore tape, which may have been created by performing the Dump function in this program, or in the IBCDMPRS program. A restore tape (see Figure 34) contains the information necessary to make a copy of the direct access volume used to create it; the Restore routine makes one or more such copies. The Restore routine returns control to the Control routine when the restore is complete, when an uncorrectable error makes it impossible to continue processing, or when processing is interrupted while awaiting completion of an output operation.

When it is first entered, the Restore routine attempts to obtain main storage for two buffers. If it is able to obtain enough storage for at least one buffer, processing continues; if not, the routine sets a switch and returns control to the Control routine.

If storage is available for at least one buffer, the routine determines the validity of the output volume specifications. The output volumes must all be of the same type, but the system residence volume(s) may not be specified.



Limits Record:	A 24-byte record containing extent limits and restore tape identifier, located after the initial tape mark on the first volume of the restore tape.
Control Record:	A variable-length record containing the channel program required to write the associated track, located immediately before the track image record for the track.
Track Image Record:	A variable-length record containing the count, key, and data fields of the records on the track.
Track Record:	A 24-byte record containing, in the first 4 bytes, the reel number and termination code.

Figure 34. Restore Tape Format

If the output volume specifications are valid and the volumes are available, the routine opens the input tape, checks the limits record to insure that the tape is a restore tape and that the volume used to create it is the same type as that specified for output.

If so, the routine builds an ECB, IOB, and DCB for each output volume. If there are multiple output volumes, the control blocks for the first are stored in the function block, and those for the additional devices are stored in the copy blocks.

When it has constructed the control blocks, the routine uses the RDJFCB and OPEN (type J) macro instructions to read the JFCB and open the VTOC data set on each output volume, and uses the Password Protection routine (IEHDPASS) to make the required security checks on the volume's data sets.

The Restore routine uses the EXCP macro instruction to read the Format 4 DSCB from each output volume, then extracts and saves the alternate track information. Since the VTOC will be replaced with the VTOC from the volume used to create the restore tape, the alternate track information from the output volume must be placed in the new VTOC.

When initialization is complete, the Restore routine uses the EXCP macro instruction to read a control record and a track image record from the restore tape. The control record contains the channel program necessary to write the track image record to the output volumes. The Restore routine updates the channel program with the correct data addresses, then issues the EXCP macro instruction for each output volume.

When it has issued the EXCP, the routine returns control to the Control routine. When it is re-entered to continue performing the function, the routine waits for the output operations to be completed. When the operations are completed, the routine again reads from the restore tape and repeats the procedure.

At end-of-volume, the routine reads the trailer record from the restore tape and determines whether there are additional tape volumes to process. If the first four bytes of the trailer record contain X'FFFFFFFE', the restore is complete. The routine updates the Format 4 DSCBs in the output volumes, places a completion message in the message data set, releases the main storage it obtained, closes the input and output DCBs, and returns control to the Control routine. If the trailer record does not indicate that the restore is complete, the return issues the EOV macro instruction to have the next volume mounted, and continues processing.

Performing the Analyze and Format Functions

When the Analyze or Format function is specified, the Control routine passes control to the Analyze/Format routine (module IEHDANAL). This routine (shown in Chart 33) performs surface analysis and formatting functions for disk and drum volumes (or passes control to module IEHDCELL to perform these functions if the device is a data cell drive) and passes control to module IEHDVTOC to construct and write IPL, volume label, and VTOC records. When processing is terminated, either because the function has been completed or because a computing system error has made it impossible to continue, the routine returns control to the Control routine. The routine also returns control to the Control routine during concurrent operations when processing is interrupted for an I/O wait.

Initialization

When the Analyze/Format routine is first entered, it is given the address of the function block specifying the function to be performed.

Note: The format of the function-dependent area of the function block, as is used in the performance of the Analyze and Format routines, is shown in Figure 35.

If the function is to be performed on more than one device, copy blocks have been chained to the function block; the routine constructs an IOB, ECB, and DCB for each volume. It stores the blocks for the first volume in the function block, and stores the blocks for the additional volumes in the copy blocks. If a volume is new (unlabeled) the routine makes sure that the device containing that volume is offline, then uses the SVC routine to construct a DEB in protected storage, but performs no open. Otherwise, the routine uses the RDJFCB and OPEN (type J) macro instructions to read the JFCBs and open the VTOC data sets.

When it has performed the open or constructed the DEB, the routine uses the Password Protection routine to make security checks on the volume. On the return, it initializes a channel program to analyze and format or to format each device, then stores the channel program in the appropriate function or copy block. If the devices are 2321 Data Cell Drives, the construction and storing of the channel program, as well as the execution of the surface analysis

and formatting procedures is performed in module IEHDCELL; if the devices are disks or drums, these functions are performed in module IEHDANAL.

Surface Analysis and Formatting Procedures -- Disk and Drum Volumes

The nature of the channel program used depends on whether a surface analysis or formatting operation is being performed, whether a flag test has been specified, whether multiple passes are to be made on each track, and whether the volume is a disk or drum. The sequence of commands in each case is shown in Figure 36. Note that the two Analyze/Format channel programs are virtually identical, except for the first two commands, as are the two Format Only channel programs. The first two commands are different because an unused disk has no home addresses, and no successful search could be made. Also, since defective tracks on a drum are not flagged, rewriting the home address will not destroy any previously written flags.

The first part of the Analyze/Format channel program is executed on each pass; maximum length R0s are written twice and read back twice, and the home addresses are read twice. If a flag test is to be done, the data is transferred on the second home address read, and the field is checked for the presence of a defective track flag.

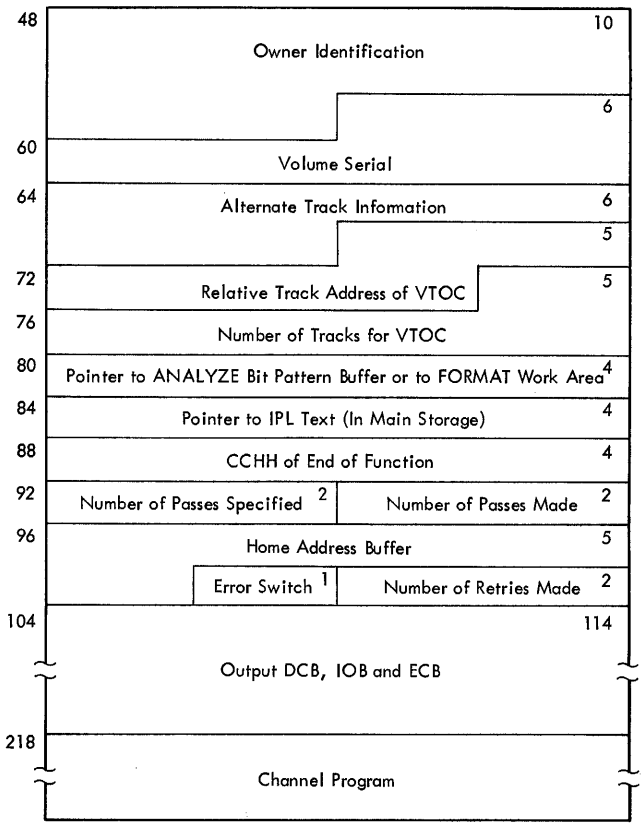


Figure 35. IEHDASDR Function Block -- Analyze/Format Area

	Analyze/Format		Format only	
	Drum or Disk (no flag test)	Disk (flag test)	Drum	Disk
All Passes	Write Ha TIC *+8 Write R0[1] Read Ha Read R0[1] Search Ha TIC *-8[1] Write R0 Read Ha Read R0	Search Ha TIC *-8 Write R0[1] Read Ha Read R0 Search Ha TIC *-8 Write R0 Read Ha[2] Read R0		
Last Pass Only	Search Ha TIC *-8 Write R0[3] Read Ha Read R0	Search Ha TIC *-8 Write R0[3] Read Ha[2] Read R0	Write Ha TIC *+8 Write R0[3] Read Ha Read R0	Search Ha TIC *-8 Write R0[3] Read Ha[2] Read R0

[1]Write maximum length possible (full track).
[2]Transfer Ha into main storage and test for flags (no data transferred on other reads).
[3]Write standard (8-byte) R0.

Figure 36. Analyze/Format Channel Programs

The second part of the Analyze/Format channel program (which duplicates the Format Only channel program for the corresponding device type) is executed only on the last pass. A standard (8-byte) R0 is written, and in the case of a disk device, the home address is tested for flags.

During concurrent operations, the routine returns control to the Control routine when the EXCP macro instruction for each device has been issued; it is eventually reentered to wait for completion of a channel program.

When a channel program is completed, the Analyze/Format routine determines whether any errors have occurred. If not, and if there are other channel programs that have not been completed, the routine enters the wait again. It repeats this procedure until either an error occurs, or until all channel programs have been completed.

When all channel programs have been completed, the routine determines whether additional passes have been specified. If so, it re-issues the EXCP macro instruction for each device and repeats the procedure until all required passes have been made.

When the last pass has been made, the routine reinitializes the channel programs for each device so that they apply to the next track, and repeats the entire analysis/format procedure.

Error Procedures -- Disk and Drum Volumes

There are two classes of errors that can occur during a surface analysis operation: errors that indicate a failure of the computing system, and errors that indicate a defect in the volume being analyzed. Those errors that indicate machine malfunctions are handled by the normal I/O Supervisor error routines. If such an error cannot be corrected, the Analyze/Format routine terminates the function, closes the volumes, and returns control to the Control routine. If the function being performed is the format function, all errors arehandled in this manner.

When a surface analysis is being performed, however, a distinction is made between the two types of errors. The errors that indicate that a track is defective, and are handled by the Analyze/Format routine, are Data Check and (for the 2314 Direct Access Storage Facility only) Track Overflow.[1] When such an error is encountered, the Analyze/Format routine retries the channel program until an error is encountered again or until the channel program has been retried ten times with no errors. If an error occurs the track is declared defective, and the routine places a message describing the defective track in the message data set. If the device is a drum, no alternate track can be assigned by the program, and the IBM Field Engineer should be notified. If the device is a disk, the Analysis/Format routine issues SVC 82 and the Alternate Track Assignment routine is used to assign an alternate track. If the track is in the alternate track area, however, no alternate will be assigned; the track will be flagged defective to prevent its future assignment.

When an alternate track is assigned, the Analyze/Format routine places a message describing the alternate track in the message data set.

[1] On the last "READ RO", the routine also handles a No Record Found/Missing Address Markers condition.

Surface Analysis and Formatting Procedures -- Data Cell Volumes

The surface analysis and formatting of a data cell volume is performed by module IEHDCELL, which is used as a subroutine by the Analyze/Format routine. Module IEHDCELL writes a home address, a standard length (8-byte) R0, and a maximum length R1 on each track of a cylinder, then reads each home address, R0, and R1 back to check for errors. The channel programs used for writing and reading are as follows:

Writing	Reading
Write HA	Read HA
Write R0	Read R0
Write Count-Key-Data	Read Count-Key-Data

The routine repeats the procedure, cylinder by cylinder, until each track on the volume has been read and verified. When the analysis of a strip, subcell or cell is complete, the routine makes additional (address compare) checks to verify correct positioning.

Error Recovery Procedures -- Data Cell Volumes

Most of the errors that may be encountered while performing the surface analysis of a data cell volume are handled by normal I/O Supervisor error procedures, and if they cannot be corrected, the function is terminated. There are two exceptions to this procedure:

- No Record Found and Missing Address Markers: The I/O Supervisor error recovery routine is used, but if the errors occur together, and no recovery

is possible, module IEHDCELL places a message describing the defective track in the message data set, and causes an alternate track to be assigned.

- Data Check: If this error occurs, module IEHDCELL retries the channel program up to 113 times. If the channel program is executed successfully once, the track is considered good. If no successful execution occurs the track is considered defective. In that case a message describing the defective track is placed in the message data set, and an alternate track is assigned.

The alternate track assignment procedure is the same as that used for disk volumes. The alternate track area of the volume is checked first, and defective tracks found in that area are flagged. No alternate tracks are assigned to defective tracks in the alternate track area. If the defective track is not in the alternate track area, module IEHDCELL places a message describing the defective track in the message data set, issues SVC 82 to have an alternate track assigned, then places a message describing the alternate in the message data set.

Supplying a VTOC and IPL Records

When the last track on each device has been analyzed or formatted, the Analyze/Format routine passes control to module IEHDVTOC (see Chart 34). This module constructs and writes the IPL Bootstrap, IPL Text, VTOC, and Volume Label records.

When it is entered, module IEHDVTOC determines whether it is to write an IPL record on the output volumes. If so, and if the IPL text is on external storage, the routine opens the appropriate data set and reads the text into main storage.[1]

If it is to write an IPL program, the routine constructs two IPL Bootstrap records and writes them to records 1 and 2 on track 0 of each output volume. The IPL program itself is written on track 1 before the bootstrap records are written (if the devices are 2303s or 2311s) or on record 4 of track 0, with the same channel program used to write the bootstrap records (if the devices are 2301s or 2314s).

---

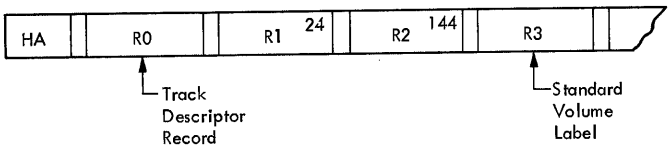
[1]The IPL program may be supplied in the input stream (in which case it is in main storage when IEHDVTOC is entered), it may be in a sequential data set, or it may be a member of a partitioned data set, e.g., member IEAIPL00 of SYS1.SAMPLIB.

If no IPL program is to be written, module IEHDVTOC writes a program on record 1 of track 0 that will cause the computing system to enter the wait state if an attempt is made to execute the IPL procedure using that volume. The format of the records is shown in Figure 37.

Whether it writes an IPL program or not, module IEHDVTOC constructs and writes a standard volume label and a VTOC. The standard volume label is written on record 3 of track 0; it contains the volume serial provided by the user or (if none was provided) the original volume serial. If an owner name has been supplied, the routine places it in the label; if not, the field remains blank. The VTOC constructed and written by IEHDVTOC consists of a Format 4 DSCB, a Format 5 DSCB, and enough dummy (Format 0) DSCBs to fill out the VTOC.

Performing the Label Function

When the label function is specified, the Control routine passes control to the Label routine (module IEHDLABL). The Label routine (Chart 36) replaces the volume serial and, optionally, the owner identification fields of the volume label.



Note: R1 for a non-IPL volume is a 24-byte record having the following format:

PSW	X'0006000000000000F'	
CCW1	X'03000000000000001'	(NOP)
CCW2	X'00000000000000000'	(Dummy CCW)

R1 for a volume that can be loaded contains a 24-byte IPL bootstrap record having the following format:

PSW	X'0000000000000000'	
CCW1	X'06003A9860000060'	(Read Record 2)
CCW2	X'08003A9800000000'	(Transfer to Record 2)

R2 is always a 144-byte record having the following format:

CCW1	X'07003AB840000006'	Seek IPL Track
CCW2	X'31003ABE40000005'	Search for IPL Record
CCW3	X'08003AA000000000'	Repeat until found
CCW4	X'0600000020000E29'	Read IPL Program
Seek	X'00000000000'	Seek Address
Search	X'0000000101'	Search Address

101 bytes of zeros for padding

Figure 37. Format of Track 0, Records 0 and 1

When it is entered, the routine identifies the device to be labeled and verifies that the device is a direct access device.

It uses the RDJFCB macro instruction to read the JFCB into a buffer in the function-dependent area of the function block (Figure 38). It then uses the OPEN (type J) macro instruction to open the VTOC data set.

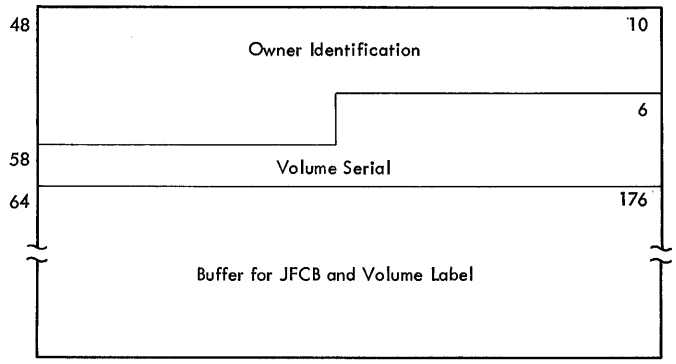


Figure 38. IEHDASDR Function Block -- Label Area

The routine reads the volume label (the data portion of record 3 of track 0) by issuing an EXCP macro instruction. The initial request causes the special End-of-Extent appendage to be entered; the appendage changes the extent limits in the DEB to permit access to track 0, and the volume label is brought into main storage.

When the I/O operation is complete, the Label routine stores the new volume serial and, if one is provided, it stores the owner name. It uses the EXCP macro instruction to write the label, uses the SVC 82 routine to place the new volume serial in the UCB, places a message in the output data set, and returns control to the Control routine.

Performing the GETALT Function

When the GETALT function is specified, the Control routine passes control to the GETALT routine (module IEHDGETA). This routine (which is shown in Chart 37) uses the Alternate Track Assignment routine (SVC 82) to assign an alternate track for the specified disk or data cell track. If the specified track is an assigned alternate track, another alternate track will be assigned in its place. No records will be transferred from the specified track to its alternate. If the specified track is an unused track in the alternate track area, however, no alternate will be assigned; the specified track will be flagged to prevent its future use.

When module IEHDGETA is entered, it verifies that the specified volume is a disk or data cell volume, then uses the RDJFCB and OPEN (type J) macro instructions to read the JFCB into the function-dependent area of the function block (Figure 39) and open the VTOC data set. If the open is successful, the routine checks to see that the specified track is not track 0 or does not contain system data such as a volume label or VTOC.

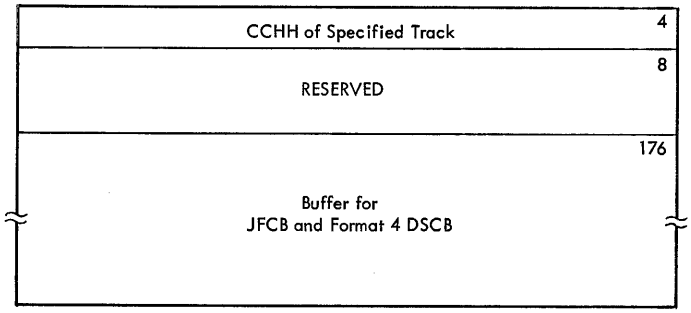


Figure 39. IEHDASDR Function Block -- GETALT Area

If the volume is not a disk or data cell volume, if the open was not successful, or if the specified track is either track 0 or the first track of the VTOC , the routine places an error message in the message data set, terminates the performance of the function, and returns control to the Control routine. If the specified track is a part of the VTOC (other than the first track), the routine issues a warning message and assigns an alternate track.

If the specified track does not contain a volume label or VTOC, the Control routine places a message describing the specified track in the message data set, and issues SVC 82 to execute the Alternate Track Assignment routine.

On the return from the Alternate Track Assignment routine (SVC 82) the GETALT routine places a message describing the alternate track assignment in the message data set, closes the VTOC data set, and returns control to the Control routine.

IEHDASDR Service Routines

There are several service routines used by the IEHDASDR program in the performance of its functions:

- IEHDMSGB, the Message Builder routine, is entered with a pointer to the common work area and a number corresponding to the message. The routine selects the message from the message CSECT. (IEHDMSGS), then moves the message to the output buffer in the common work

area. Certain messages contain "empty" areas that must be filled in by the caller of the Message Builder routine; when this is the case, the Message Builder routine loads a pointer to the empty area, and passes the pointer to its caller.

- IEHDPRINT, the Message Writer routine is entered with a pointer to the common work area (which contains the output buffer and the address of the SYSOUT DCB). The routine uses QSAM to write a header record at the beginning of each page, a copy of each control statement, completion messages, error messages, and (optionally) the contents of a direct access device.

- IEHDDATE, the Date routine, is entered via a CALL macro instruction from the Message Writer routine. The Date routine uses the TIME macro instruction to determine the date, and stores the date (in the form MM/DD/YY) in an 8-byte area furnished by the Message Writer routine.

- IEHDSCAN, the Scan routine, is entered via a LINK macro instruction issued in the Control routine. It reads a control statement (if necessary) and performs a syntax check on one field, then stores the result of the scan in a 1-byte field (SWITCHRD) in the common work area. On the return, it passes the control routine the length of the field and a pointer to the beginning address of the field.

- IEHDPASS, the Password Protection routine (Chart 38), is entered with: 1) an indication of the operation being performed, 2) an indication of whether the purge option was specified in the control statement, 3) a pointer to the function block, and 4) a pointer to a buffer area for reading DSCBs. It uses the Open or Scratch routine to check the password required for access to each security protected data set against the password supplied by the operator. If an incorrect password is issued, or if no password is issued, the routine returns a condition code to its caller, which then terminates the function. In addition, the Password Protection routine determines whether there are any data sets on the output volumes whose expiration dates have not passed. If so, and if the PURGE parameter is specified, it gives the operator the opportunity to terminate the function or to override the expiration dates of all unexpired data sets and continue processing. If the PURGE parameter is not specified, and if an unexpired data set is encountered, the function in terminated.

- IGG019P8, the End-of-Extent Appendage routine is entered from the Input/Output Supervisor. The routine modifies the extent limits and file masks in the DEBs for each direct access volume to be processed, to permit access to the entire volume.

- IGG019P9, the Abnormal-End Appendage routine, is entered from the Input/Output Supervisor. It is used during performance of the Analysis function to bypass normal IOS Error routine processing of Data Checks for all direct access devices.

- IGC0008B, the Alternate Track routine (SVC 82), is entered with a pointer to the parameter list shown in Figure 40. The routine has three basic functions:

  (1) It builds a DEB for handling new direct access volumes,

  (2) It assigns an alternate track for a specified (defective) track, and

  (3) It updates UCBs to reflect new volume serials or VTOC location changes.

Build DEB for New Volume

8F	UCB Address
80	DCB Address

Assign Alternate Track

Function	UCB Address
CCHH of Defective Track	
80	Ptr. to Alternate Track Information (GETALT)

Update UCB

08	UCB Address
New Volume Serial (or Zero)	
80	MBCCHHR of VTOC (if new volume)

Figure 40. SVC 82 Parameter Lists

\Chart 27. IEHDASDR Overall Flow

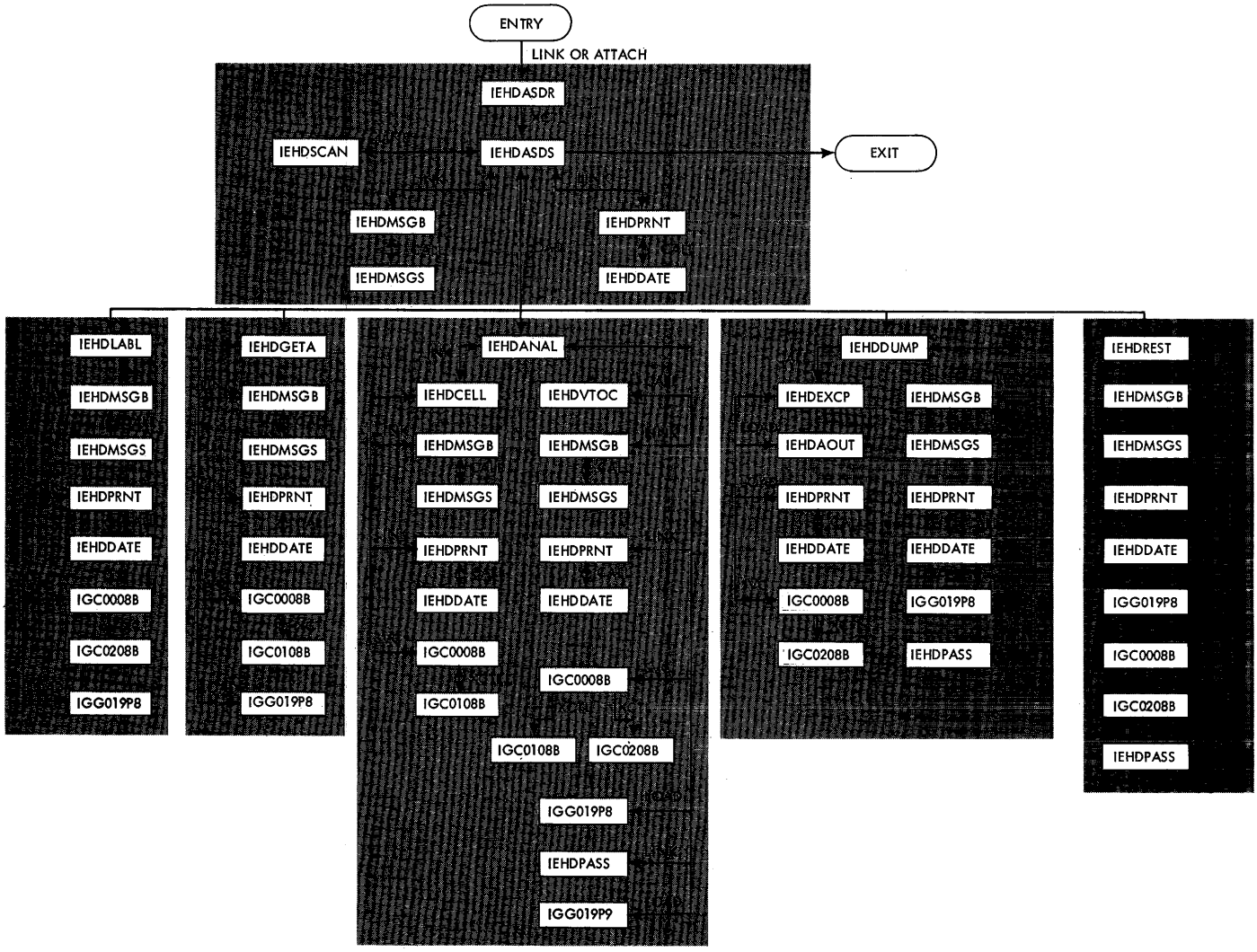


Chart 28. IEHDASDR Control Routine (Part 1 of 2)

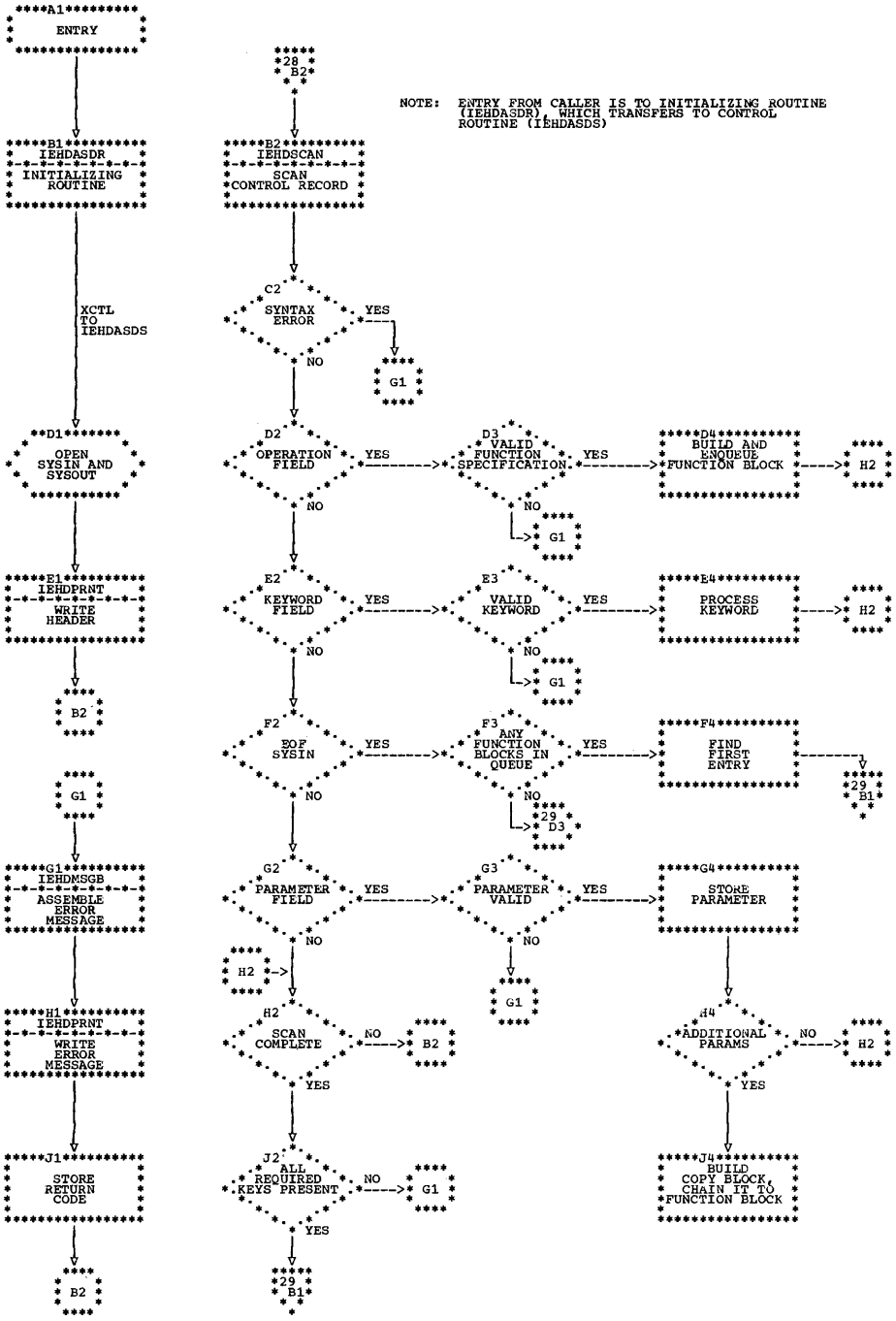
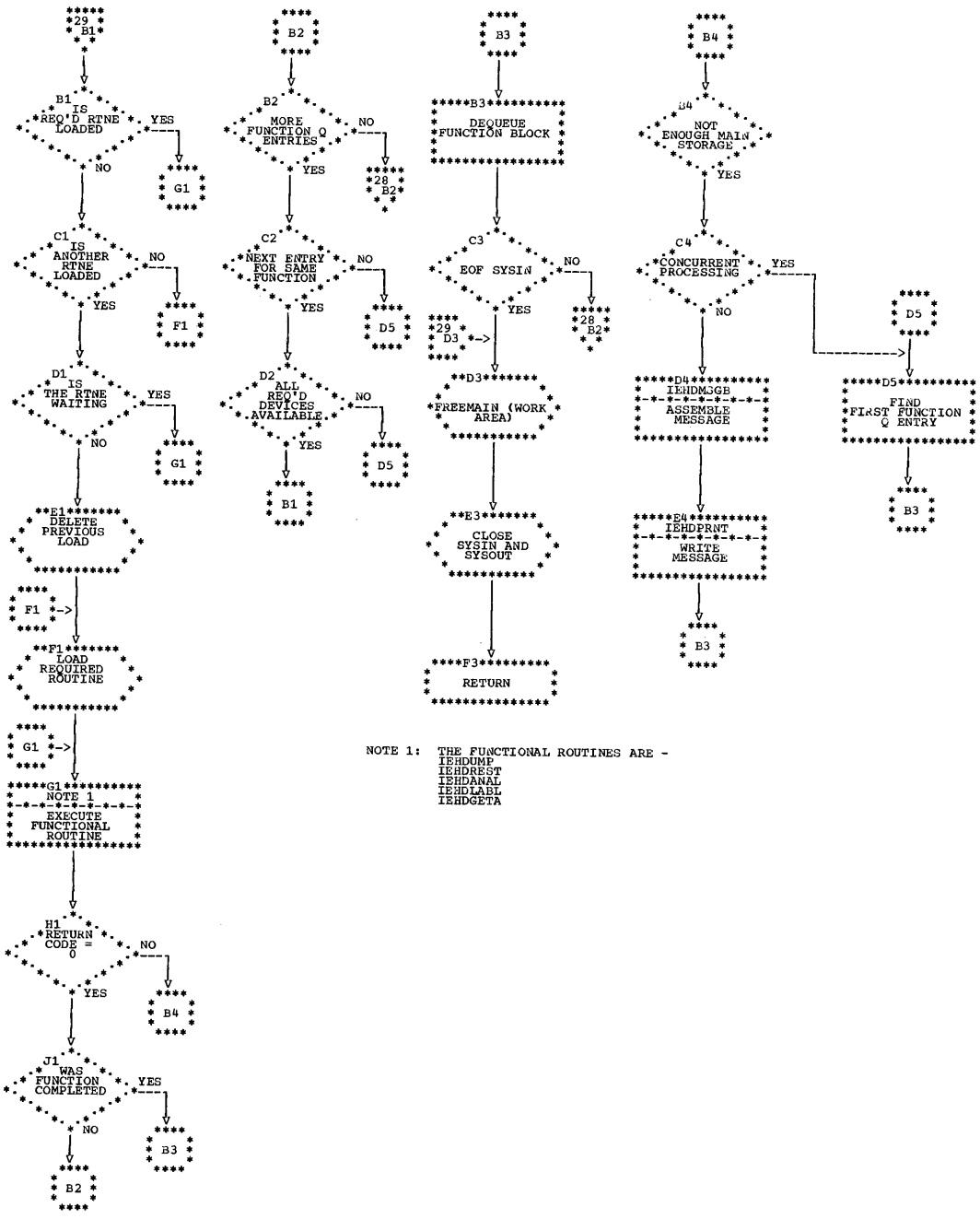


Chart 29. IEHDASDR Control Routine (Part 2 of 2)



NOTE 1: THE FUNCTIONAL ROUTINES ARE -
IEHDUMP
IEHDREST
IEHDANAL
IEHDLABL
IEHDGETA

Chart 30. IEHDASDR Dump Routine

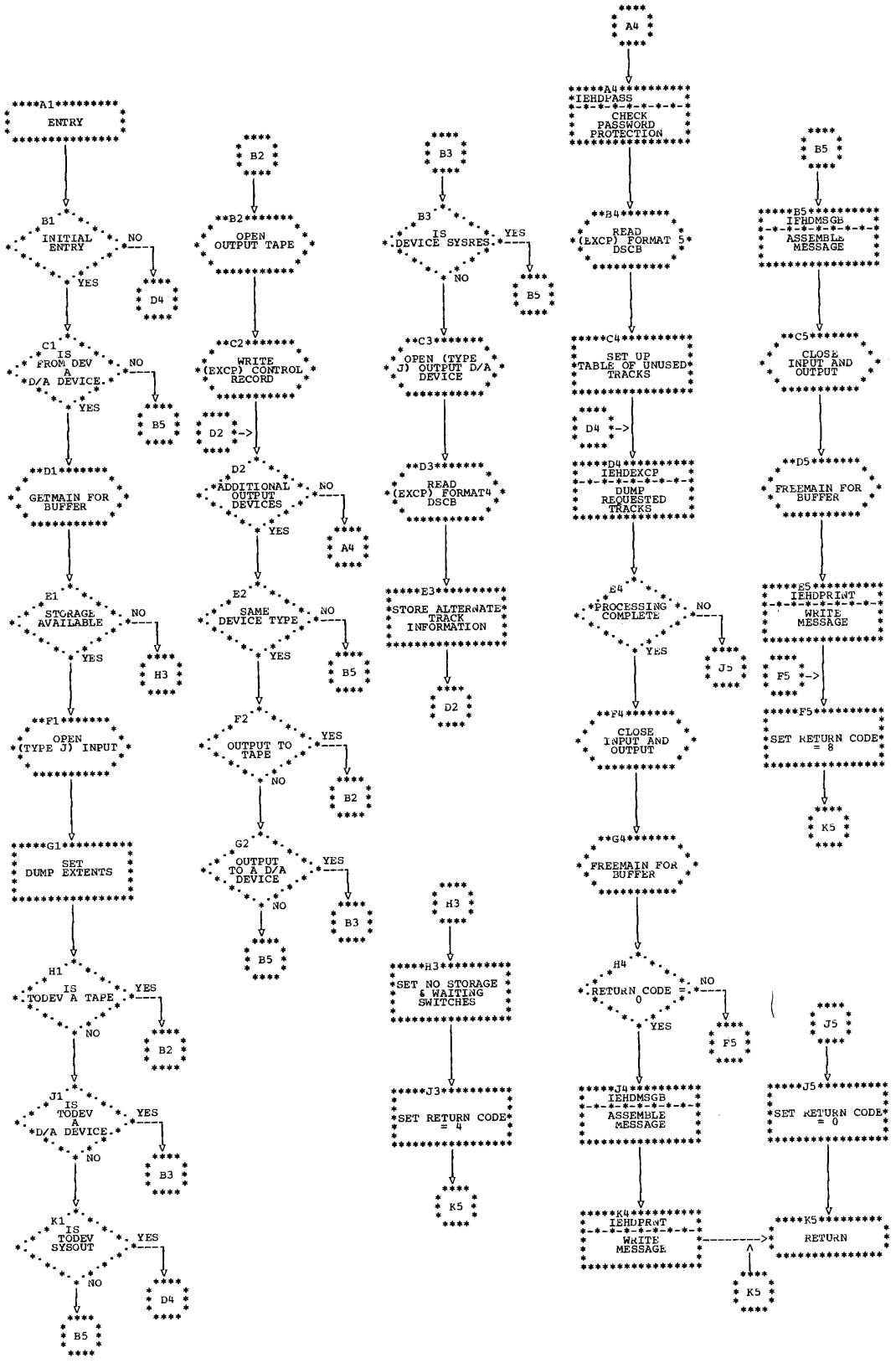


Chart 31. IEHDASDR EXCP Routine

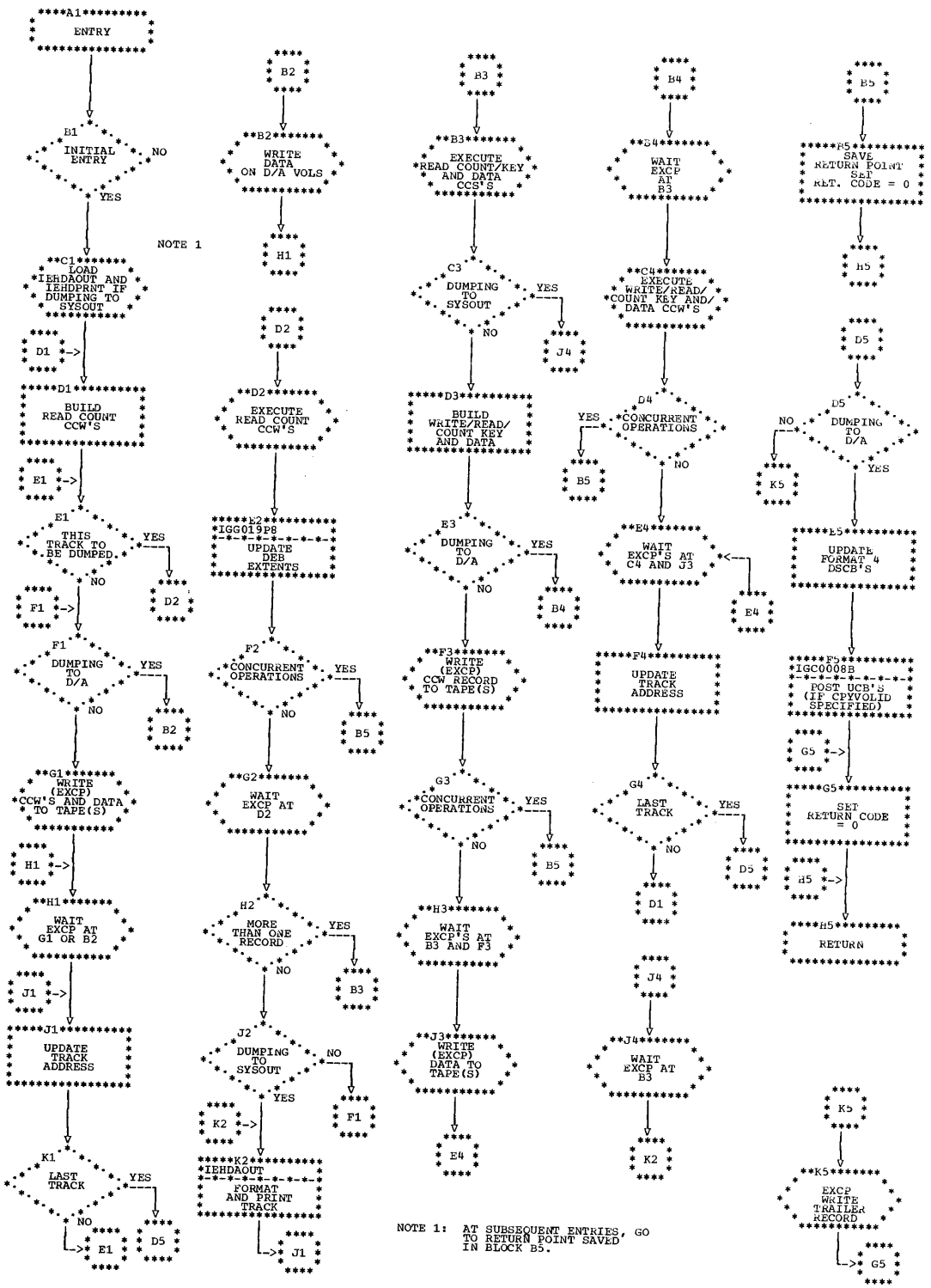


Chart 32. IEHDASDR Restore Routine

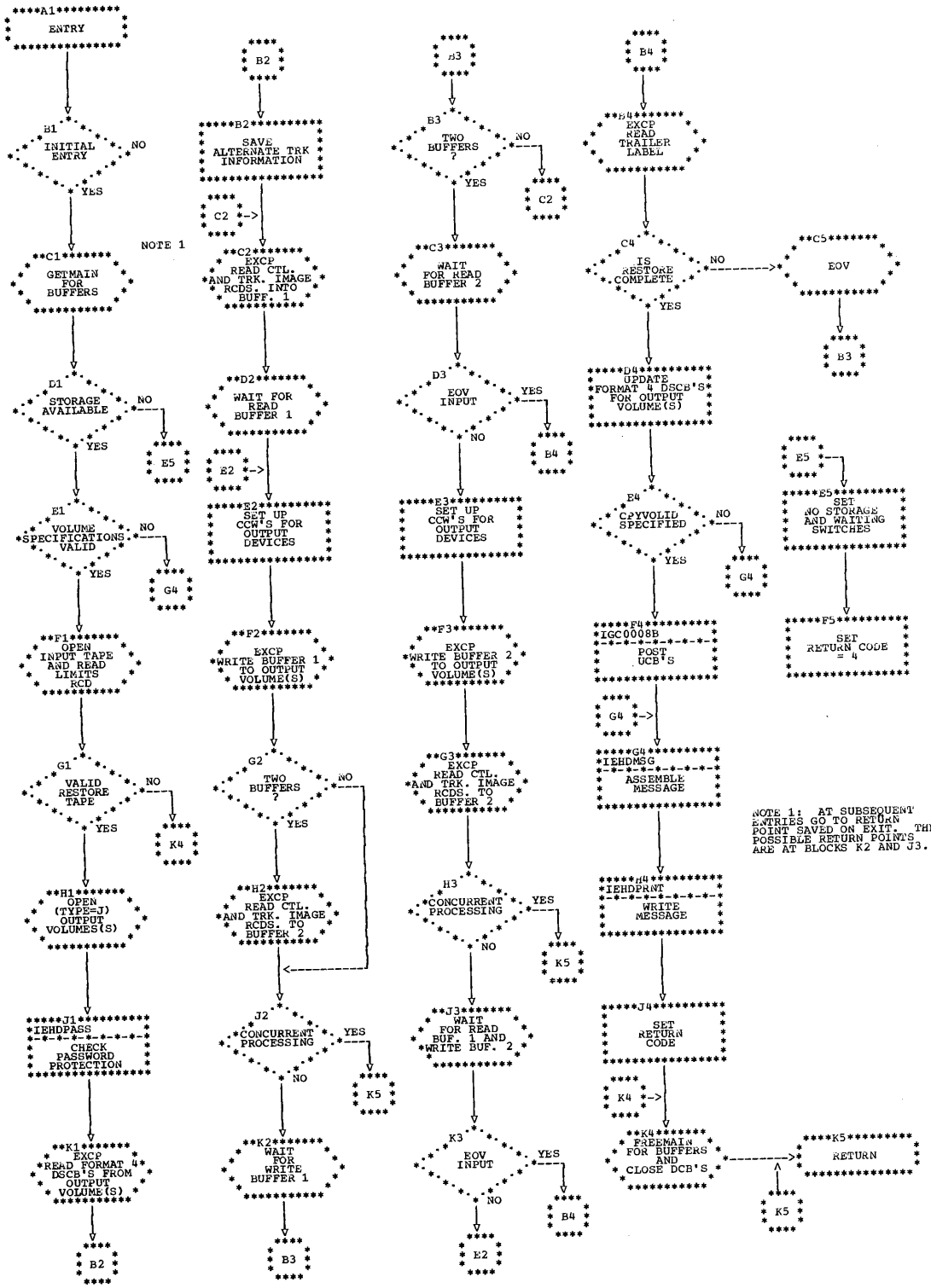


Chart 33. IEHDASDR Analysis Routine

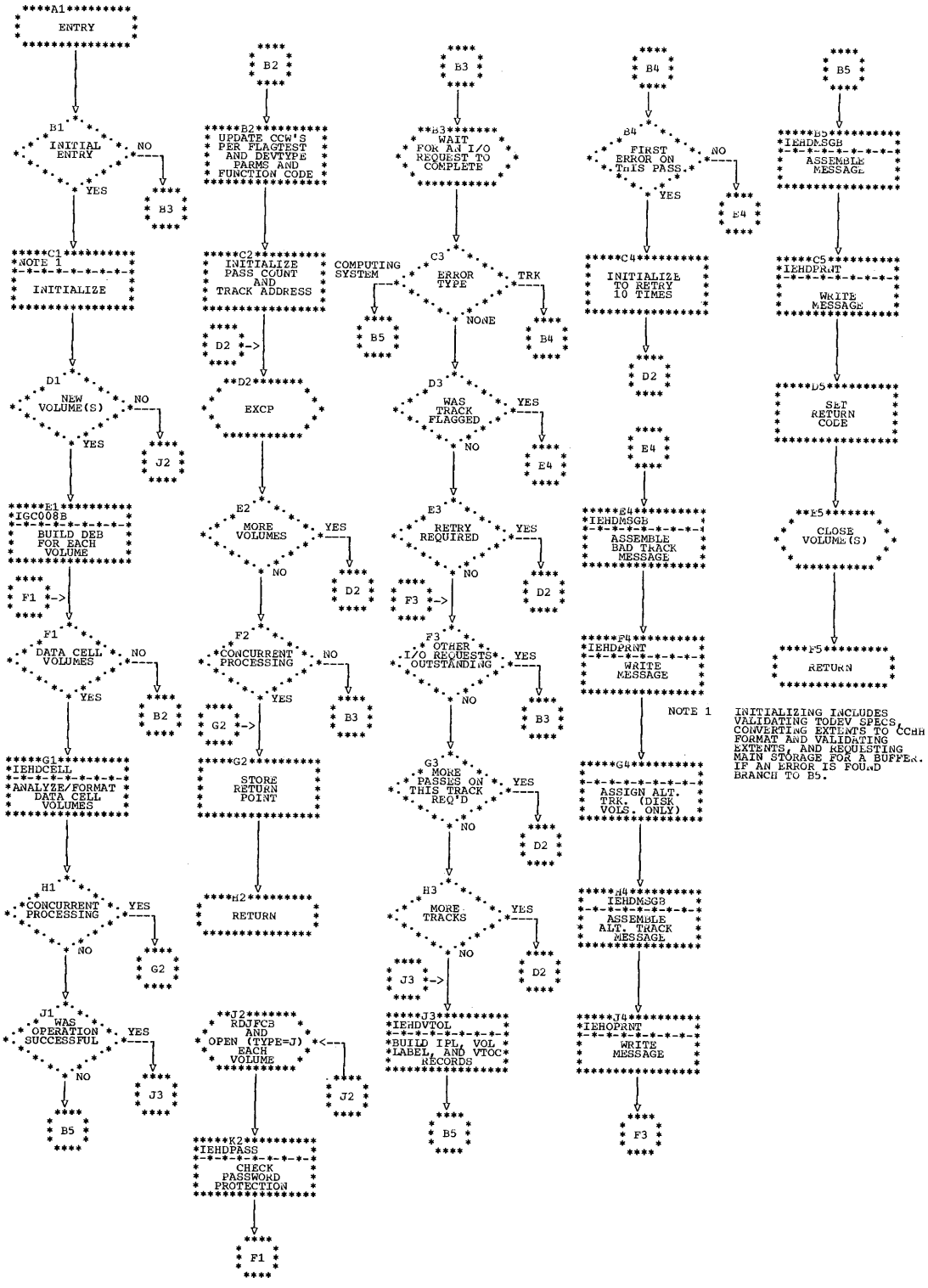


Chart 34. IEHDASDR VTOC Routine

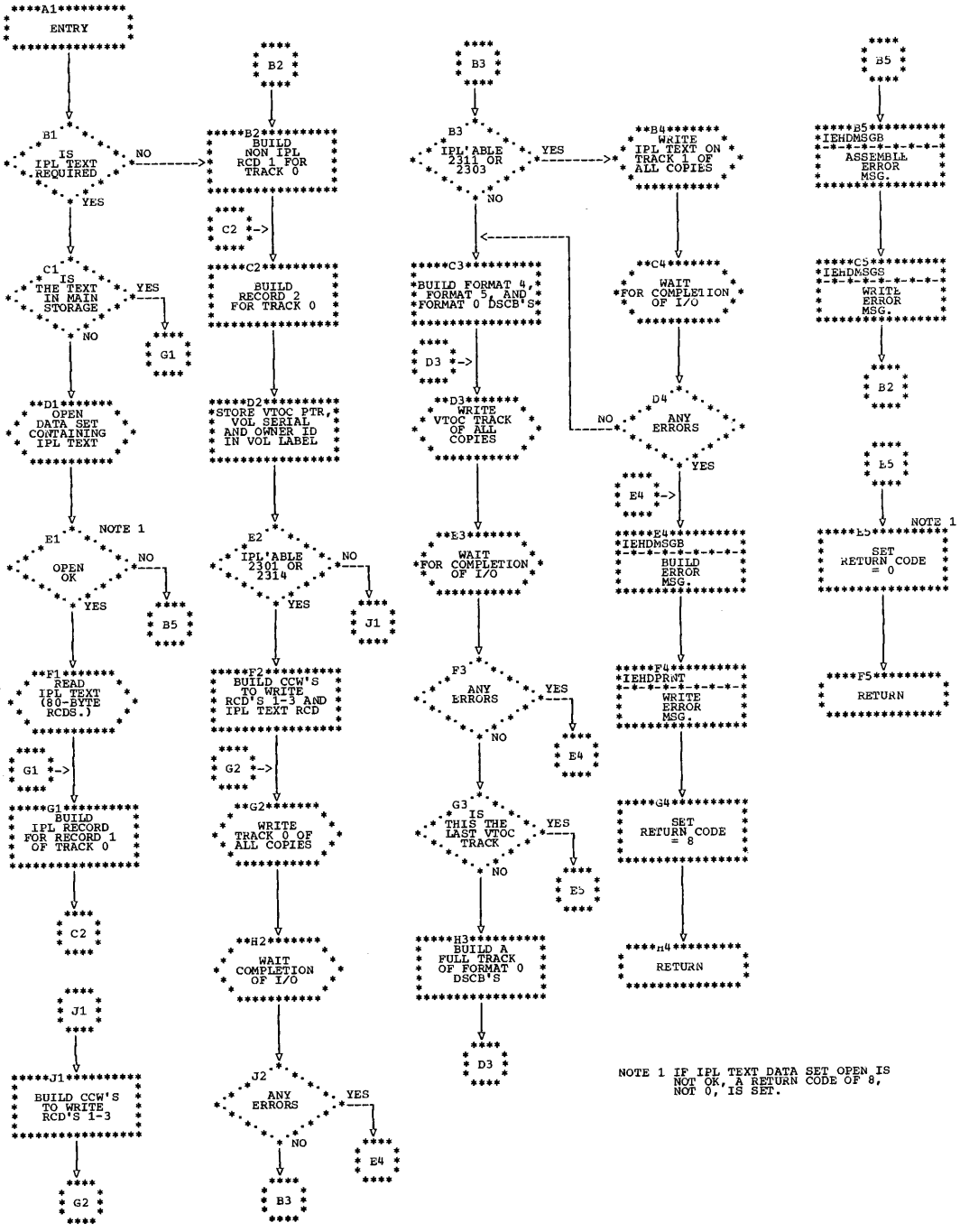


Chart 35. IEHDASDR Data Cell Analysis Routine

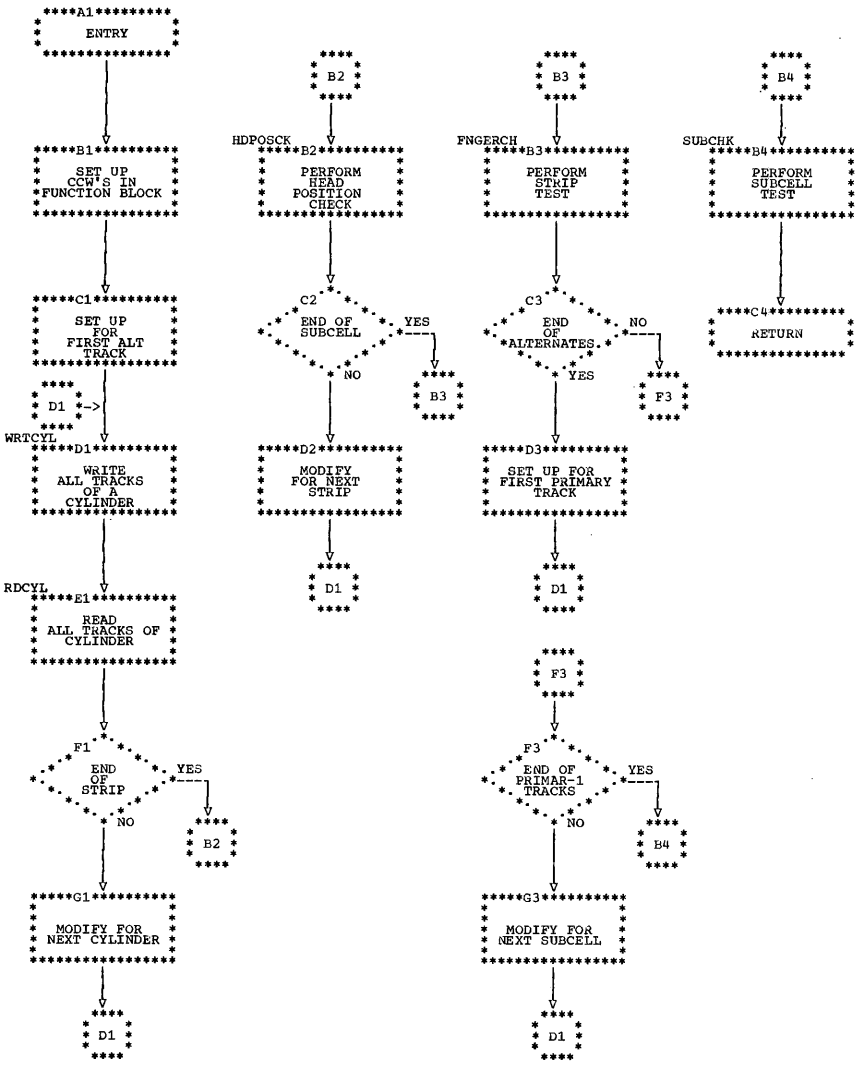


Chart 36. IEHDASDR Label Routine

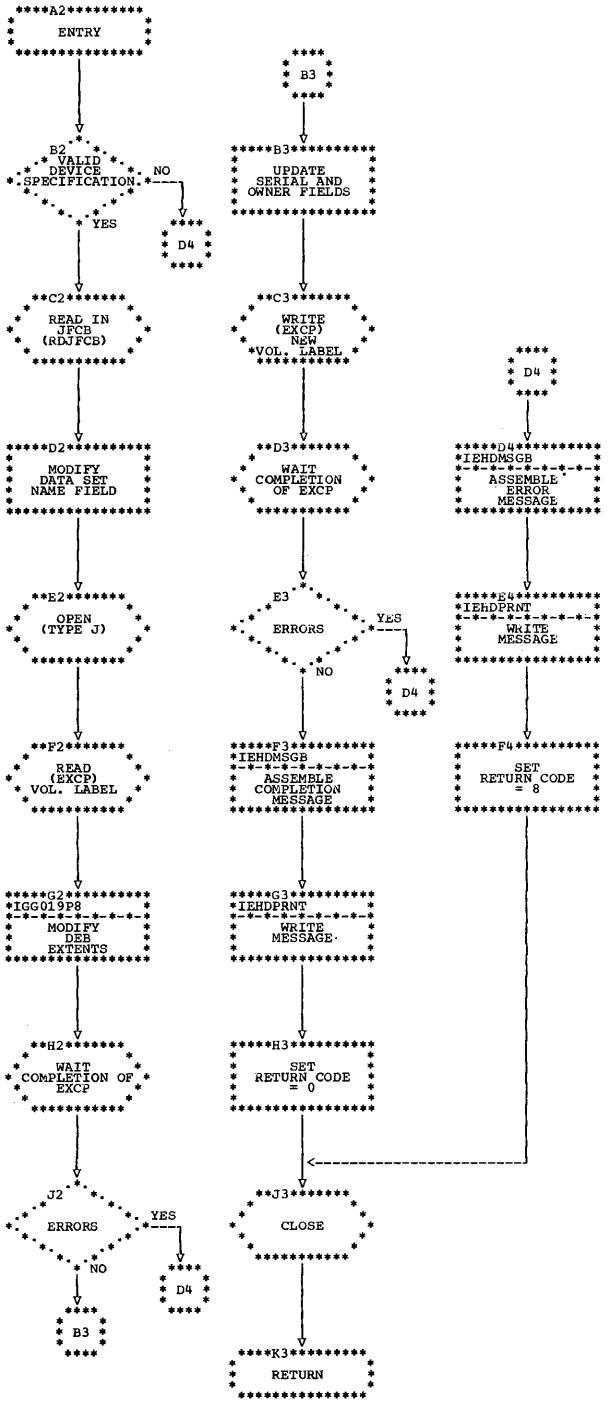


Chart 37. IEHDASDR GETALT Routine

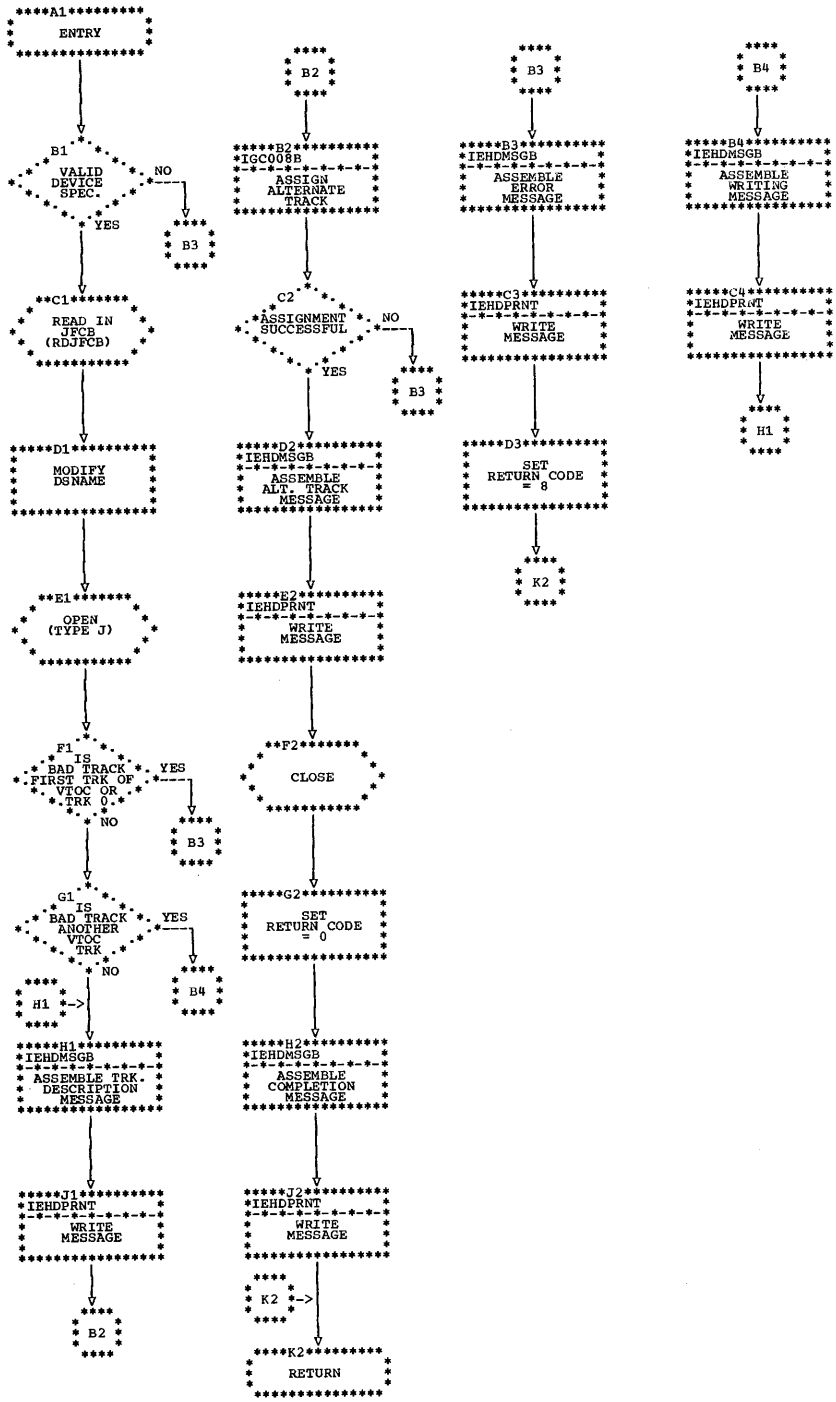


Chart 38. IEHDASDR Password Protection Routine

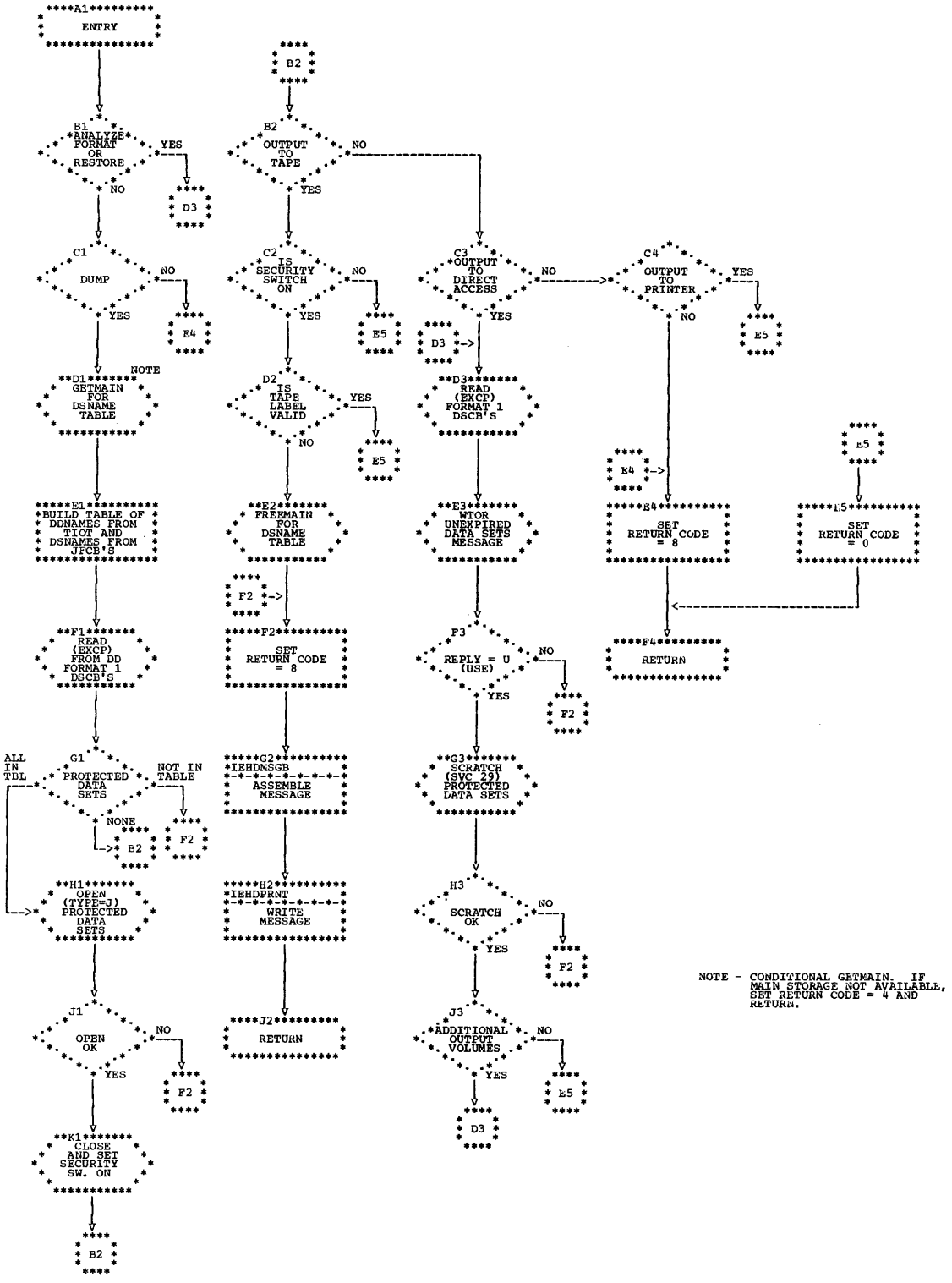
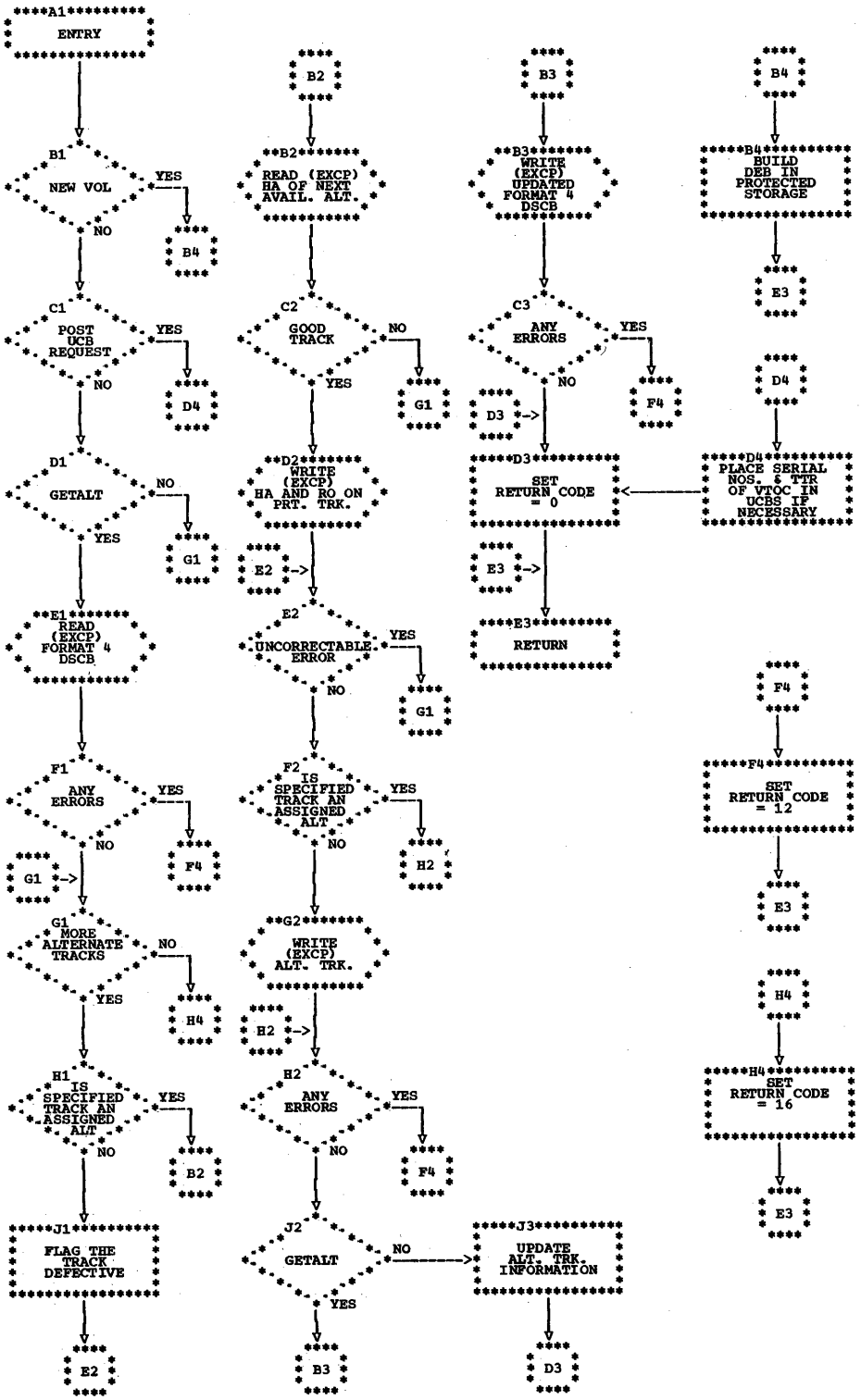


Chart 39. IEHDASDR SVC 82 Routine



Data Set Utility Programs

This section of the manual describes the nine data set utility programs: IEBCOPY, IEBCOMPR, IEBGENER, IEBPTPCH, IEBUPDAT, IEBUPDTE, IEBISAM, IEBEDIT, and IEBDG. These programs are executed under the Operating System/360. For their operation, however, they require user-supplied control statements in the input job stream.

IEBCOPY, IEBCOMPR, IEBGENER, and IEBPTPCH are designed as overlay programs, each consisting of three segments: the root segment, the control card analyzer segment, and the processor segment. The root segment alone is loaded initially; it links to the control card analyzer segment. When the control statements have been analyzed, control is returned to the root, which links to the processor segment.

The data set utility programs use QSAM for both reading the SYSIN data set and putting out the SYSPRINT data set. Both data sets may have a blocking factor that is other than one.

The storage requirements for buffers and tables are dynamically determined at execution time to optimize space allocation and thus permit the data set utility program to take full advantage of any storage that is available. If more storage is requested than can be supplied by the Main Storage Supervisor routines, the task is automatically terminated. If, however, the request cannot be filled immediately because of priority scheduling within a multi-tasking environment, the execution of the utility program can be delayed until its storage requirements are met.

Updating Partitioned and Sequential Data Sets (IEBUPDTE)

The IEBUPDTE utility program incorporates both IBM and user-generated source language modifications into sequential data sets or into partitioned data sets. The input and output data sets contain blocked or unblocked logical records of 80 bytes or less.

The program can:

- Add, copy, and replace members of data sets.
- Add, delete, replace, and renumber the records within an existing member or data set.
- Assign sequence numbers to the records of a member or data set.
- Create a sequential master data set from an input partitioned data set, and vice versa.

At the completion or termination of the program, the highest return code encountered within the program is passed to the calling program. The utility program can also produce a message data set containing a listing of the contents of the output data set, the control statements submitted to the utility program, and, if applicable, error messages.

Data definition (DD) statements needed to run the program are as follows:

- SYSUT1, which defines the old master data set (sequential or partitioned).
- SYSUT2, which defines the new (updated) master data set (sequential or partitioned).
- SYSIN, which defines a sequential data set containing the transactions to be applied to the old master data set.
- SYSPRINT, which defines a sequential data set containing either changes to the old master or contents of the new master, as well as utility control statements used and any error messages generated.

PROGRAM STRUCTURE

The program consists of three segments (or load modules): the root segment (IEBUPDTE), the control card analyzer segment (IEBASCAN and IEBBSCAN), and the intialization and exit routine modules (IEBUPNIT and IEBUPXIT).

The Root Segment

The main functions of the root segment are the processing of records and the printing of messages. The segment contains three control sections (CSECTs): IEBUPDTE, IEBUPLOG, and IEBUPDT2. The following text discusses the functions of each CSECT.

- IEBUPDTE receives initial control from the supervisor, obtains storage for the communication region IEBUPCON, and passes control to module IEBUPNIT for initialization. For writing a header

message on the SYSPRINT output device, CSECT IEBUPDTE passes control to CSECT IEBUPLOG. After the return (of control) from CSECT IEBUPLOG, CSECT IEBUPDTE gives control to CSECT IEBUPDT2 to begin the actual processing of records.

- IEBUPLOG is a closed subroutine, which writes messages and records on SYSPRINT. The first execution of IEBUPLOG opens SYSPRINT and the last closes it and returns control to the supervisor.

- IEBUPDT2, the heart of the program, opens the old and new master data sets, reads, processes, formats and writes the output records, and stows member names if the new master is partitioned. If user labels are to be processed on either SYSUT1 or SYSUT2, CSECT IEBUPDT2 passes control to module IEBUPXIT through data management routines during open, close, or end of volume processing. CSECT IEBUPDT2 passes control to the segment IEBASCAN to scan and analyze control statements, and to CSECT IEBUPLOG to log messages and records on SYSPRINT.

The Control Card Analyzer Segment

The main functions in analyzing control cards are performed by two CSECTs in this segment: IEBASCAN and IEBBSCAN. The following text discusses the functions of each CSECT.

- IEBASCAN scans and analyzes control statements and sets appropriate flags in the region IEBUPCON. CSECT IEBASCAN gives control to CSECT IEBUPLOG to print a copy of the control statement. To scan the individual parameters of each control statement, CSECT IEBASCAN gives control to CSECT IEBBSCAN with a doubleword parameter list, located at address STOREG in the communication region. For the reading of a control statement continuation card, CSECT IEBBSCAN gives control to CSECT IEBUPDT2. The first word in the list is the length of the last parameter analyzed by CSECT IEBBSCAN. The last word contains a pointer to the same parameter's location in a buffer area, SWITCHRD.

- IEBBSCAN, which is a closed subroutine, receives control from CSECT IEBASCAN and returns control either to CSECT IEBASCAN or to CSECT IEBUPDT2 (for reading another control card). CSECT IEBBSCAN scans the individual parameters on each control statement. If diagnostic messages are required as a result of the scanning, control is given to CSECT IEBUPLOG.

Initialization and Exit Routine Modules

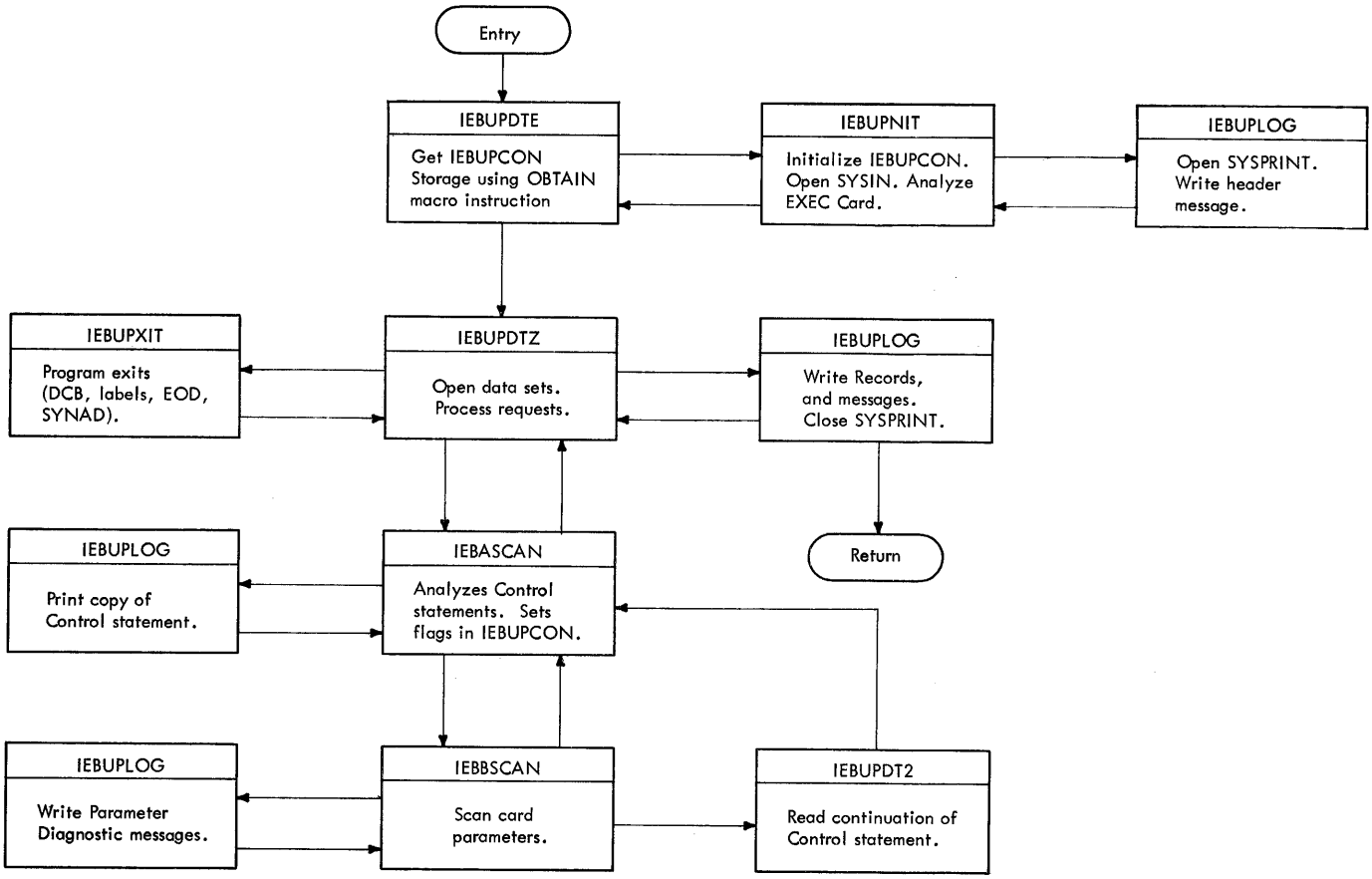
There are two modules in this group. They are described in the following text.

- IEBUPNIT is the initialization module. This module is a closed subroutine that receives control from CSECT IEBUPDTE for the purpose of initializing the region IEBUPCON, analyzing the parameters on the EXEC card, and opening the SYSIN data set.

- IEBUPXIT is the module containing the program's exit routines. For each of the three DCBs (SYSIN, SYSUT1, and SYSUT2), this module contains an entry point for each of the following closed subroutines: DCB exit, header-label exit, trailer-label exit, SYNAD exit, and end-of-data exit (there is no end-of-data exit for the SYSUT2 data set).

PROGRAM FLOW

Figure 41 shows the overall flow of the program; more detailed flow is shown in Charts 40 and 41.



• Figure 41. IEBUPDTE Overall Flow

PROCESSOR DATA FLOW

Figure 42 indicates the paths taken by data from SYSUT1 and SYSIN. The following is a breakdown of data flow within the processor (IEBUPDT2) according to the type of run: NEW or MOD.

NEW: This type of transaction involves reading data from SYSIN and writing it on SYSUT2 and (if specified) on SYSPRINT. Logical records are read in turn from SYSIN into the input buffer at SWITCHRD+1; records are then stacked in the SYSUT2 output buffer at NMWRITEP until the desired blocking factor is reached. A physical record is then written on the new master (SYSUT2). This process is repeated until SYSIN has been exhausted. If the SYSUT2 data set is partitioned (as indicated by the NAME keyword) the member name is stowed.

MOD: This type of transaction involves reading data from SYSUT1 (the old master data set) and SYSIN, merging them as indicated on function and detail statements, and writing the resultant data on SYSUT2. The updated master is written on SYSUT1 only when UPDATE=INPLACE is specified.

- For a REPRO run, a physical record is read from SYSUT1 into the buffer OMREADP, logical records are then moved individually to OMINAREA for inspection and then to the output buffer NMWRITEP until the desired output blocking factor is reached; the output record is then written on SYSUT2.
- For an ADD run, records are read from SYSIN and are stacked in the output buffer NMWRITEP until the desired blocking factor is reached. The output record is then written on SYSUT2. If the data set is partitioned, the member name specified is stowed in the directory of the new master. If numbering of records is specified, it is performed in the input buffer, SWITCHRD+1.
- For a REPL run, the flow is similar, except that the new data from SYSIN replaces the member specified. Numbering, if specified, is performed in the input buffer SWITCHRD+1.
- For a CHANGE run, which operates within a data set or member of a partitioned data set, records may be changed,

deleted, numbered or added, depending on the detail statements and data cards following the change statement. When one of these statements (DELETE, NUMBER, or DATA) has been read from SYSIN into the buffer SWITCHRD+1, a record is read from the old master (SYSUT1) into the buffer OMREADP and is processed as follows (see also Figure 42).

1. A logical old master record is moved from the buffer OMREADP to OMINAREA, and its sequence number (OM) is compared against SEQ1, if number or delete is in effect, and otherwise against the SYSIN record sequence number.

2. If OM is less than SEQ1, the old master logical record is moved to the output buffer NMWRITEP, and the next old master record is moved into OMINAREA.

3. If SEQ1 is less than or equal to OM, and OM is less than or equal to SEQ2, the old master is updated:

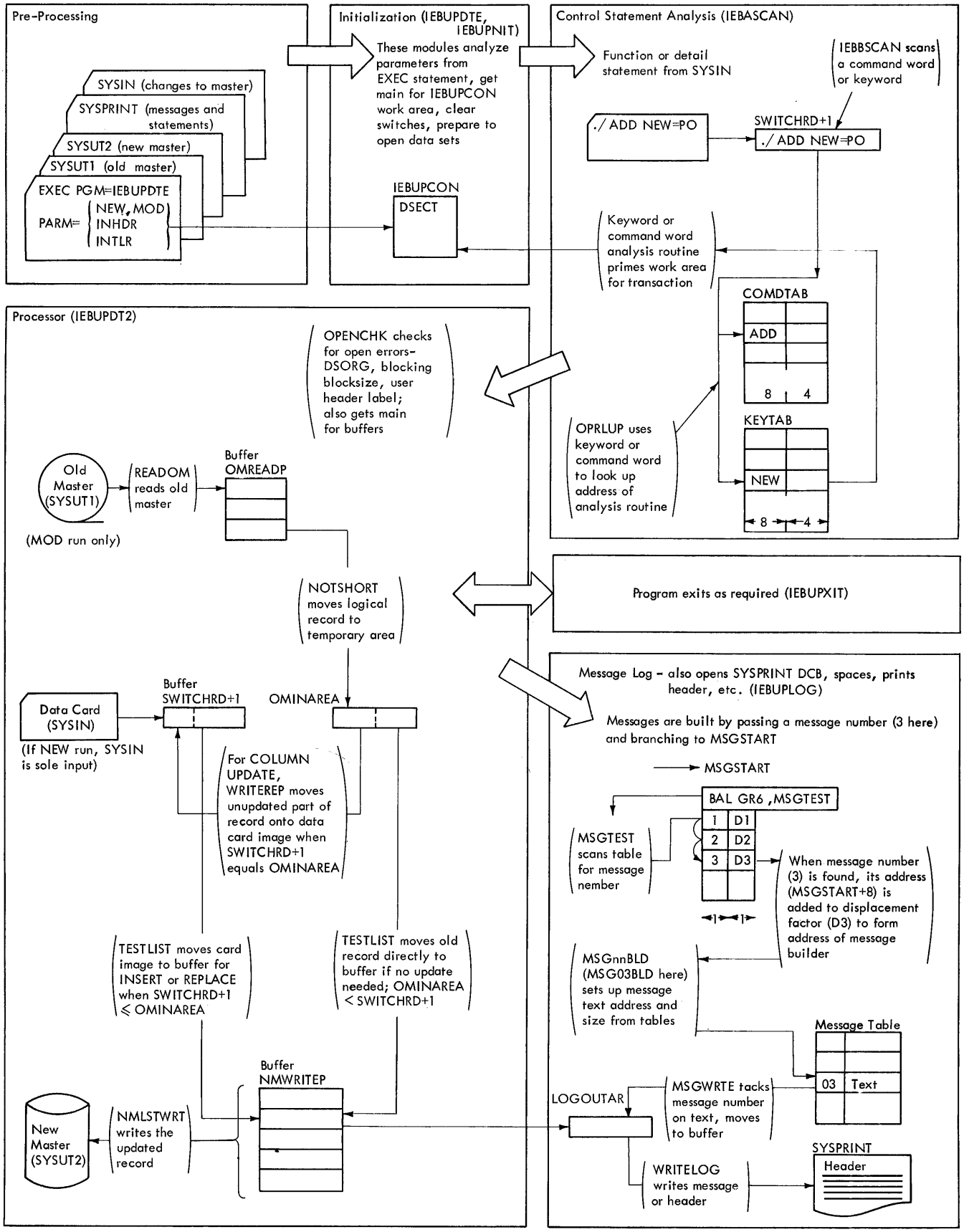
   - If the old master logical record is to be deleted, the next old master logical record is moved to overlay it in OMINAREA.

   - If data is to be inserted, (if the SYSIN sequence number is less than OM) the SYSIN data statement in SWITCHRD+1 is renumbered if necessary and moved to the output buffer NMWRITEP, and the next SYSIN record is read. If OM equals the SYSIN sequence number, the SYSIN record replaces the old master record.

   - If the SYSIN sequence number equals OM and COLUMN UPDATE was specified, the portion of the record in OMINAREA not to be updated is moved to its corresponding relative position in the buffer SWITCHRD+1, this updated record is renumbered if necessary and then moved to the output buffer NMWRITEP, and the next SYSIN and old master records are read.

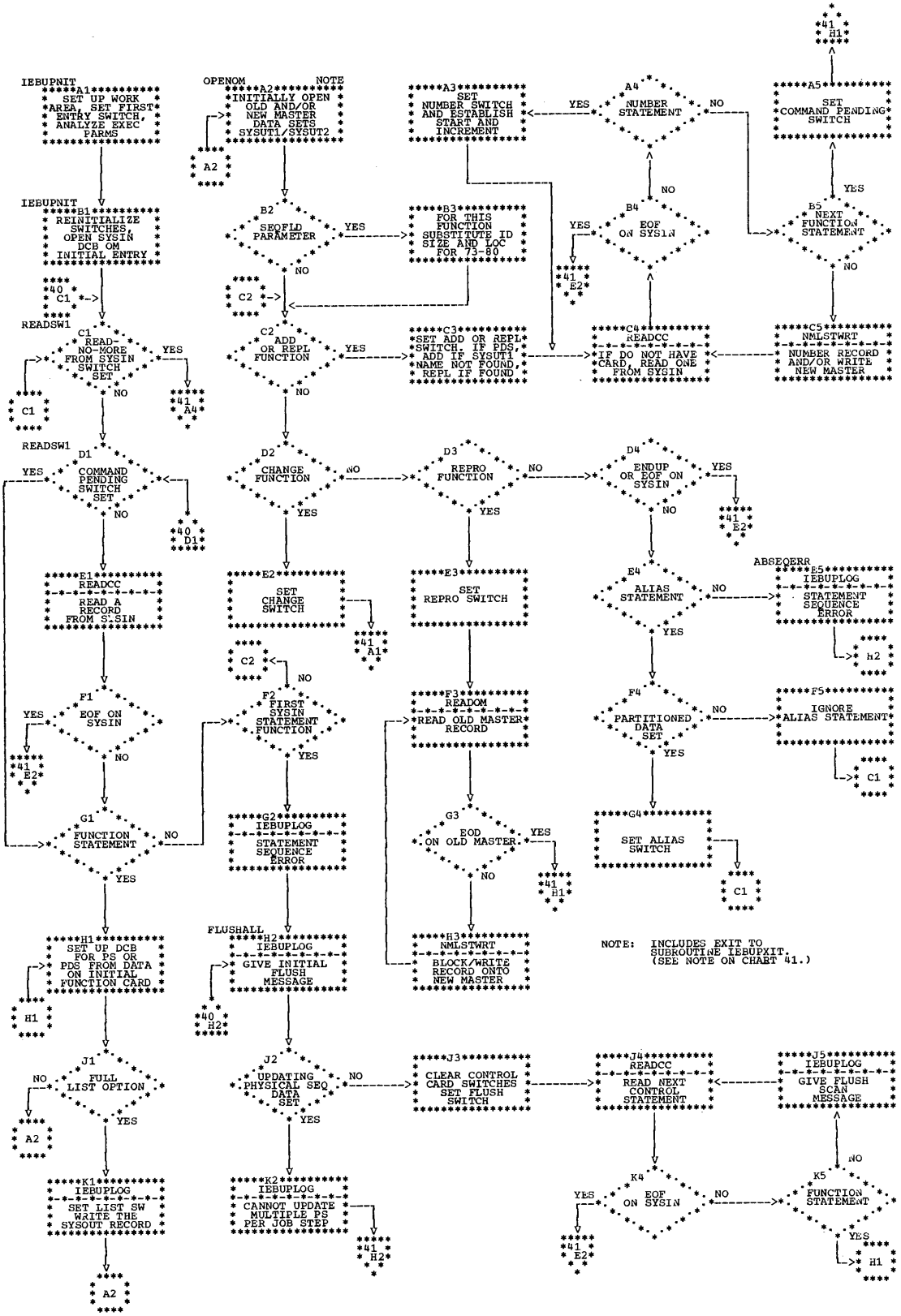
   - If the old master record is to be numbered, the indicated sequence number is stored in it, and the updated master record is moved to the output buffer NMWRITEP. The next old master logical record is then moved into OMINAREA.

4. When OM is greater than SEQ2, IEBUPDTE checks to see that record SEQ1 was actually processed (deleted or numbered), if it was not, an error message is written and the member update terminates. If it was, the next record from the old master is moved into OMINAREA, and the next record from SYSIN is read into SWITCHRD+1. Processing of the previous number or delete statement is considered finished.

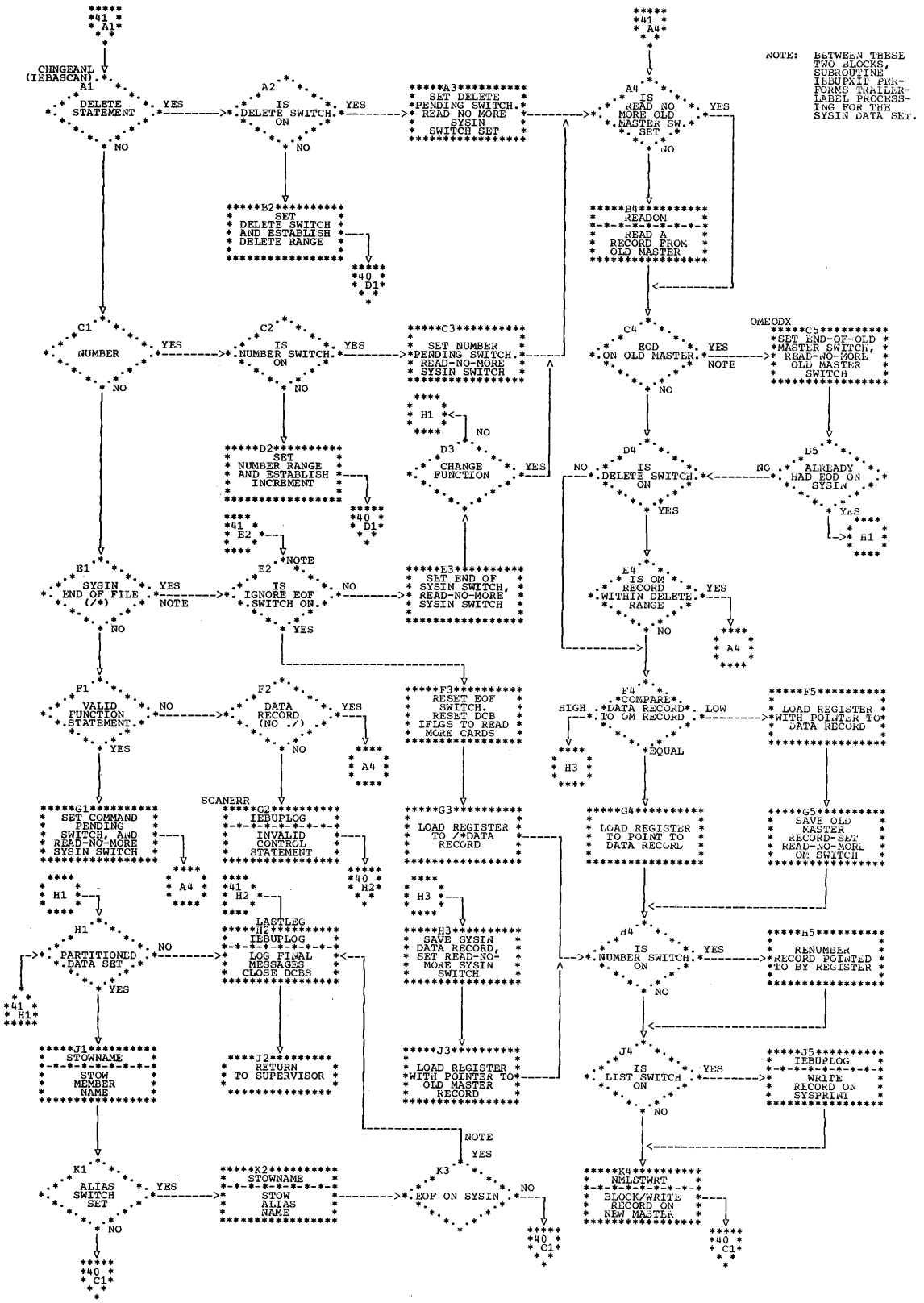


•Figure 42. IEBUPDTE Principle of Operation

• Chart 40. IEBUPDTE (Part 1 of 2)



• Chart 41. IEBUPDTE (Part 2 of 2)



Copying and Merging Partitioned Data Set Members (IEBCOPY)

The IEBCOPY program reproduces all or selected members of a partitioned data set. During the copy operation, physical data set compression (in-place recovery of unusable partitioned data set space) can occur since only the currently active members are processed. In addition, this program may be used to merge members from one data set into an already existing data set.

Input to the IEBCOPY program must be a partitioned data set. The data set must reside on a direct access device and be contained within one physical volume. The input records can be U, F, or V format. If F or V format, they can be blocked or unblocked. Keys, relative track address pointers (TTRNs) within the directory, and note lists are permitted.

The output of the IEBCOPY program is also a partitioned data set. It must reside on a direct access device and be contained within one physical volume.

PROGRAM STRUCTURE

The program (Figure 43) consists of three segments: the root segment, the control card analyzer segment, and the processor segment.

The Root Segment

The root segment initializes the program. It consists of routine IEBCOPYA.

IEBCOPYA
: stores optionally specified data definition (DD) names in a common table area for later insertion into their corresponding data control blocks (DCB) and inserts an optionally supplied initial page number into the page line to be written on the system print (SYSPRINT) data set.

The Control Card Analyzer Segment

The control card analyzer segment, IEBCOPYC, reads and processes the control cards. It consists of two routines: ANALY and ACTCCS.

ANALY
: calls ACTCCS to process the control cards. Based on the parameters supplied in the control cards the ANALYZER sets switches and builds parameter tables.

ACTCCS
: opens SYSIN; reads the control cards and passes the location, length, and identification of the parameters to the ANALY routine; and then closes SYSIN.

The Processor Segment

The processor segment, IEBCOPYD, performs the copying operation. It consists of nine routines: MAIN, BDIF, REJECT, TOTAL, FIRST, REBLOCK, SETOPSW0, EOD, and COMREAD.

MAIN
: saves the length of the member name table and performs initialization to force the reading of the entire input directory if member exclusion is requested. This routine also opens the input data set (SYSUT1) for reading by means of BPAM and determines, during the DCB exit, whether a total or a selective copy is requested. If a total copy is requested, the DCB parameters are saved and the accessing method (BPAM) is changed to BSAM to allow the directory to be read.

BDIF
: employs user-supplied member names and aliases to extract the corresponding entries from the input directory when an inclusive copy is requested.

REJECT
: compares all the names in the input directory against the list of user-supplied names when an exclusive copy is requested. If a match is obtained, the corresponding member is not processed.

TOTAL
: reads the input directory a block at a time into a buffer, calculates the length of the table required to store the directory entries, requests storage for the table, and rereads the input directory (exclusive of user data) into the table. At the conclusion of TOTAL, the input buffer is released and the address and length of the table is saved.

FIRST
: tests for TTRNs in the user data field of the directory and reads the note list, (if one exists) into the note list buffers. (See the publication, IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646 for a description of note lists.) Then, except for the compress function, the routine reads a normal record. For the compress function, this routine then gives control to the COMREAD routine.

REBLOCK
initiates read and write operations as required by the status of the in/out buffer and supplies the move (HMOVE) subroutine with the logical record length and the "from" and "to" addresses.

SETOPSW0
writes the member record (in original form, reblocked, or as an update note list) on the output data set (SYSUT2). (For the compress function, this routine is not used. The COMREAD routine does the writing of records in this case.)

EOD
stores the required data in the output directory (member names, aliases, user data, etc.).

COMREAD
performs the reading and writing operations when the compress function is specified. If note lists (records which contain pointers to blocks within a given member of a partitioned data set) are present, this routine will update them. The routine also reads and writes the member records (of a PDS) one track at a time and updates the TTRNs of a user data field when necessary.

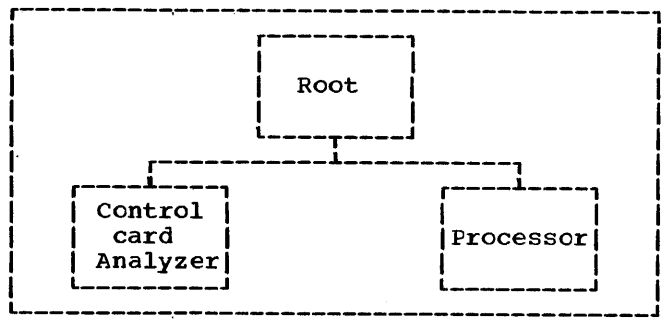


Figure 43. Overlay Structure of the IEBCOPY Program

PROGRAM FLOW

Charts 42 and 43 show the flow of control through the program. After the program is entered, it sets switches, assigns data areas, and opens SYSPRINT. The header line is written on SYSPRINT at this time using the optionally supplied initial page member.

The control card analyzer routine picks up the control statements from SYSIN and places them in tables within the IEBCOPY program.

A test is then made to determine if an exclusive copy was requested. For an exclusive copy, the user lists the names of the members that are not to be copied. The input data set directory is then read to determine the names of the members that are to be copied. If an inclusive copy is specified, all members listed are copied.

Next, the input data set (SYSUT1) is opened. If a total, an exclusive, or a compress copy is to be performed, the DCB parameters are saved and the basic sequential access method (BSAM) is used to read the directory. For the compress function, storage areas will also be allocated for use as buffers. Once the directory is read and all the entries are stored in a table, the access functions are performed either by using BPAM (for all but the compress function) or by using the XDAP (execute direct access program) macro instruction (for the compress function). For a description of the XDAP macro instruction, see the publication IBM System/360 Operating System: System Programmer's Guide, Form C28-6550.

The output data set is then opened and, during the DCB exit, the DCBs of the input (SYSUT1) and output (SYSUT2) data sets are checked for valid reblocking requests.

For a valid reblocking request, switches are set to establish a linkage to the reblocking routine. Space for the in/out buffer is also allocated at this time. If there is to be reblocking, a second buffer (in/out) is obtained. The length of the in/out buffer is equal to the input block size plus the key length.

The program is now ready for the names of the members that are to be copied. If the copy is to be either total, exclusive, or compress, the entire directory has already been read and saved. If the copy is inclusive, however, the member names and aliases which were provided by the user in the control statements and the corresponding entries are extracted from the directory at this time.

Directory entries, related to the members that are to be copied, are sorted and grouped by member name and physical disk address (TTR). A member name precedes all its aliases. If member exclusion is requested, the names in the directory are compared against the user-supplied names. When a match occurs, that member is not processed.

The user data field for the member name extracted from the input directory is interrogated. If the user data field contains note list pointers, a note list buff-

er is allocated and the note list is read to determine its length.

After the note list (if one exists) is read, the next processing steps depend upon whether the compress function has been specified.

Copying Without Data Set Compression

If data sets are to be copied without compression, a physical record is read into the in/out buffer. If reblocking is requested, a reblocking routine affects the new block size. The HMOVE subroutine is used to transfer logical records from the in/out buffer to the reblocking buffer from which the new block is written. When reblocking is not requested, physical records are written directly from the in/out buffer.

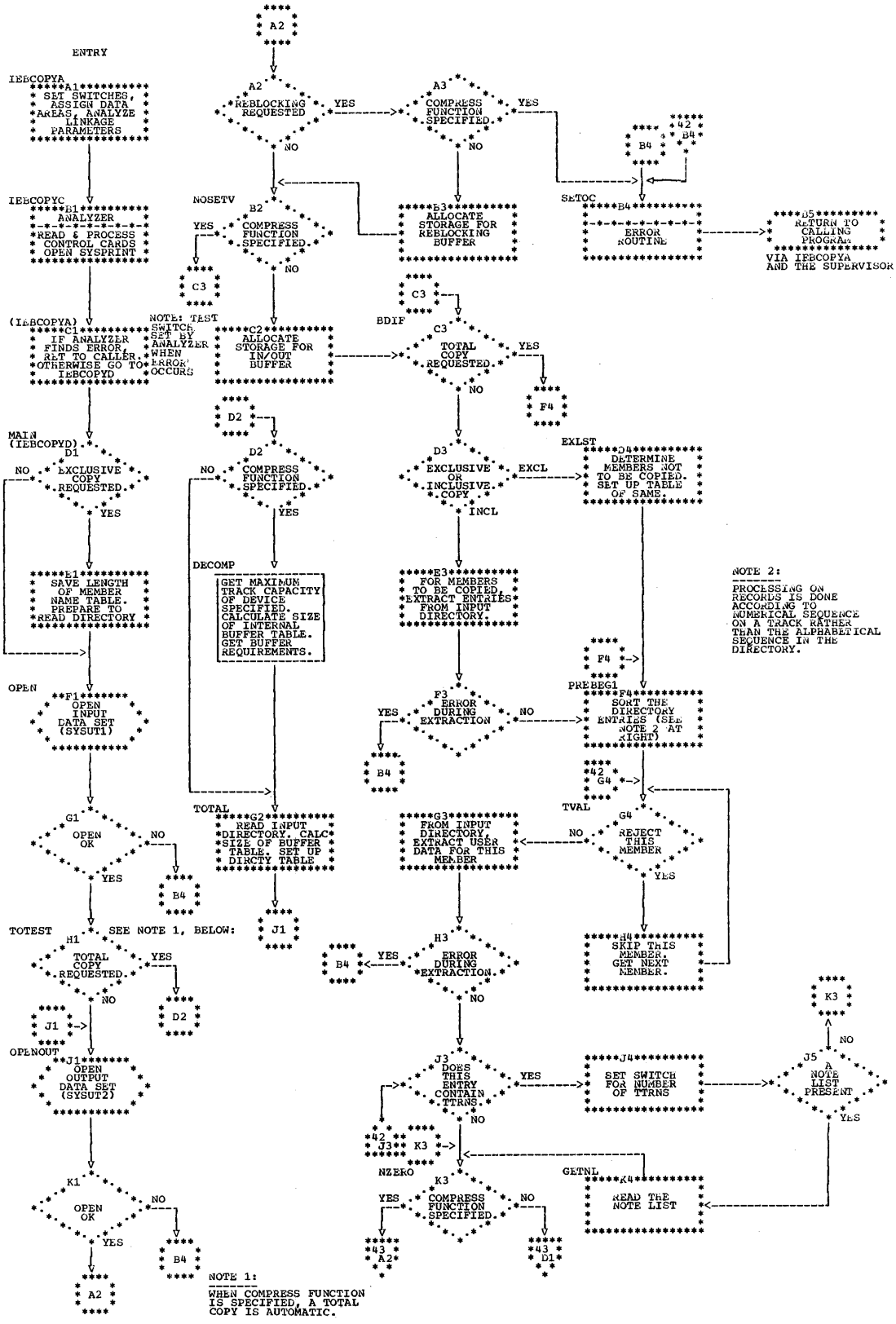
Before writing records for which reblocking has not been requested, the track address (TTR) for each physical record is compared to the entries within the note list. If a match occurs, a switch is set to indicate that pointers have been found that will require updating. After each physical record is written, the track address pointer (TTRN) for the output record is noted. This new (output) pointer replaces the former (input) pointer in either the directory entry or the note list (or both) depending on where it appeared in the input.

When the end-of-data for a member is reached, the member name and all aliases pertaining to that member are stored in the output directory. If the member name table indicates that more members remain to be copied, the copying process resumes. If the member name table is exhausted, job termination is initiated; registers are restored, a termination (normal or abnormal) message is written onto SYSPRINT, the proper return code is set, and control is returned to the control program.

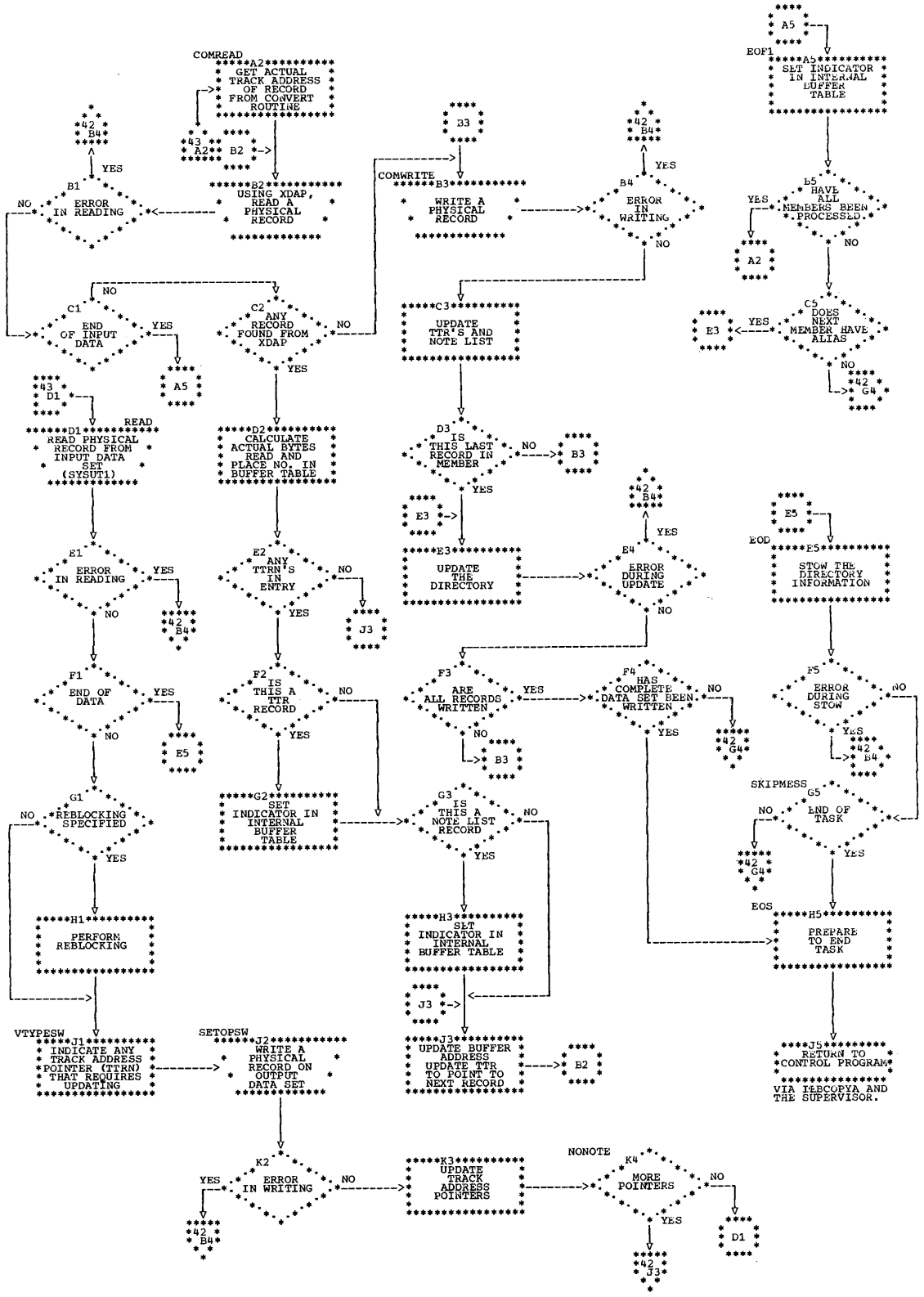
Copying With Data Set Compression

When data set compression has been specified (by the PARM = COMPRESS parameter on the EXEC control card), the COMREAD routine first uses a subroutine to convert the relative track address of a member record to an actual track address. Then the utility program obtains the blocksize (from the data set parameters) and uses the XDAP macro instruction to read the record into a buffer. The actual number of bytes read into the buffer is calculated from the residual byte count appearing in the channel command word. If the record ccntains TTRNs or is a note list, an indicating switch in the buffer table is set. After all records on a track have been read and inspected, they are written on the output data set (SYSUT2). When all the records of a member have been written, any TTRNs in the directory and any note lists are updated. Processing then continues as described for copying without the compress specification.

• Chart 42. IEBCOPY - Copying and Merging Partitioned Data Set Members (Part 1 of 2)



• Chart 43. IEBCOPY - Copying and Merging Partitioned Data Set Members (Part 2 of 2)



Comparing Records (IEBCOMPR)

The IEBCOMPR program compares either two sequential or two partitioned data sets at the logical record level. With one exception, data sets containing records greater than 32,756 bytes in length are compared at the physical record level. The records being compared can be U, F, V, or VS format. F, V, and VS format records may be either blocked or unblocked. For partitioned data sets, VS format records are not compared. If keys are present they are compared.

The utility program will use either QSAM (move mode) or BSAM processing to compare the records, depending on the following parameters describing the records of the data set: RECFM, logical record length, presence of record keys. (See Table 1 for details.)

All user header and trailer labels are compared unless control statements indicate otherwise. The program prints the labels if they are unequal. Optional user exits are provided so that the user can process his own labels.

PROGRAM STRUCTURE

The IEBCOMPR program (Figure 44) consists of three segments: the root segment, the control card analyzer segment, and the processor segment.

The Root Segment

The root segment consists of two control sections (CSECTS): IEBCOMPM and IEBCROOT. CSECT IEBCOMPM contains the standard messages for the IEBCOMPR utility program.

CSECT IEBCROOT consists of the two routines COMPARE, and LLEORI.

COMPARE
: sets all switches and tables to their starting or original values.

LLEORI
: opens SYSPRINT, writes the header line using the optionally supplied initial page number on SYSPRINT.

The Control Card Analyzer Segment

The control card analyzer segment reads and processes the control cards. It consists of two routines: IEBCANAL (containing control section ANALY) and IEBCCS02 (containing control section ACTCCS).

ANALY
: calls ACTCCS to process the control cards and then, based on the parameters supplied in the control cards, sets switches and creates parameter tables.

ACTCCS
: opens SYSIN, reads the control cards, and passes the location, length, and identification of the parameters to ANALY.

The Processor Segment

The processor segment performs the actual compare operation. It consists of the routines IEBCMAIN, IEBCQSAM, and IEBCULET. The routine IEBCMAIN contains six subroutines: DIRBUFF1, STARTBSA, SDSOBEG, READSET1, COMPAR, and BLPRT.

DIRBUFF1
: compares the directories of the input data sets if they are partitioned by

• Table 1. Access Methods Used for Comparing Records

Level of Comparison	Data Set	Records Have Keys	RECFM			Logical Record Length	Access Method
Physical Block	SYSUT1 SYSUT2	Yes Yes	VS VS			Not a factor	BSAM
Physical Block	SYSUT1 SYSUT2	No No	VS VS			Greater than 32,756 bytes in at least one data set.	BSAM
Logical Record	SYSUT1 SYSUT2	No No	VS VS	VS V	V VS	Less than 32,756 bytes for both data sets.	QSAM
Logical Record	SYSUT1 SYSUT2	Not a Factor	F F	U U	V V	Less than 32,756 bytes for both data sets.	BSAM

reading the directories and compares the member names.

STARTBSA
uses BSAM to open the data sets, SYSUT1 and SYSUT2, being compared, and obtains the necessary DCB information from each: block size, record length, record format, and key length. If user input header or trailer labels are saved to be compared as data when user input header or trailer label exits are taken during Open or End-of-Data processing, this routine compares the user header labels from both data sets and prints the labels if they are unequal.

SDSOBEG
examines the key lengths, the logical record lengths (F and VS formats only), and record formats of both data sets. Any discrepancy in the data sets results in an error message and termination of the task. If this routine determines that QSAM is required to process variable spanned (VS) records, it closes the data sets SYSUT1 and SYSUT2 and gives control to the routine IEBCQSAM to perform the processing.

READSET1
reads and deblocks physical records. Note: Deblocking on data sets with VS records is not done when comparing records whose length is greater than 32,756 bytes.

COMPAR
compares logical records. Unequal records are identified and printed. If a user routine is not provided and ten consecutive records fail to compare equally, this routine skips to the next member in each partiticned data set or terminates the task if the data sets are sequential.

BLPRT
prints internal hexadecimal data in Extended Binary-Coded-Decimal Interchange Code (EBCDIC) characters.

The routine IEBCQSAM contains the control section QSAM and processes data sets containing records that: do not have keys, are less than 32,756 bytes long, and are of format VS (see Table 1). In effect, this routine functions as a closed subroutine for routine IEBCMAIN, and it uses the subroutines COMPAR and BLPRT.

The routine IEBCULET contains the control section USERLAB. This routine, which functions as a closed subroutine of routine IEBCMAIN, saves, in main storage, the input header and trailer labels for both the SYSUT1 and SYSUT2 data sets. Routine IEBCULET is entered during the opening of, and when reaching the end of, the data sets SYSUT1 and SYSUT2. Exits to user input header and trailer label processing routines are taken from this routine.

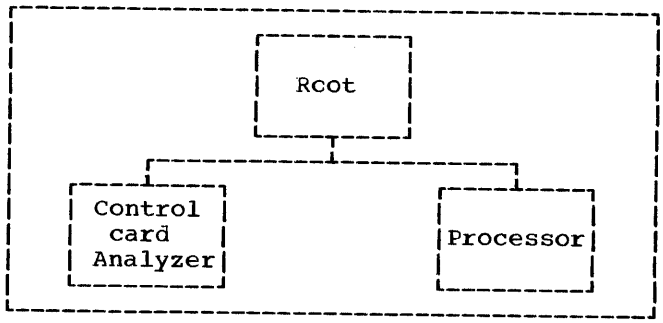


Figure 44. Overlay Structure of the IEBCOMPR Program

PROGRAM FLOW

Chart 44 shows the flow of control through the IEBCOMPR program. After this program is entered, it sets switches and tables to their original or starting values and opens SYSPRINT. A header is written on SYSPRINT at this time, using the optionally supplied initial page number.

The control card analyzer, ANALY, picks up the control statements from SYSIN and places them in tables within the IEBCOMPR program.

The ddnames for each data set are picked up from the ddname list and saved for later in the messages. Switches are also set at this time for each user exit that is specified.

The organization of the input data sets SYSUT1 and SYSUT2, can be either sequential or partitioned. If it is partitioned, storage must be allocated for tables. To determine the amount of storage needed, the program opens SYSUT1 with BSAM, reads the directory, and scans the user data field for member names, aliases, track address pointers, and note lists. When this is done, SYSUT1 is closed.

If SYSUT1 and SYSUT2 are partitioned, they are opened with BSAM and the directories are compared. Member names that compare equally are stored in the TNSET table. Member names that do not compare cause the member name with the lower binary value to be printed and assumed missing from the other data set. Also, user data fields for either member names or aliases that do not compare are printed.

Note list pointers associated with member names that compare equally are stored in tables TTRSET1 and TTRSET2 for SYSUT1 and SYSUT2, respectively. When the directory comparison is complete, SYSUT1 and SYSUT2 are closed.

At this point the program begins to compare logical records. The input data sets are opened and the necessary information is extracted from each DCB; i.e., block size, record length, record format, and key length. If a user exit is taken to compare, as data, the user input header labels from two sequential data sets, this routine performs the comparison of the appropriate labels. If the input data sets are sequentially organized, the user header labels from both data sets are compared unless control statements indicate otherwise. The program prints the labels if they are unequal.

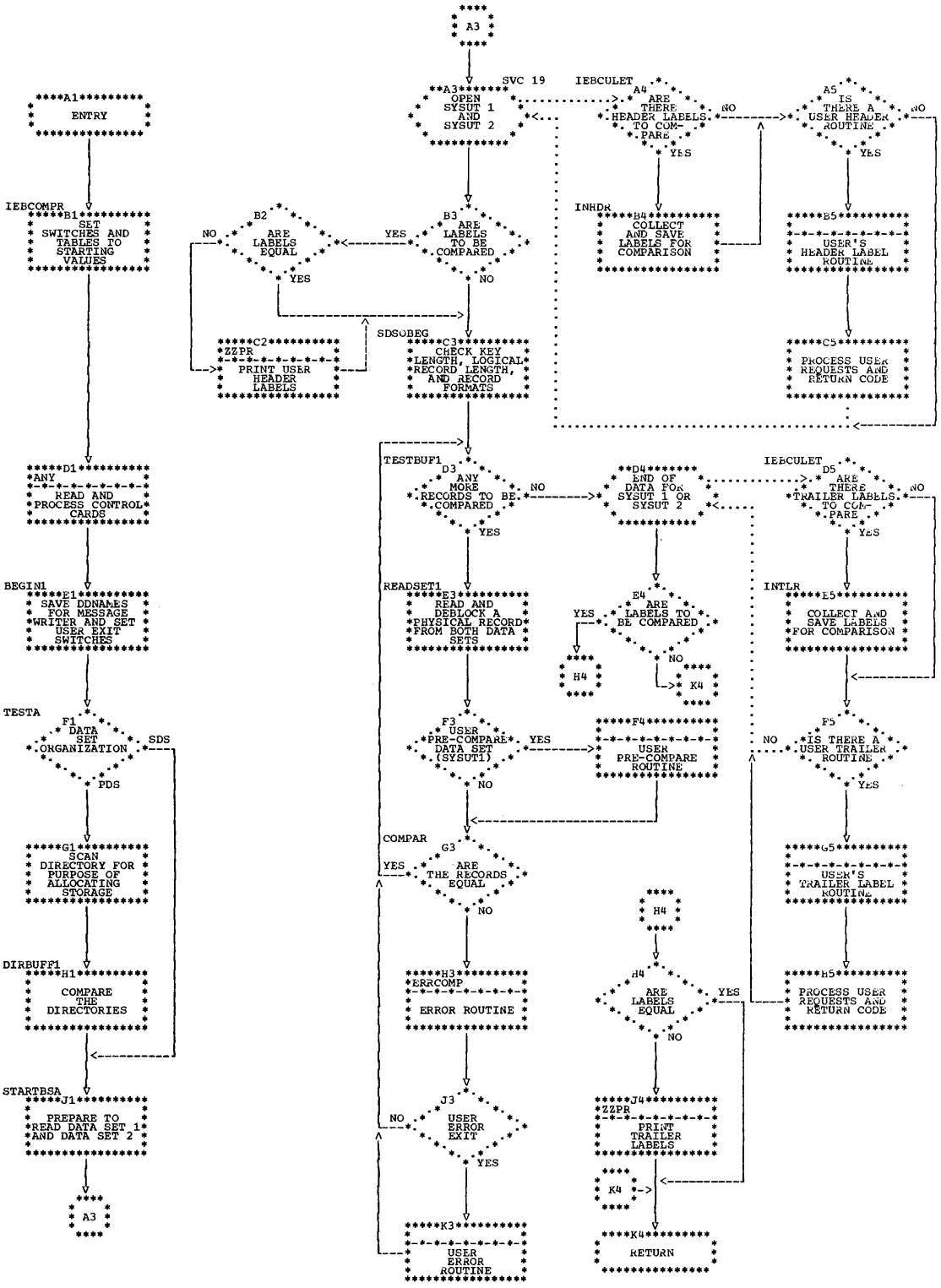
The record formats, the key lengths, and the logical record length (F and VS format records only) of the input data sets are compared. If there is any inconsistency, a message is printed and processing is terminated.

A physical record is read from each input data set and deblocked. (Note: Deblocking is not done when the data sets being compared have records whose lengths are greater than 32,756 bytes.) If there is no user pre-compare routine, a record from each data set is compared a character at a time until all the records are compared.

Records that do not compare are identified and printed. If a user error routine is provided, control is transferred to it. If a user error routine is not provided and this is the tenth consecutive error, processing either terminates if the input data sets are sequential or skips to the next member if the input data sets are partitioned.

After the last record is processed, the input data sets are closed; the total number of records compared is printed. If a user exit is taken to compare, as data, the user input trailer labels from two sequential data sets, this routine performs the comparison of the appropriate labels. If the input data sets are sequentially organized, the user trailer labels from both input data sets are compared unless control statements indicate otherwise. The program prints the labels if they are unequal.

• Chart 44. IEBCOMPR - Comparing Records



Copying and Modifying Records (IEBGENER)

The IEBGENER program copies a sequential data set, or converts a sequential data set into a partitioned data set, or adds members to an existing partitioned data set. Editing facilities are available with all operations of this program.

The input to the IEBGENER program must be a sequential data set. The data set can reside on any device. The input records can be U, F, V, or VS format. If F, V, or VS format, they can be blocked or unblocked.

The output of the IEBGENER program can be either a sequential or a partitioned data set. If the output data set is partitioned, it must reside on a direct access device and note lists will not be permitted.

PROGRAM STRUCTURE

The IEBGENER program (Figure 45) consists of three segments: the root segment, the control card analyzer segment, and the processor segment.

The Root Segment

The root segment initializes the program and writes messages on SYSPRINT. It consists of three routines (IEBGENER, HWRMSG, and HCDWR) and a message module, IEBDGMSG.

IEBGENER
: sets switches, assigns data areas, opens SYSPRINT, and writes the header line with a user supplied initial page number (if any) on SYSPRINT.

HWRMSG
: writes error messages on SYSPRINT.

HCDWR
: writes, on SYSPRINT, the control cards that are read by the control card scanner (IEBGSCAN) routine.

IEBGMESG
: contains the text of error messages that are written by HWRMSG.

The Control Card Analyzer Segment

The control card analyzer segment reads and processes the control cards. It consists of two routines: IEBGSCAN and IEBCCS02.

IEBGSCAN
: calls IEBCCS02 to process the control cards and then, based on an analysis of the parameters supplied in the control cards, IEBGSCAN sets switches and creates parameter tables for use by the processing modules IEBGENR3, IEBGENS3, and IEBGEN03. The addresses of the tables are in a list to which general register 1 points when this routine has finished its processing.

IEBCCS02
: opens SYSIN, reads the control cards and then passes the location, length, and identification of the parameters to IEBGSCAN.

The Processor Segment

The processor segment consists of a root module, IEBGENR3, and two processing modules, IEBGENS3 and IEBGEN03. The root module opens and closes the data sets and performs all label processing. It gives control to either of the other two modules (IEBGENS3 and IEBGEN03) for editing and copying functions. Module IEBGENS3 is used for variable spanned records and IEBGEN03 is used for all other record formats. The entire segment consists of these three modules and the following routines: IEBEDIT2, IEBLENP2, IEBMOVE2, IEBCONH2, IEBCONP2, and IEBCONZ2.

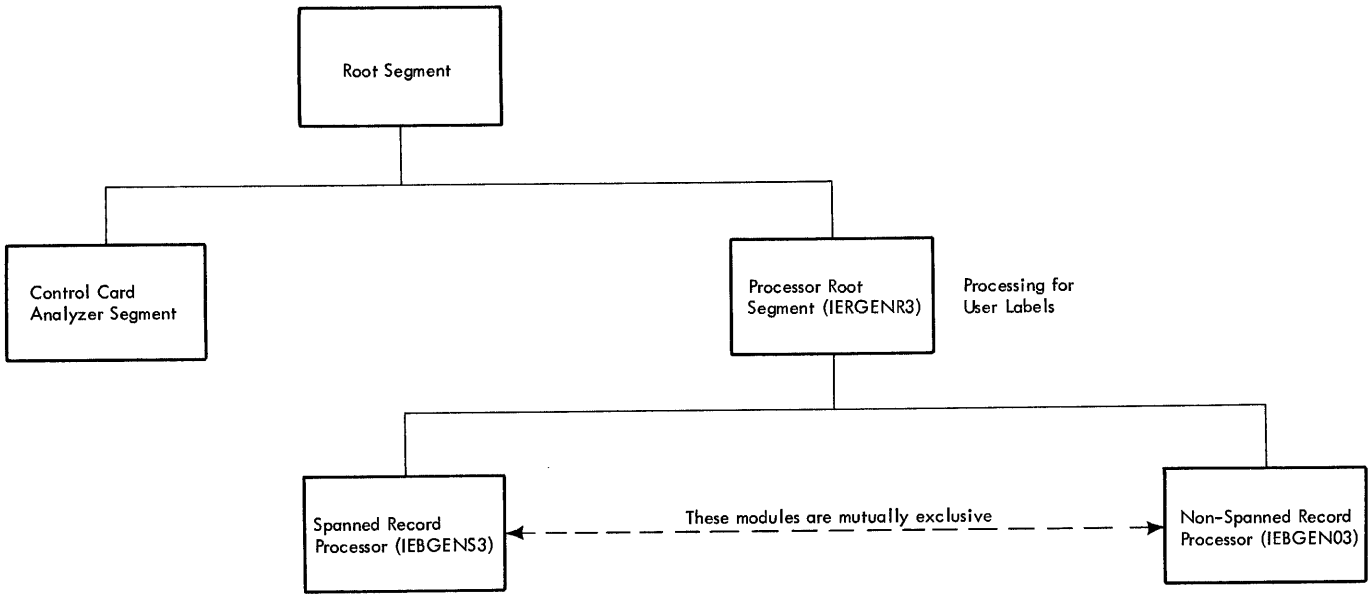
IEBGENS3
: for variable spanned records, this processing module either gets and puts logical records or reads and writes physical blocks, depending on DD card parameters and/or information in the data set control block. The module links to editing and/or conversion subroutines as required by control statements. It returns control to the root module.

IEBGEN03
: for all but variable spanned records, this module reads the input from the SYSUT1 data set, deblocks the records, edits them if required, and writes the output on SYSUT2 with proper blocking. The module links to editing and/or conversion subroutines as required by control statements. It returns control to the root module.

IEBEDIT2
: moves the logical records from the input buffer to the output buffer with field editing. One field is moved at a time, and converted if necessary.

IEBLENP2
: calculates the total length of the output records based on the lengths of the fields to be moved. Conversion is then performed on each.

IEBMOVE2
: moves bytes of data from one area of main storage to another.



•Figure 45. Overlay Structure of the IEBGENER Program

IEBCONH2
converts the data from H-set BCD to EBCDIC characters.
IEBCONP2
converts the data from packed to zoned decimal format.
IEBCONZ2
converts the data from zoned to packed decimal format.

Charts 45 and 46 show the flow of control through the IEBGENER program. After the program is entered, it sets switches, assigns data areas, analyzes linkage parameters, and opens SYSPRINT. A header line with user initial page number (if any) is written on SYSPRINT at this time.

The control card analyzer, IEBGSCAN, picks up the control statements from SYSIN and places them within the IEBGENER program.

The DD name for each data set is picked up from the DD name list and stored in the HDDNAMES table. Then the input (SYSUT1) and the output (SYSUT2) data sets are opened. A user exit may be taken at this point to process user header labels.

Next, a physical record is read into the read buffer and then moved to the input work area for deblocking and processing. At this point, the record is available to the user via a user exit

The program reads the next physical record from the input data set to refill the vacated input buffer.

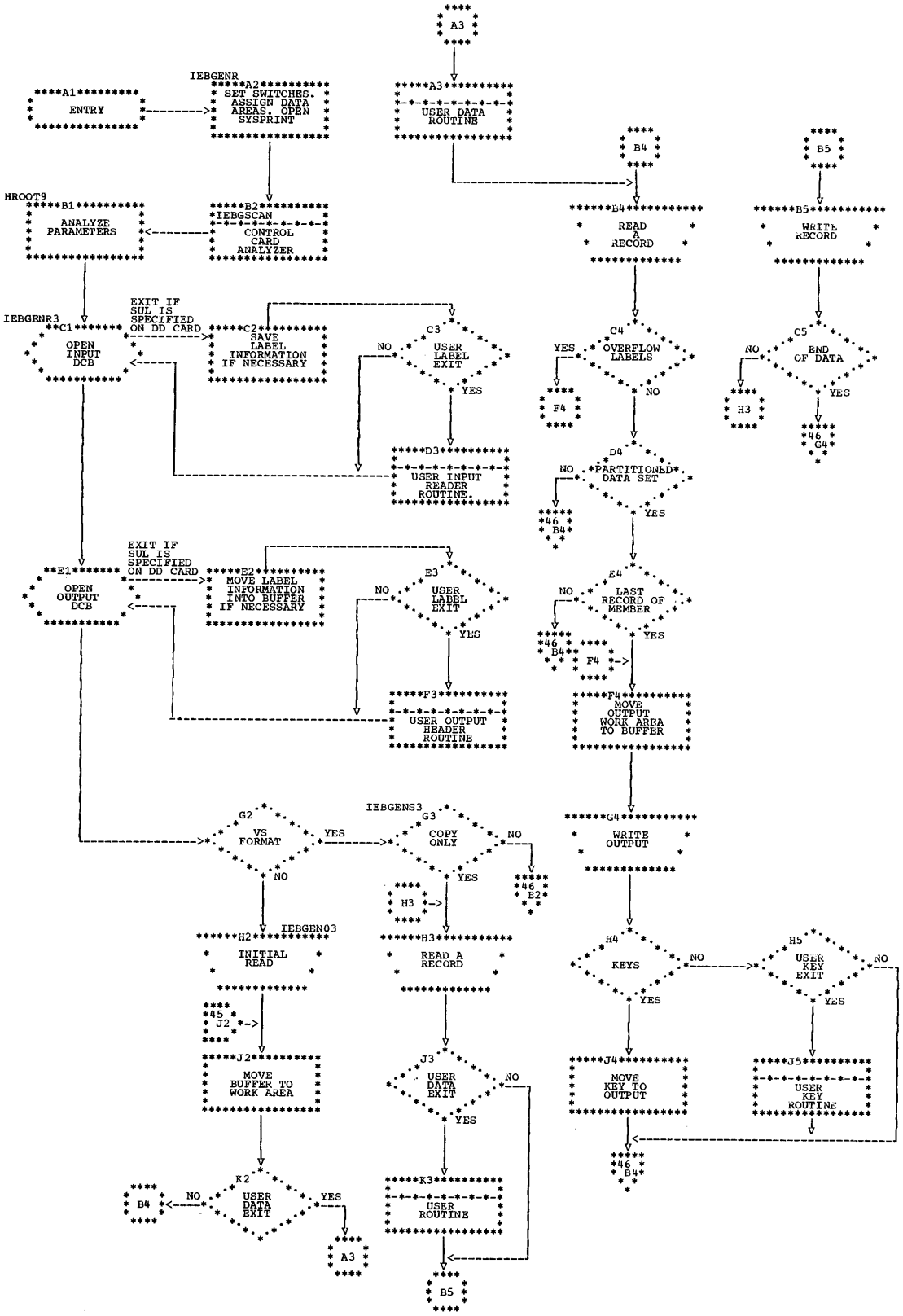
Logical records are moved one at a time to the output work area. If editing is requested by the user, the requested conversion of each field of each logical record is performed.

A test is performed before a record is moved from the input work area to the output work area to determine whether space is available in the output work area. If space is not available or if the output work area contains the last record of a partitioned data set, records in the output work area are moved to the output buffer and written on the output data set. If the output contains keys, the keys are also written out. A user exit permits the user to insert keys.

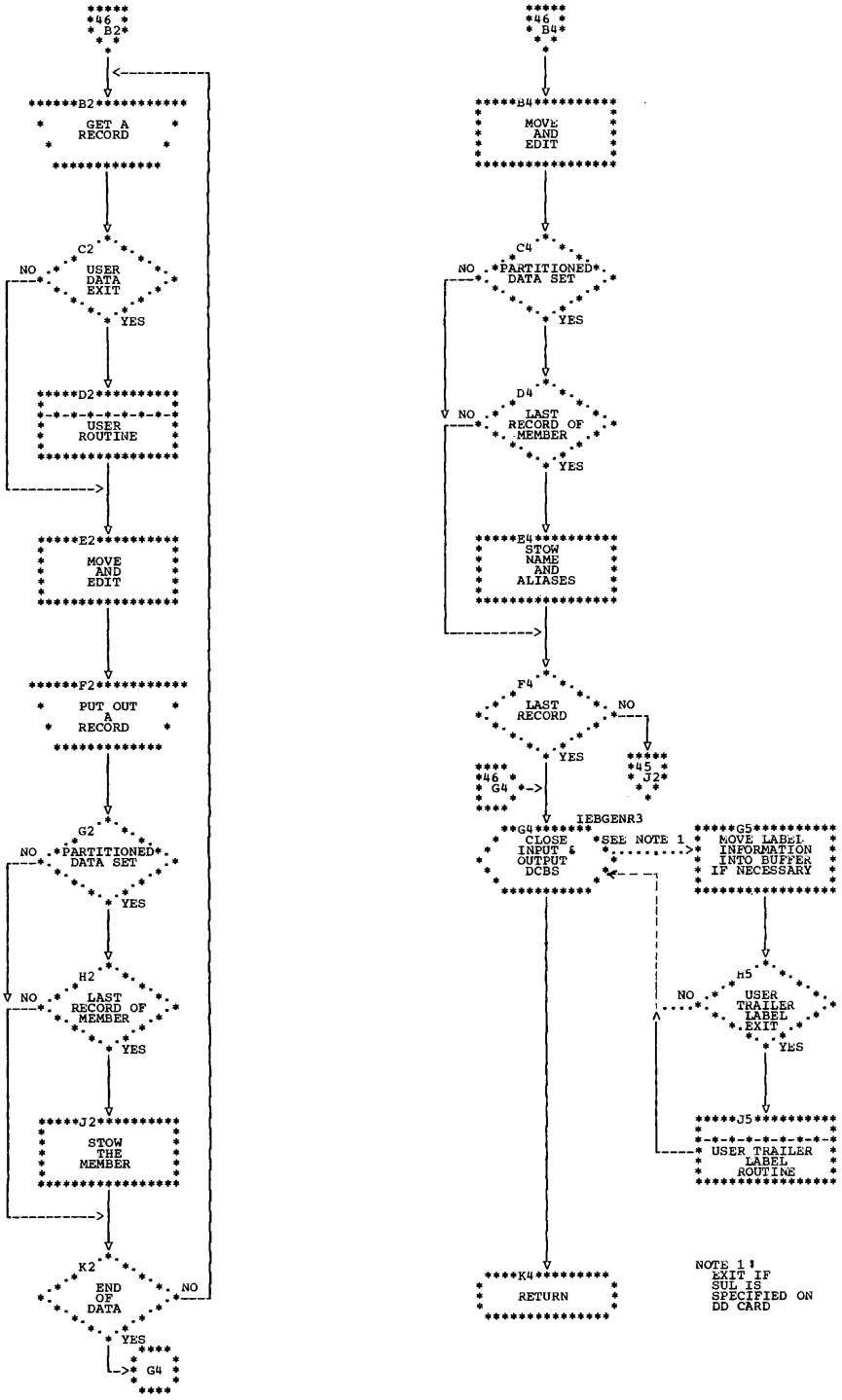
A test follows the movement of each record from the input to the output work areas to determine whether the output data set is partitioned or sequential. If the output data set is partitioned and the last record for a member was previously written, the member name and aliases are stored in the directory.

After the last record is processed and written, the input and output data sets are closed. During the closing a user exit may be taken to process user trailer labels. Control is then returned to the invoker.

• Chart 45. IEBGENER - Copying and Modifying Records (Part 1 of 2)



• Chart 46. IEBGENER - Copying and Modifying Records (Part 2 of 2)



Printing and Punching Records (IEBPTPCH)

The IEBPTPCH program prints or punches all or selected portions of a sequential data set, a partitioned data set, or specified members of a partitioned data set.

The input to the IEBPTPCH program can be either a sequential or a partitioned data set. The input records can be U, F, or V format. If F or V format, they can be blocked or unblocked.

The output of the IEBPTPCH program is put on a printer or a card punch. Note lists are permitted in the output only when the standard format is used.

PROGRAM STRUCTURE

The program (Figure 46) consists of three segments: the root segment, the control card analyzer segment, and the processor segment.

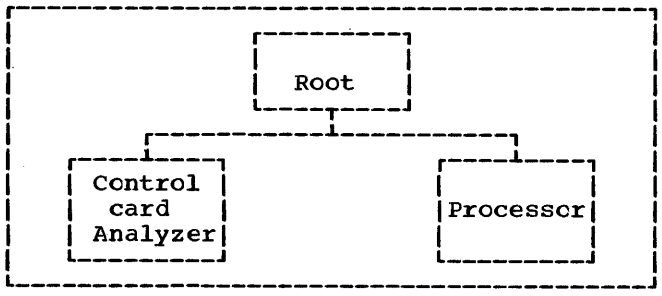


Figure 46. Overlay Structure of the IEBPTPCH Program

The Root Segment

The root segment initializes the program, and consists of one routine, PRPCH, which links to PPANAL.

The Control Card Analyzer Segment

The control card analyzer segment reads and processes the control cards. It consists of two routines: PPANAL and ACTCCS.

PPANAL
: calls ACTCCS to process the control cards and then, based on the parameters supplied in the control cards, sets switches and creates parameter tables.

ACTCCS
: opens SYSIN, reads the control cards and then passes the location, length, and identification of the parameters to PPANAL.

The Processor Segment

The processor segment performs the printing and punching operations. It consists of twelve routines: PRPUN, TOTAL, MEMBLOC, PPSDS1, RDCH, PREFORM, RECDLOC1, RECPROC, RECPREP, FORMS, FORMU, and CLOSEI0.

PRPUN
: examines the parameters supplied by the PPANL routine in the control card analyzer segment and performs initialization based on these parameters.

TOTAL
: reads the directory, extracts the name and location of each entry, and sorts the entries by TTR and alias indicator so that members can be written in the order of their physical occurrence in the data set and written only once.

MEMBLOC
: obtains the name and location of the next partitioned data set member to be written and then positions the data set so that the member can be read.

PPSDS1
: determines whether there are user written record groups. If no editing is indicated, it prepares to write the sequential data set or the member in the standard format. It also prepares to skip logical records within the member or the sequential data set.

RDCH
: reads a physical record. If note lists are to be omitted and the current record is a note list, another physical record is read.

PREFORM
: deblocks and writes out the records if PREFORM is specified.

RECDLOC1
: deblocks the physical record.

RECPROC
: initiates logical record processing, examines the identification (ID) of the record to determine if it is last record in a group, examines the logical record count to determine if the record should be skipped, and provides the user access to the input record.

RECPREP
: tests for the end of page on printed output and determines the format for the current logical record.

FORMS
: writes a logical record in the standard format. If necessary, it seg-

ments the input record into multiple output records.

FORMU
edits a logical record in accordance with user specifications. Before the record is written, the user can again access the output record.

CLOSEIO
prepares to end the task and relinquish control to the control program or the invoker.

PROGRAM FLOW

Chart 47 shows the flow of control through the IEBPTPCH program. After the program is entered, it sets switches, assigns data areas, and analyzes the internal system-provided parameters. A header is written on SYSPRINT at this time, using the optionally supplied initial page number.

The control card analyzer routine (PPANAL) picks up the control statements from SYSIN and places them in tables within the IEBPTPCH program. The output data set, (SYSUT2) is then opened for printing or punching.

If the input data set, SYSUT1, is partitioned and the entire data set is to be read and processed, the program reads the directory and extracts the name and location of each entry. The entries are then sorted by TTR and alias so that members can be written in the order of their physical occurrence on the direct access device.

Next, initialization is performed to enable the input data set to be read. A user exit can be taken at this point to process the user header label on the input data set if it is sequentially organized.

If the user's routine returns an action code of 16, the utility program will complete the opening of the input data set, print and punch (if so specified) any header labels that have already been read (up to the point for which the action code was set), close the input data set, and terminate the processing. The utility will then return control to the supervisor.

If the input data set contains variable spanned records, the DCB exit routine, during the opening of the SYSUT1 data set, tests the record length and the record format parameters. The action taken is indicated in Figure 47.

After the data set has been opened, the access method indicator field in the DCB is set to indicate the use of the QSAM MOVE mode.

If the input data set is partitioned, the name and location of the member that is to be processed is obtained and the data set is positioned so that the member can be read.

Any user-supplied titles are written at this time. If the input is partitioned, the member currently being processed is identified. A new page is started for printed output or a new sequence number is initiated for punched output. The program then determines whether there are user written record groups and performs initialization accordingly. If there is no editing, initialization is performed to process the sequential data set or member in the standard format.

The RDCH routine reads a physical record and determines whether a note list is present. If the physical record is a note list and it is to be omitted, the routine reads the next physical record when the basic sequential access method is being used. When the queued sequential access method (QSAM) is used, the routine gets a logical record. QSAM is used only for a sequential data set having both a logical record length that does not exceed 32,756 bytes and variable spanned records.

The PREFORM routine deblocks and writes out records if the user has control charac-

DCB Parameter		Action Taken
RECFM	LRECL	Work Area for IEBPTPCH
VS or VBS	Greater than 32,756	RECFM field in utility work area is set to U. LRECL field in utility work area is filled with the DCB blocksize.
VS or VBS	Equal to or less than 32,756	RECFM field in utility work area is set to V. BLKSIZE field in utility work area is filled with the DCB logical record length.

•Figure 47. Work Area Settings for Support of Variable Spanned Records

ters in the input data set and specifies the keyword PREFORM. All other control statement requests are ignored, but are checked for validity.

The RECDLOC1 routine deblocks the physical record and obtains the length and location of the next logical record. When no logical records remain in a block, the RECDLOC1 routine returns to the RDCH routine, and another physical record is read.

A user exit can be taken at this point to process a logical record before it is processed by the program.

The processing of a logical record includes checking the record ID to determine whether it is the last record in a record group and testing the record count to see if the record should be skipped. If the record is the last record of a record group, a switch is set for subsequent testing. If the record is to be skipped, control is passed to the end of record group test.

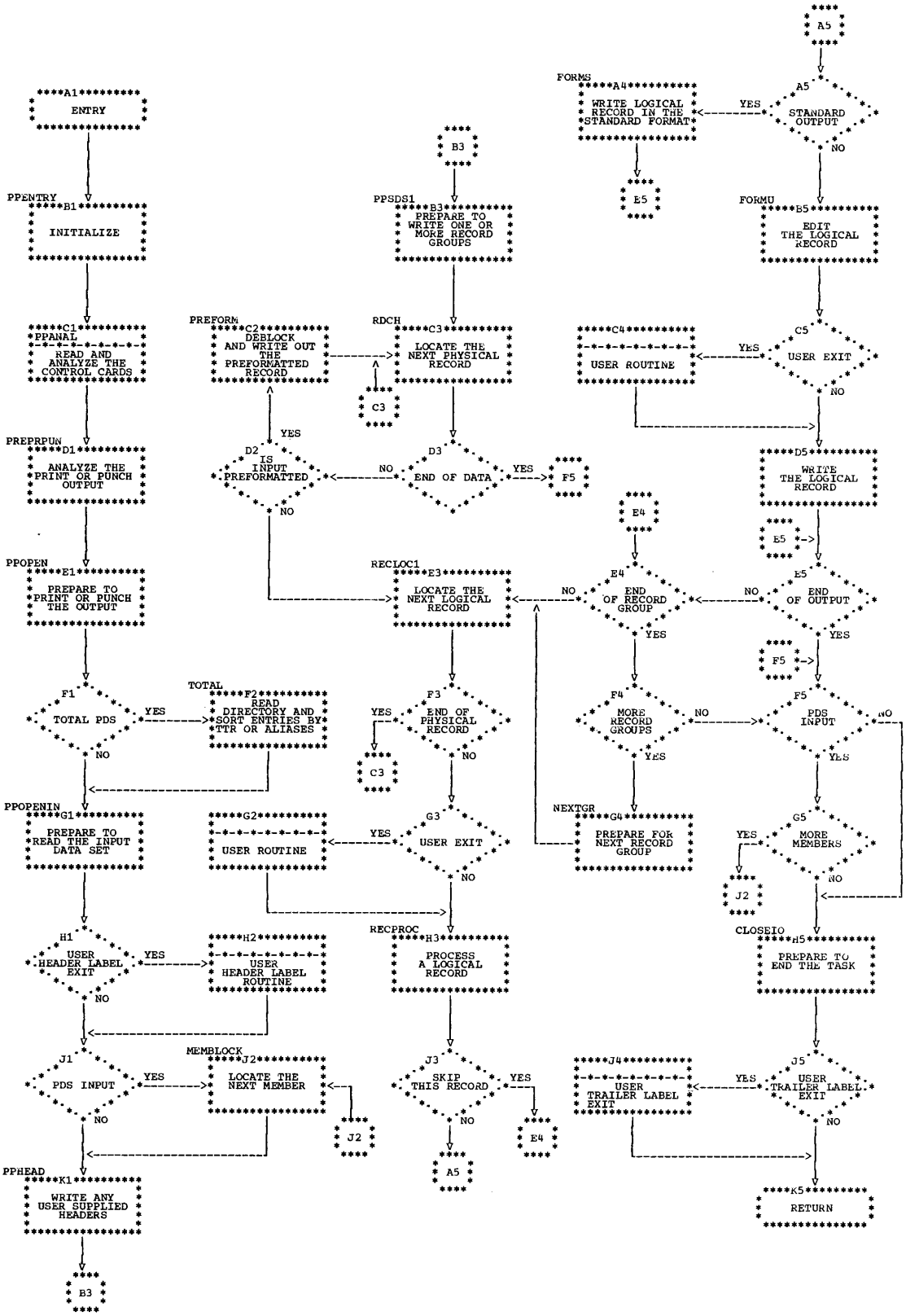
Next, the output format of the logical record is tested to determine if it is to be standard or user defined. The FORMS routine writes a logical record in the standard format; and when necessary, segments the input record into multiple output records. The FORMU routine edits a logical record a according to user specifications. A user exit may be taken before the record is written to allow the user to perform additional editing.

After the last record in each record group is written, the NEXTGR routine performs reinitialization to allow the next record group to be processed.

When the end of data is reached on an input partitioned data set, the name and location of the next member is obtained and the data set is positioned to the next member. When the end of data for the last member or for a sequential data set is reached, the input data set is closed. A user exit can be taken at this point to process the user trailer on the input data set if it is sequentially organized.

The processing of trailer labels employs the same use of return action code 16 as described in this section for header label processing.

• Chart 47. IEBPTPCH - Printing and Punching Records



Operating on an Indexed Sequential Data Set (IEBISAM)

The IEBISAM program is executed under the operating system to copy, unload, load, or print an indexed sequential data set. As examples, this program can be used to create a back-up copy of a data set, or to improve the accessibility of a data set by eliminating wasted track space and overflow areas. The program, which may be either executed as a job step or called by an executing program, consists of six load modules (see Chart 48) that reside in the linkage library, LINKLIB.

The Initializing routine determines which function has been specified, then passes control to one of four functional (or processing) modules. The selected processing module performs its specified function, then passes control to the Terminating routine, which writes messages, terminates processing, and returns control to the calling routine. (Note: If invalid specifications or parameters have been specified, the Initializing routine sets the appropriate message and completion code indications and gives control directly to the Terminating routine.)

The IEBISAM program may be executed as a job step, or it may be called by a program executing a job step. If it is to be executed as a job step, the step's EXEC statement specifies the program IEBISAM, and the EXEC statement's PARM field specifies the function to be performed as a parameter (COPY, UNLOAD, LOAD, or PRINTL). If the IEBISAM program is to be called by a program executing a job step, the calling program must specify the function by providing 'EXEC statement parameters' and ddnames as shown in this publication in the section "Auxiliary Parameters." In either case, the job control language statements that describe the step during which IEBISAM is to be executed must include DD statements to define the input, output, and message data sets.

Figure 48 gives a module directory and summary for the IEBISAM program, and Charts 49-56 outline the individual routines of the program. For more information regarding the use of the program, refer to the SRL publication IBM System/360 Operating System: Utilities, Form C28-6586.

INITIALIZING IEBISAM

The Initializing routine is in the load module (IEBISAM) that is entered whenever the IEBISAM program is requested. This routine (Chart 49) obtains main storage for a work area, then inspects the specifications under which the program is to run.

If no options have been specified in the PARM field of the EXEC statement, the program assumes the (default) option to unload the data set. If a function specification is not valid, this routine stores a completion code, assembles a message, and uses the XCTL macro instruction to pass control to the Terminating routine, IEBISF.

In all other situations (i.e., those in which correct procedures have been followed), the Initializing routine assembles the necessary information and gives control to the appropriate module.

COPYING AN INDEXED SEQUENTIAL DATA SET

If the copy function was specified, control is passed to the Copy routine in module IEBISC. This routine creates an output data set containing the same records as the input data set, but with newly built indexes and empty overflow areas. The Copy routine (Chart 50) opens the input data set (SYSUT1) and the output data set (SYSUT2) for use by QISAM and checks the DCB parameters:

- The DCBLRECL parameters must be the same for both the input and the output data sets.
- The DCBRECFM parameters must be the same (F or V) for both the input and the output data sets.
- For the output data set, the DCBBLKSI parameter must be a multiple of the DCBLRECL parameter.
- The DCBRKP parameter must be smaller than the DCBRECL parameter minus the DCBKEYLE parameters.

If the input and output data sets are opened successfully, and the DCB parameters are valid, the Copy routine uses the PUT (locate mode) and the GET (move mode) macro instructions to read the records in logical sequence from the input data set and write them into the output data set.

If the data sets are not opened successfully, if the DCB parameters are not valid, or if an unrecoverable input/output error is encountered, the routine stores both a completion code and a message code. Processing on the data set is terminated; the data sets are closed; and control is given to the Terminating routine.

Module ID	CSECT	Summary	Chart ID
IEBISAM	IEBISAM	Receives control from calling routine. Gets work area used by processing modules. Gets program parameters, alternate ddnames, page number. Passes control to appropriate module.	49
IEBISC (Copy)	IEBISC	Produces copy of input data set with new indexes. Uses PUT and GET macro instructions.	50
IEBISU (Unload)	IEBISU	Retrieves an indexed sequential record and passes its length and address to IBISSO. Analyzes return code from IEBISSO and sets success or error indication for IEBISF.	51
	IEBISSO	Unloads the indexed sequential record(s) into physically sequential 80-byte card images.	52
IEBISL (Load)	IEBISL	Reconstructs an indexed sequential record from 'unloaded' data passed by IEBISSI. Checks DCB parameters OPTCD, RECFM, LRECL, BLKSIZE, RKP, NTM, KEYLEN, and CYLOFL against corresponding DD statement information.	53
	IEBISSI	Retrieves unloaded (80-byte card images) records Maintains pointer to current input area. Maintains number of bytes remaining to be processed on a given card image. Checks each card image for proper sequence.	54
IEBISPL (Print)	IEBISPL	Produces printed copy of input data set. Provides for user exit and/or suppression of data conversion.	55
IEBISF	IEBISF	Receives control from initializing or processing module. Prints appropriate message and returns completion code to calling routine.	56

Figure 48. Module Directory, Summary, and Chart IDs for IEBISAM Program

UNLOADING AN INDEXED SEQUENTIAL DATA SET

If the unload function was specified, control is passed from the Initializing routine to the Unload routine (in module IEBISU) to create a physical sequential data set containing the information from the input (indexed sequential) data set. This information is put in 80-byte card images on either a magnetic tape volume or a direct access volume. Figure 49 illustrates the data flow and format during unloading operations.

Module IEBISU contains two control sections (CSECTs): IEBISU (Chart 51), which reads records in logical sequence from an indexed sequential data set; and IEBISSO (Chart 52), which reblocks the records and writes them into a physical sequential data set.

Obtaining Indexed Sequential Records

After module IEBISU is entered at CSECT IEBISU, the Unload routine opens the input data set and determines the format of the records. (If the relative key position is the high-order byte of the record (i.e., DKBRKP equals zero), the key field is treated separately when the records are put into the output data set.)

CSECT IEBISU uses CSECT IEBISSO as a subroutine; initially, IEBISU passes control to IEBISSO to open the output (unloaded) data set, then again to write the input (indexed sequential) DCB into the output data set. After the DCB has been written, IEBISU uses the GET (locate mode) macro instruction to obtain a record from the input data set, then passes control to CSECT IEBISSO.

Building the Output Data Set

CSECT IEBISSO performs the reblocking and writing of the input indexed sequential records into the output data set. The output data set is a physical sequential data set consisting of 80-byte logical records. The 80-byte records contain the key and

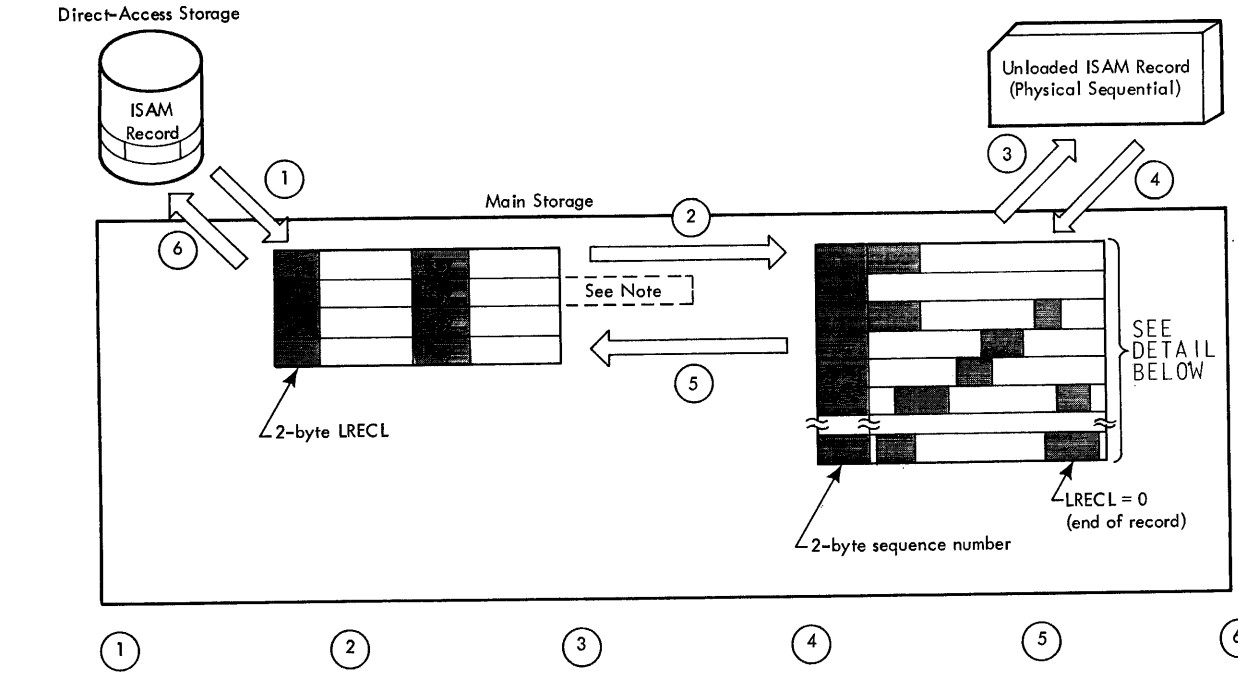
data fields of the indexed sequential data set, together with length indicators and sequence numbers (see Figure 49).

The first 154 bytes of DCB information for the indexed sequential data set are written in the first two physically sequential records (those with sequence numbers zero and one) of the output data set. The first 80-byte logical record contains the physical sequence number zero, followed by the length indicator 154. (The length indicator represents the number of bytes between one length indicator field and the succeeding length indicator field.) The first 76 bytes of the DCB for the input data set complete the first logical record. The second 80-byte logical record contains the sequence number one, followed by the next 78 bytes of the input data set's DCB. The information in the first 154 bytes of the input DCB includes the following fields: CPTCD, RECFM, LRECL, BLKSIZE, RKP, NTM, KEYLEN, and CYLOFL. (See Figure 47, and the section "Data Control Block--ISAM" in the publication IBM System/360 Operating System: System Control Blocks, Form C28-6628.) The remaining 80-byte logical records (beginning with sequence number two) contain the images of the records in the input data set. The last 80-byte logical record of the unloaded (physical sequential) data set contains from zero to two bytes of zeros following the last byte of input record data.

At the first entry to CSECT IEBISSO, the Unload routine opens the output DCB and checks the DSORG and BLKSIZE parameters: the DSORG parameter must be PS, and the BLKSIZE parameter must be multiple of 80. If the opening is successful and the parameters are valid, the routine issues a PUT (locate mode) macro instruction to write the DCB in the output data set. This information and control of the Unload routine are then returned to CSECT IEBISU.

On subsequent entries to CSECT IEBISSO, the output buffer is filled with indexed sequential records obtained by CSECT IEBISU. The routine stores the record length indicator first, then it stores the record key and data fields. When the routine finds the end of an input record, it returns control to CSECT IEBISU to obtain another record; when it has filled the input buffer, the routine issues a PUT (locate mode) macro instruction to write the contents of the buffer into the output data set. The physical sequence number for the output data set records is then updated.

If CSECT IEBISSO encounters an error condition (e.g., unsuccessful open, invalid DCB parameters, or an uncorrectable I/O error), it closes the output data set, sets the appropriate return code (see Chart 51), and returns control to CSECT IEBISU. IEBISU then sets both a message and a completion code, closes the input data set, and passes control to the Terminating routine.



1. Using successive GETs, an indexed sequential record is read. Each logical record is prefixed with a 2-byte LRECL (Logical Record Length)
2. The prefixed ISAM logical records are reblocked into 78-byte logical record images, each of which is prefixed with a 2-byte physical sequence number
3. The 80-byte unloaded records are written using successive PUTs
4. During a subsequent LOAD execution, unloaded records are read (using GET) in sequence until an ISAM record can be rebuilt
5. The sequence numbers are stripped and the record is rebuilt
6. The record is written using successive PUTs

Note: The current version of ISAM supports only fixed-length records.

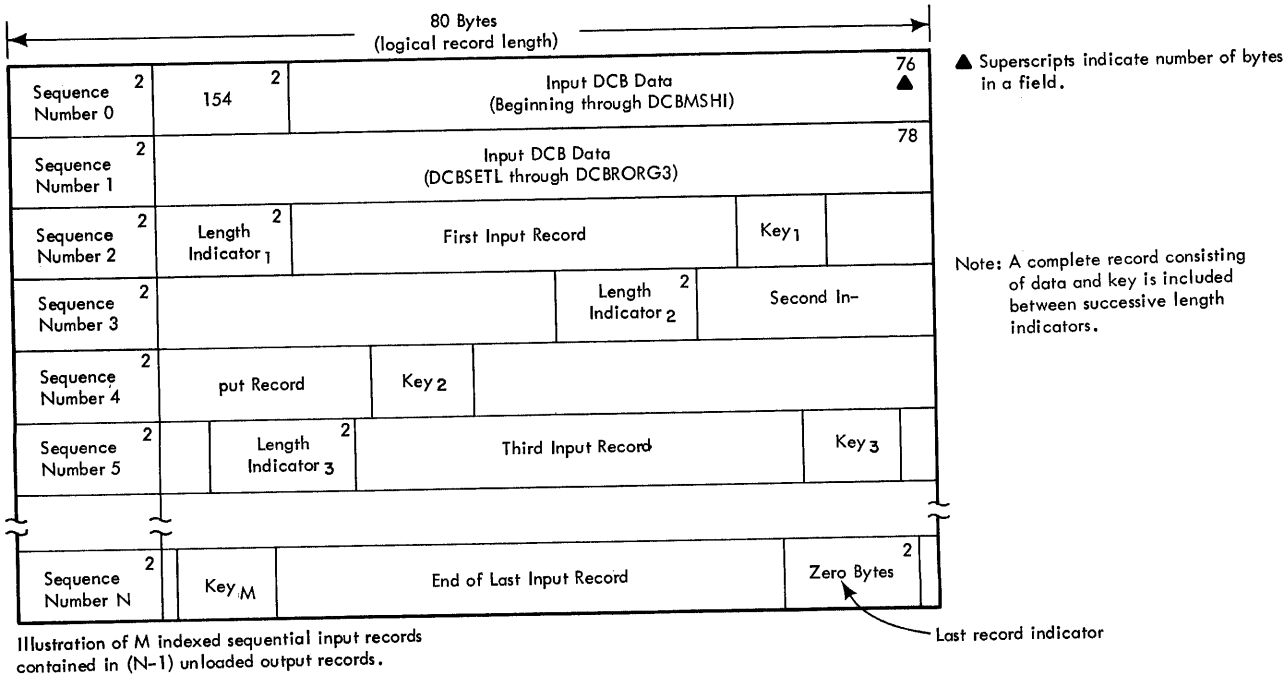


Figure 49. Unloading and Loading an Indexed Sequential Data Set

LOADING AN INDEXED SEQUENTIAL DATA SET

The Load routine is used to reconstruct an indexed sequential data set from an unloaded copy of the physical sequential data set. The output data set resulting from the loading function is placed on a direct access volume. If the original indexed sequential data set contained records in an overflow area, these records will appear sequentially arranged with the records from the original primary area when the unloaded data set is reloaded.

To perform the load function, the Initializing routine gives control to CSECT IEBISL of module IEBISL (see Chart 53). The Load routine performs its own initializing functions, then branches to CSECT IEHISSI of the same load module to get the length and address of an input record from the unloaded data set. If the return to CSECT IEBISL from CSECT IEHISSI indicates a return code other than zero, the appropriate message number and/or completion code are established, the output data set (if it had been opened as described later on) is closed, and control is given to the Terminating routine.

If CSECT IEHISSI returns the requested information and a return code of zero when it gives control back to CSECT IEBISL, CSECT IEBISL opens the output data set and checks the validity of the DCB fields. An inconsistency (or error) detected during either of the latter operations leads to procedures for closing the data set as previously described. Otherwise, if no error is detected, the PUT macro instruction is used to place the record information in the new indexed sequential (output) data set.

When all records from the unloaded (old) data set have been transferred to the new data set, the old data set is closed and control is given to the Terminating routine.

In reconstructing the new data set, the information in the first two logical records of the unloaded data set is used in establishing the DCB for the new data set. The last 78 bytes of each subsequent 80-byte logical record are used to build the records of the new data set.

CSECT IEBISSI (Chart 54) opens the input (unloaded) data set and checks for the validity of the DCB parameters for that data set. Should either the opening be unsuccessful or a DCB parameter be invalid, the data set is closed and return is made to CSECT IEBISL. Otherwise, CSECT IEBISSI proceeds to get information from the logical records of the unloaded data set and to transmit it to CSECT IEBISL so that it may be placed in the new indexed sequential data set. The GET and PUT macro instructions are used for these operations. The preceding procedures continue until either the end of the input data set is reached or a terminating error condition is reached. For both situations, the input data set is then closed, and control is returned to CSECT IEBISL.

PRINTING LOGICAL RECORDS OF AN INDEXED SEQUENTIAL DATA SET

In order to obtain a printed copy of an indexed sequential data set, a user specifies the keyword PRINTL in the PARM field of an EXEC statement. The queued indexed sequential access method (QISAM) is used to obtain the records from the input data set. The records are selected in logical sequence from both the prime and the overflow areas of the input data set. To write the records, the queued sequential access method (QSAM) uses a PUT macro instruction. Record conversion (to hexadecimal notation) and/or user exits before record printing may be specified as options.

After module IEBISAM gives control to the print module IEBISPL (Chart 55), both the input and the output data sets are opened, the success of the openings is determined, and the DCB parameters are checked for validity. If an error is encountered in any of the preceding operations, steps are taken to close the data sets and give control to the Terminating routine.

If the data sets have been opened successfully and the DCB parameters are valid, the Print routine proceeds to place a record in a buffer area prior to printing it. At this point, a user's routine may gain access to the record if the proper specification has been given on the EXEC statement. Upon return from the user's routine with a return code of either 0 or 4 (see the return code table on Chart 55), or if no user exit was taken, the data in the buffer is converted to hexadecimal notation unless the no-conversion option has been specified. The PUT macro instruction is then issued to print the record on a SYSOUT device. After all input data records have been printed, or if the routine encounters an unrecoverable error, the input and output data sets are closed and the Terminating routine is given control.

Note: A more complete interpretation of the codes returned to the print module IEBISPL by a user's exit routine is given below:

Code 0: The record currently in the buffer is to be printed, and processing of the input data set is to continue.

Code 4: The record currently in the buffer is to be printed, but processing of the input data set is to be terminated after the printing.

Code 8: The record currently in the buffer is not to be printed. Processing of the input data set is to continue.

Code 12: The record currently in the buffer is not to be printed, and processing of the input data set is to be terminated.

TERMINATING THE IEBISAM PROGRAM

Each of the other routines of the IEBISAM program may give control and a completion code to the Terminating routine (in module IEBISF). The basic function of the Terminating routine is to write an appropriate message on the SYSPRINT data set. This message indicates the result of the use of the IEBISAM program.

When module IEBISF (see Chart 56) gains control, it opens the output (SYSPRINT) data set. If the opening is unsuccessful, the approriate completion code (16) is set, the SYSPRINT data set is closed, and control is returned to the source from which the IEBISAM program was initially given control.

After a successful opening of the output data set, the PUT macro instruction is used to write the message concerning the program's result. If an error is encountered during writing, a completion code of 8 is set and returned to the caller of the IEBISAM program. (The completion codes shown on Chart 56 are those resulting from processing actvity by module IEBISF.) If no error is encountered during writing, the Terminating routine established a completion code based upon the results of the routine from which the Terminating routine received control. This code, and program control, are then given to the caller of the IEBISAM program.

Chart 48. IEBISAM - Overall Flow

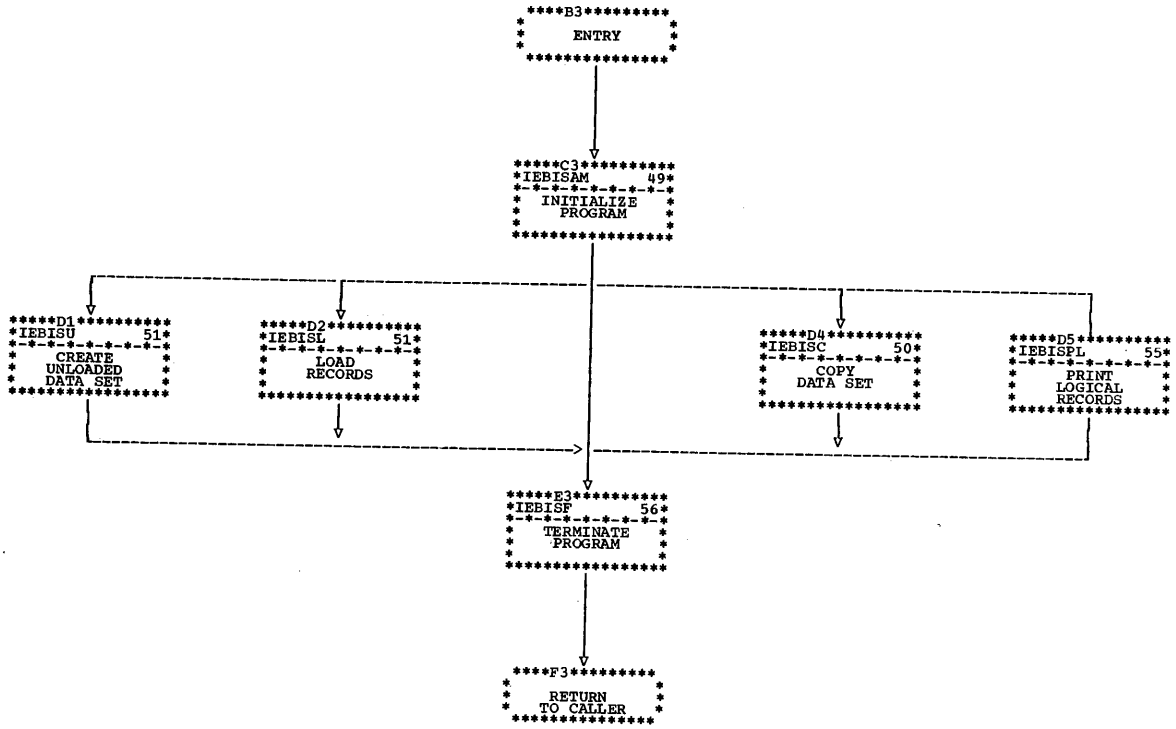


Chart 49. IEBISAM - Initialize IEBISAM Program

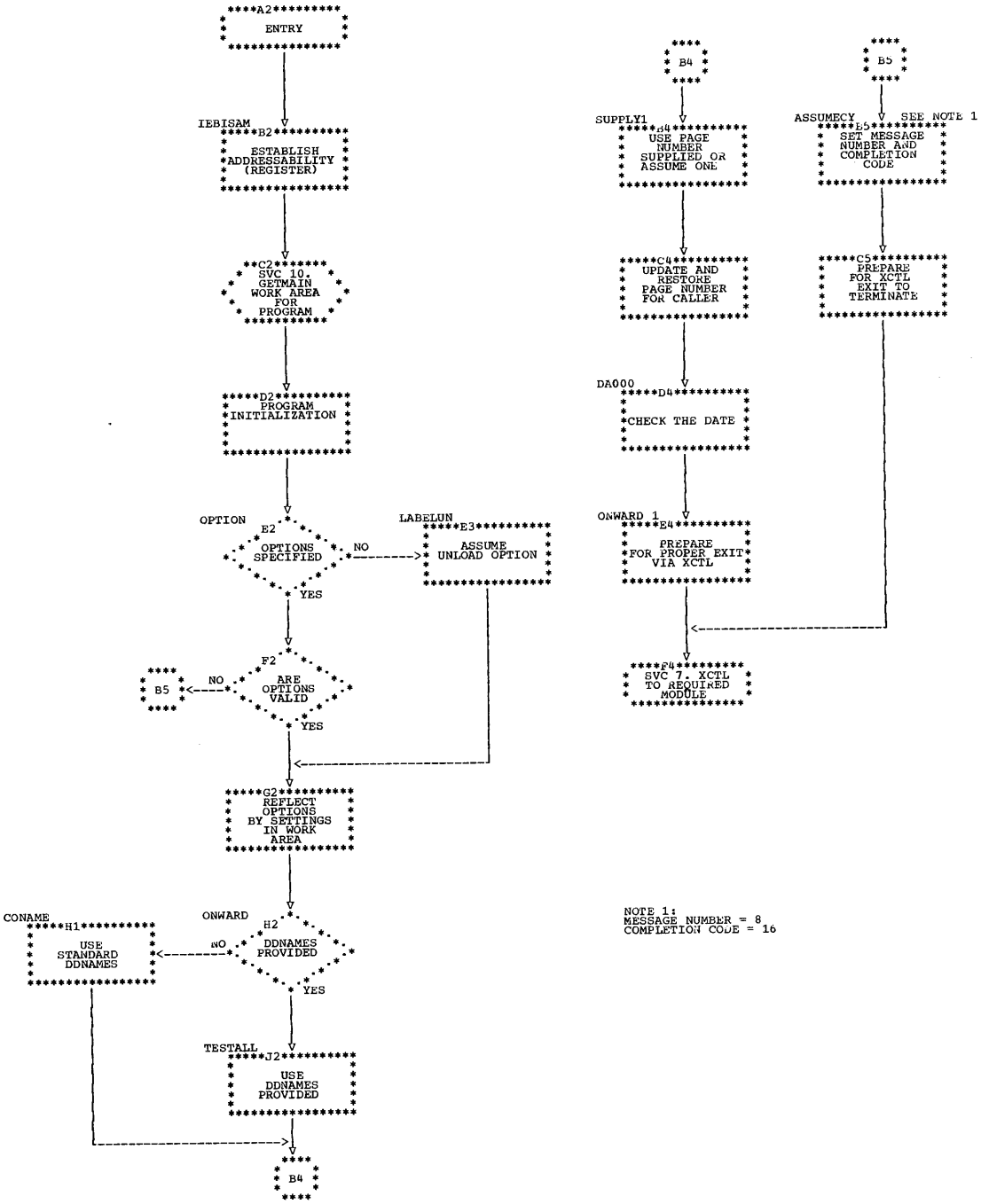


Chart 50. IEBISAM - Copy Indexed Sequential Records (IEBISC)

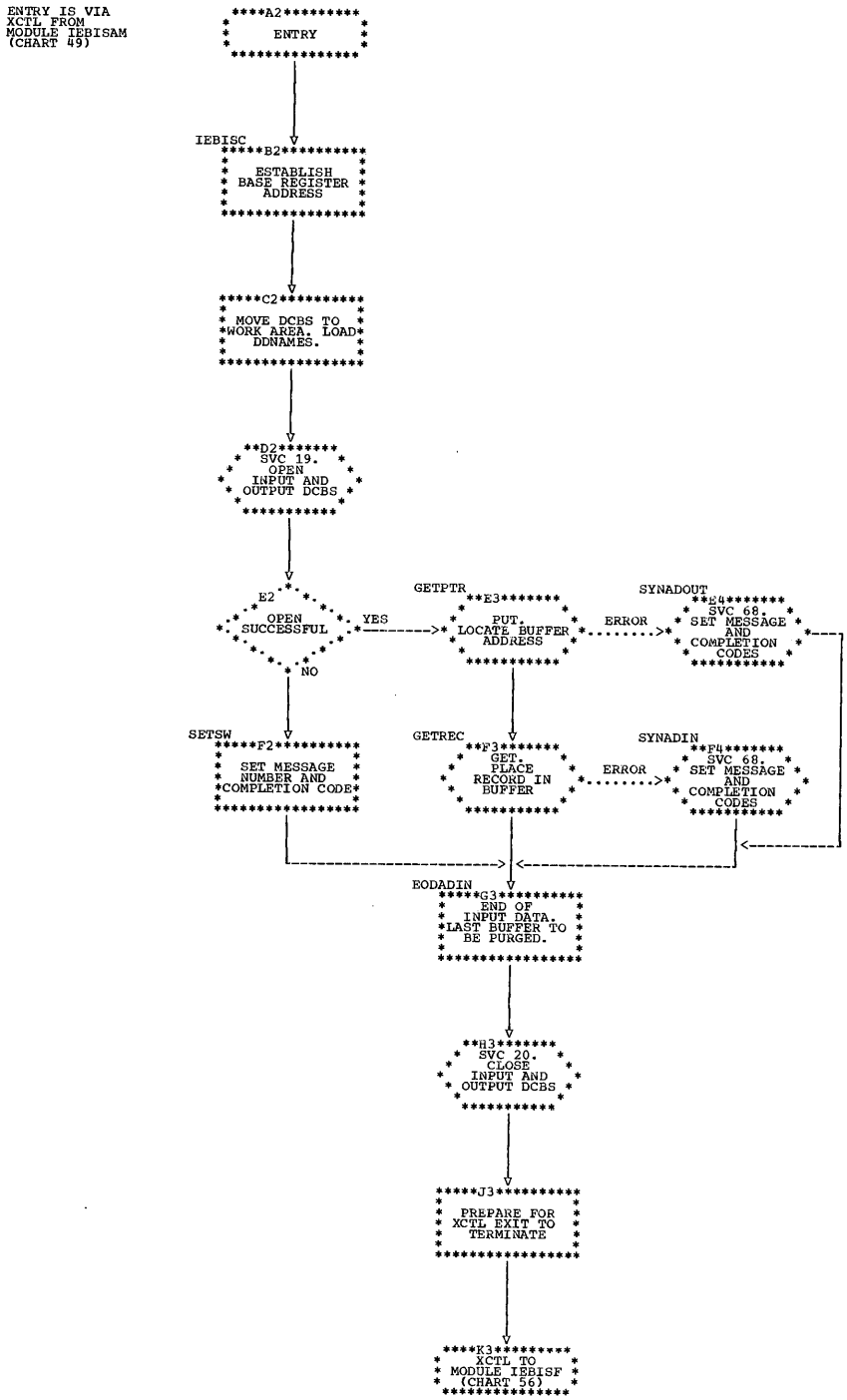
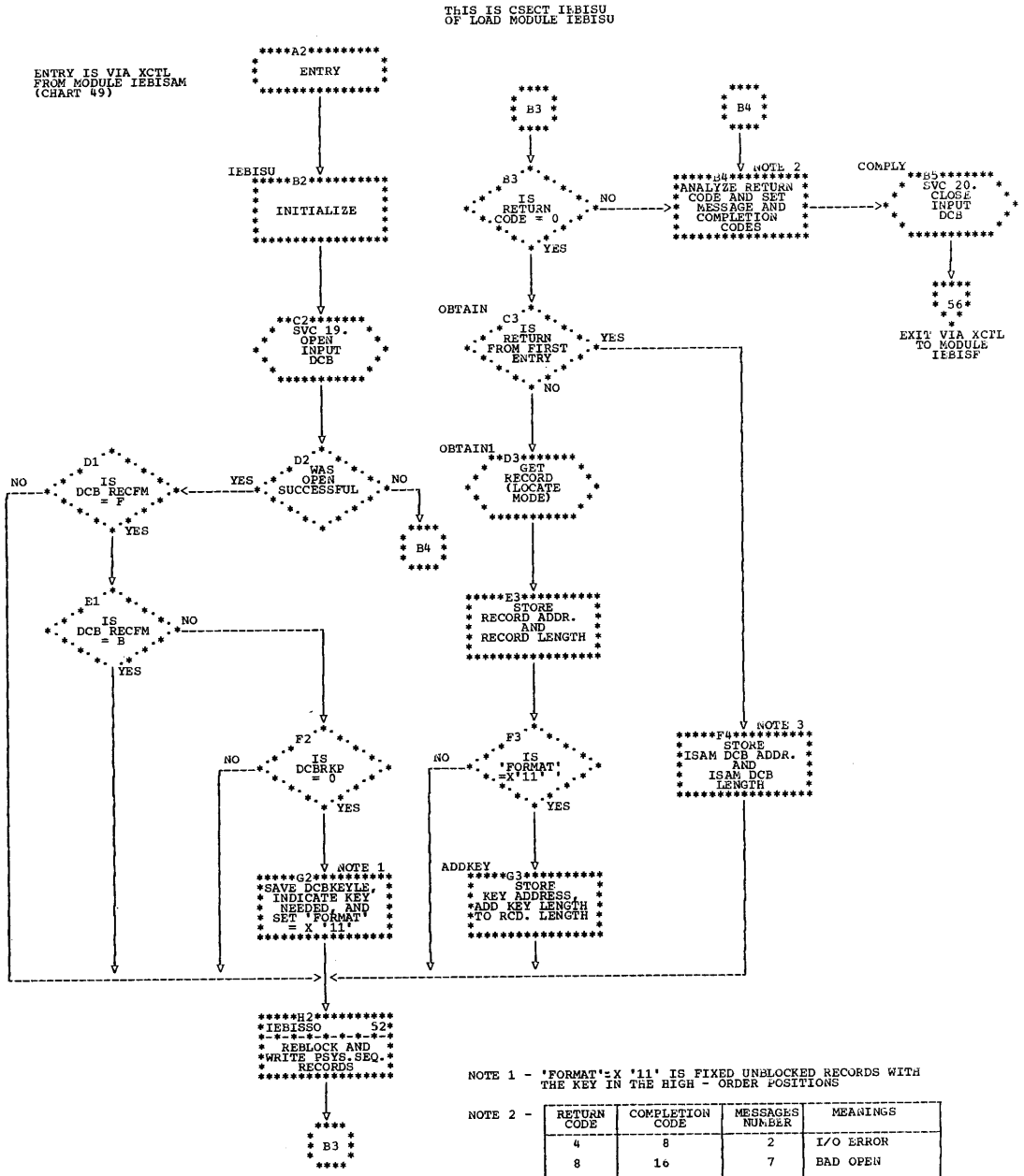


Chart 51. IEBISAM - Retrieve Indexed Sequential Records (IEBISU)



NOTE 1 - 'FORMAT'=X '11' IS FIXED UNBLOCKED RECORDS WITH THE KEY IN THE HIGH - ORDER POSITIONS

NOTE 2 -

RETURN CODE	COMPLETION CODE	MESSAGES NUMBER	MEANINGS
4	8	2	I/O ERROR
8	10	7	BAD OPEN
12	8	1	BAD DCB PARAMS

NOTE 3 - THE FIRST TWO (PHYS.SEQ.) RECORDS WRITTEN CONTAIN THE DCB FOR THE INPUT (ISAM) DATA SET

Chart 52. IEBISAM - Unload Physical Sequential Records (IEBISSO)

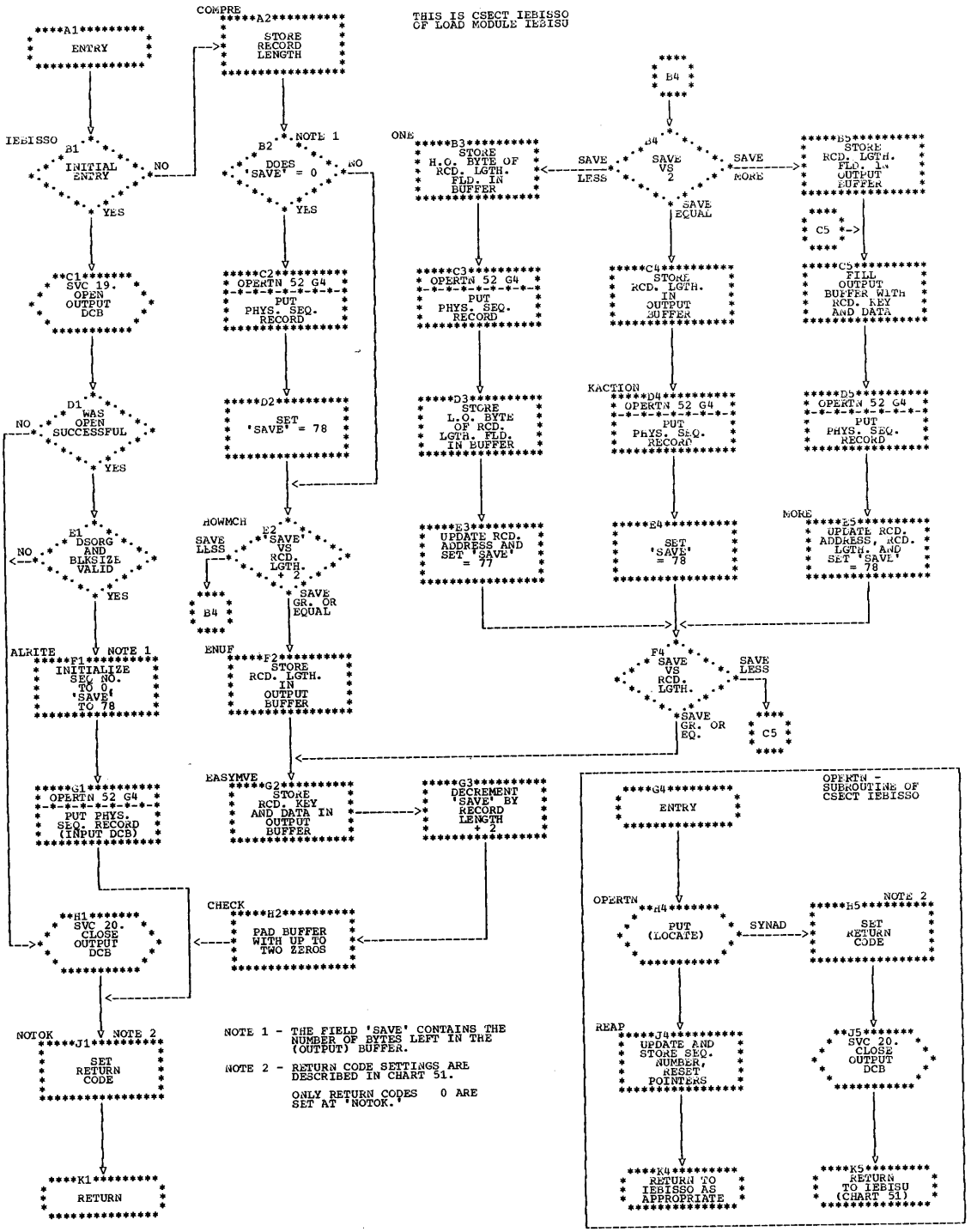
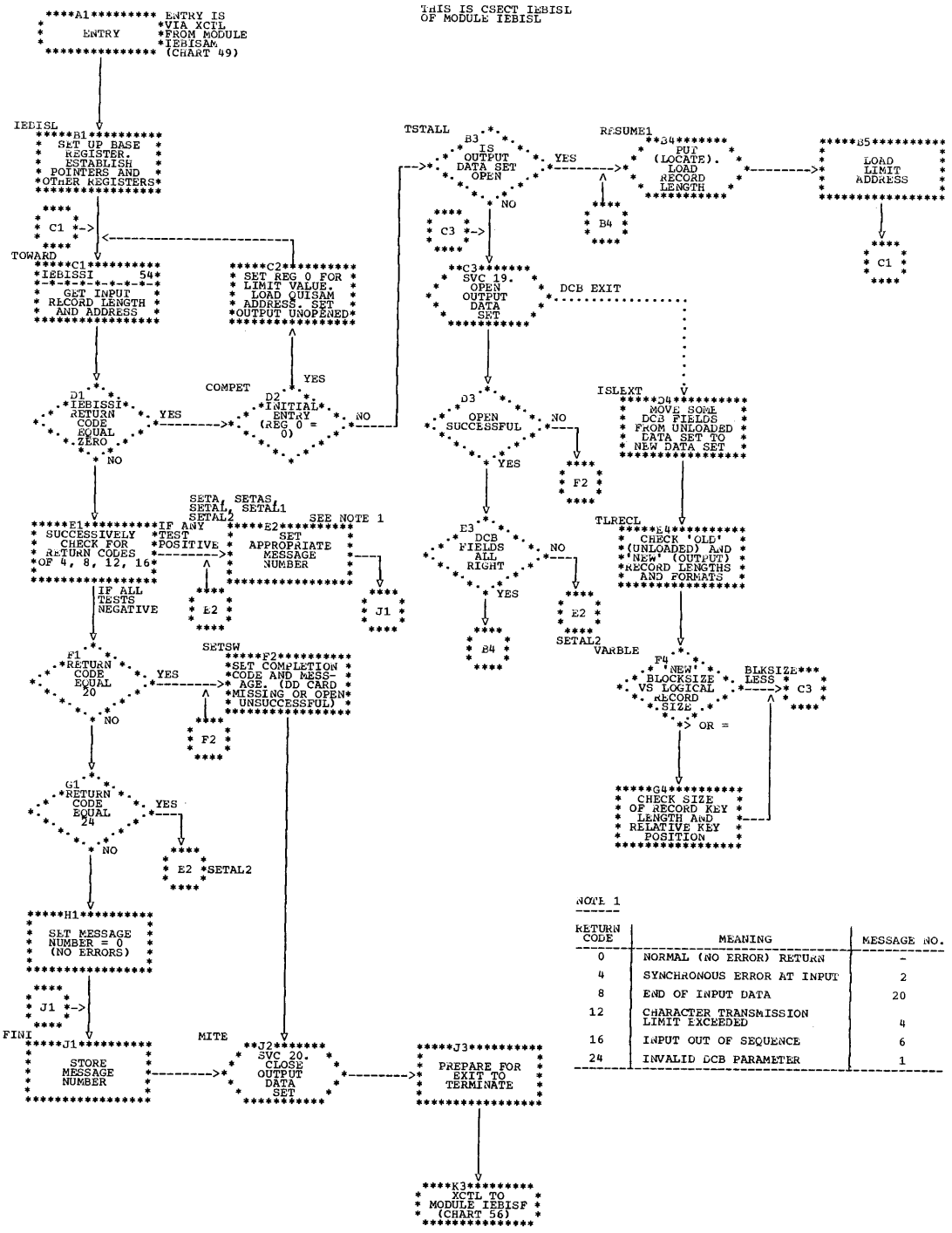


Chart 53. IEBISAM - Reconstruct Indexed Sequential Records (IEBISL)



NOTE 1

RETURN CODE	MEANING	MESSAGE NO.
0	NORMAL (NO ERROR) RETURN	-
4	SYNCHRONOUS ERROR AT INPUT	2
8	END OF INPUT DATA	20
12	CHARACTER TRANSMISSION LIMIT EXCEEDED	4
16	INPUT OUT OF SEQUENCE	6
24	INVALID DCB PARAMETER	1

Chart 54. IEBISAM - Retrieve Physical Sequential Records (IEBISSI)

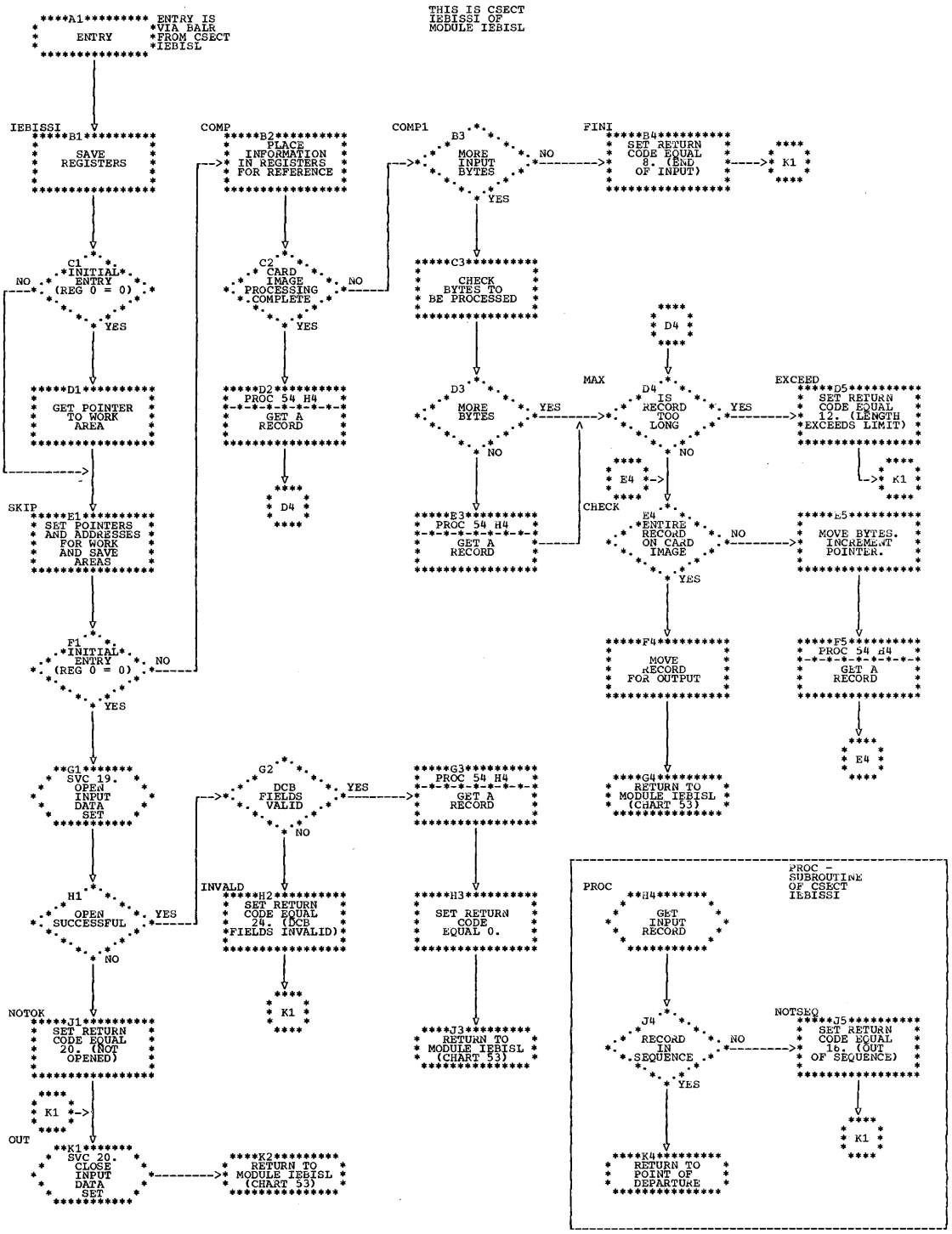
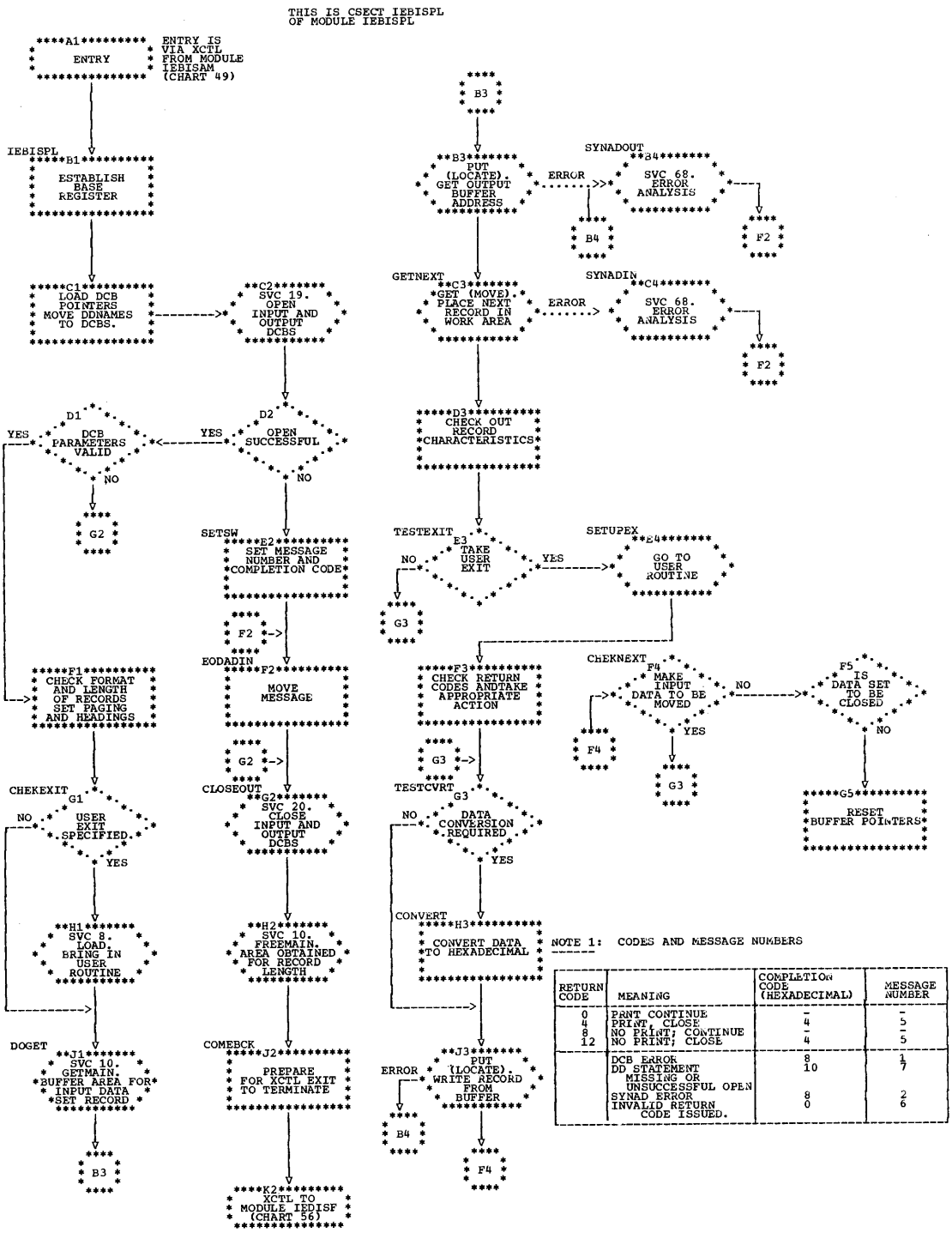


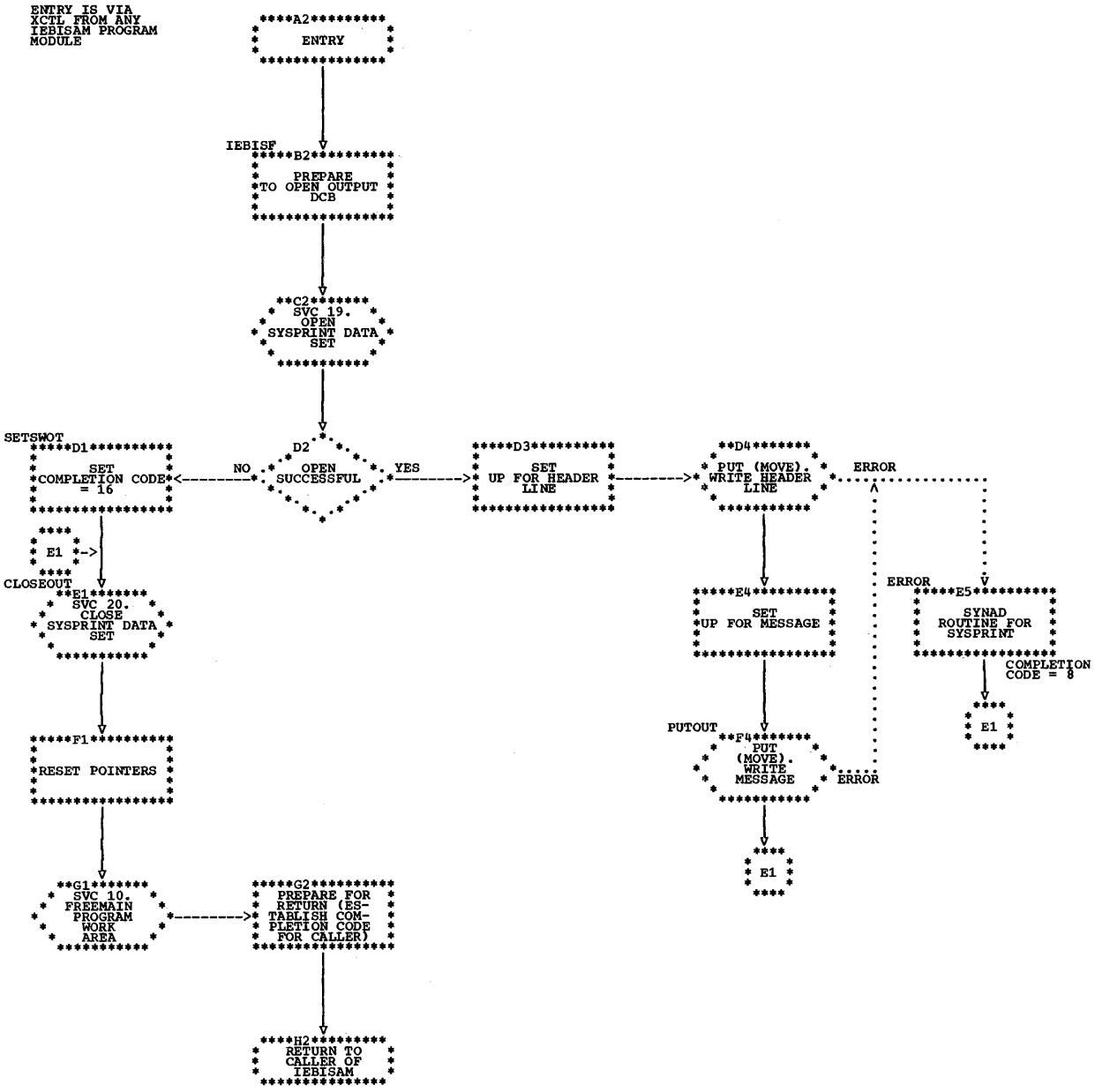
Chart 55. IEBISAM - Print logical Records (IEBISPL)



NOTE 1: CODES AND MESSAGE NUMBERS

RETURN CODE	MEANING	COMPLETION CODE (HEXADECIMAL)	MESSAGE NUMBER
0	PRNT CONTINUE	-	-
4	PRINT, CLOSE	4	5
8	NO PRINT; CONTINUE	-	-
12	NO PRINT; CLOSE	4	5
	DCB ERROR	8	1
	DD STATEMENT MISSING OR UNSUCCESSFUL OPEN	10	7
	SYNAD ERROR	8	2
	INVALID RETURN CODE ISSUED.	0	6

Chart 56. IEBISAM - Terminate IEBISAM Program (IEBISF)



Updating Symbolic Libraries (IEBUPDAT)

The IEBUPDAT program modifies a symbolic library. The program can:

- Add, copy, and replace members.
- Add, delete, replace, and renumber the records within an existing member.
- Assign sequence numbers to the records of a new member.

The input to the IEBUPDAT program consists of two data sets: the old master data set (SYSUT1) and the current transaction data set (SYSIN). The old master is a partitioned data set that contains all of the library members; the current transaction is a sequential data set that contains all of the transactions that are to be applied to the library members. The logical record length for both data sets is 80 bytes, blocked or unblocked.

The output of the IEBUPDAT program consists of two data sets: the new master (SYSUT2) and the log (SYSPRINT). The new master is a partitioned data set that contains the updated version of the symbolic library; the log is a sequential data set that contains the latest changes to the old master or, optionally the currently updated version of the old master. The logical record length on the new master is 80 bytes, blocked or unblocked; on the log it is 120 bytes, unblocked. The blocking factors of the old and new masters may be different.

The program obtains main storage for buffers by means of the getmain routine, which is called once for each buffer; the amount of storage requested is the same as the block size specified by the utility keyword parameter BLKSIZE. If the amount of storage requested is not available, the program terminates.

The current transaction is the controlling data set. Only those members of the old master for which there are current transaction entries will be processed. Old master members without current transaction entries will not appear in the new master.

PROGRAM STRUCTURE

The IEBUPDAT program (Figure 50) can be logically divided into three parts: initialization, member processor, and within-member processor.

Initialization

Initialization sets switches, assigns work areas, and opens the input and output data sets. It consists of four functions: AHEAD, ANALPRAM, OPEN1, and OPENINPT.

AHEAD
: initializes switches, work areas, and DCBs so that they can be reused.

ANALPARM
: analyzes input specifications and user header and trailer label exit routine name specifications. Errors cause termination of the program with a message.

OPEN1
: opens output data sets.

OPENINPT
: opens one or two input data sets according to optional parameter input specifications.

Member Processor

The member processor updates whole members at a time. It reads the current transaction data set and does preliminary processing of all headers: ADD, REPL, REPRO, and CHNGE. Further processing of the CHNGE header is done by the Within Member Processor. The ADD, REPL, and REPRO headers and their associated current transaction records are processed by the Member Processor.

A new-master is created by the Member-Level Processor for ADDs, REPLs, and REPRO headers. A REPRO header will cause the new-master to be written from the old-master instead of from the current transaction data set as is done for ADD and REPL headers. Processing of the current transaction header includes sequence checking of member names, determination of proper directory entry (or lack of), stowing of ALIASes, sequencing of ADDs and REPLs (through presence of NUMBR), and detection of invalid transactions (i.e., transactions that logically are out of sequence or are incorrectly prepared). The member processor consists of eight routines: READCT, SOURCECK, MAINBODY, SOURCERT, OMREADRT, LOGROUTE, NUMBRRTE, and STOWNAME.

READCT
: reads the current transaction data set and deblocks if necessary. It then checks for two illegal headers in a row (ADD, REPL, or CHNGE).

SOURCECK
: determines the type of transaction.

MAINBODY
: processes headers. It checks the member name of the current transaction header stream for proper sequence and sets up the STOWAREA area with the

directory image. If the header is an ADD, MAINBODY ensures that there is no directory entry on the old master; conversely, if the header is a REPL, REPRO, or CHNGE, a directory entry must already be on the old master.

SOURCERT
processes all source line transactions in a member following an ADD or REPL header.

OMREADRT
processes the source line transactions in a member following a REPRO header.

LOGROUTE
writes headers, ALIAS and NUMBR transactions, and error messages, on SYSPRINT.

NUMBRRTE
processes NUMBR transaction following either a REPL or ADD header.

STOWNAME
stores a member name in the output directory.

Within Member Processor

The Within Member Processor updates the records within a member. It inserts, deletes, reproduces, replaces and/or resequences source code images. Control is given to the Within Member Processor when the Member Level Processor detects a CHNGE transaction and verifies the existence of the named member on the old master data set. The Within Member Processor retains control, processing a member of the old master as specified by the record of the current transaction data set until another header record or the ENDUP record is read. Control is then returned to the Member Level Processor.

The within member processor consists of four routines: RRFINDOM, RRSOURCE, RRDELET, and RRNUMBER.

RRFINDCM
reads the first record of the old master member being changed; then reads and checks the current transaction for the type of transaction. Control is passed to the appropriate transaction routine.

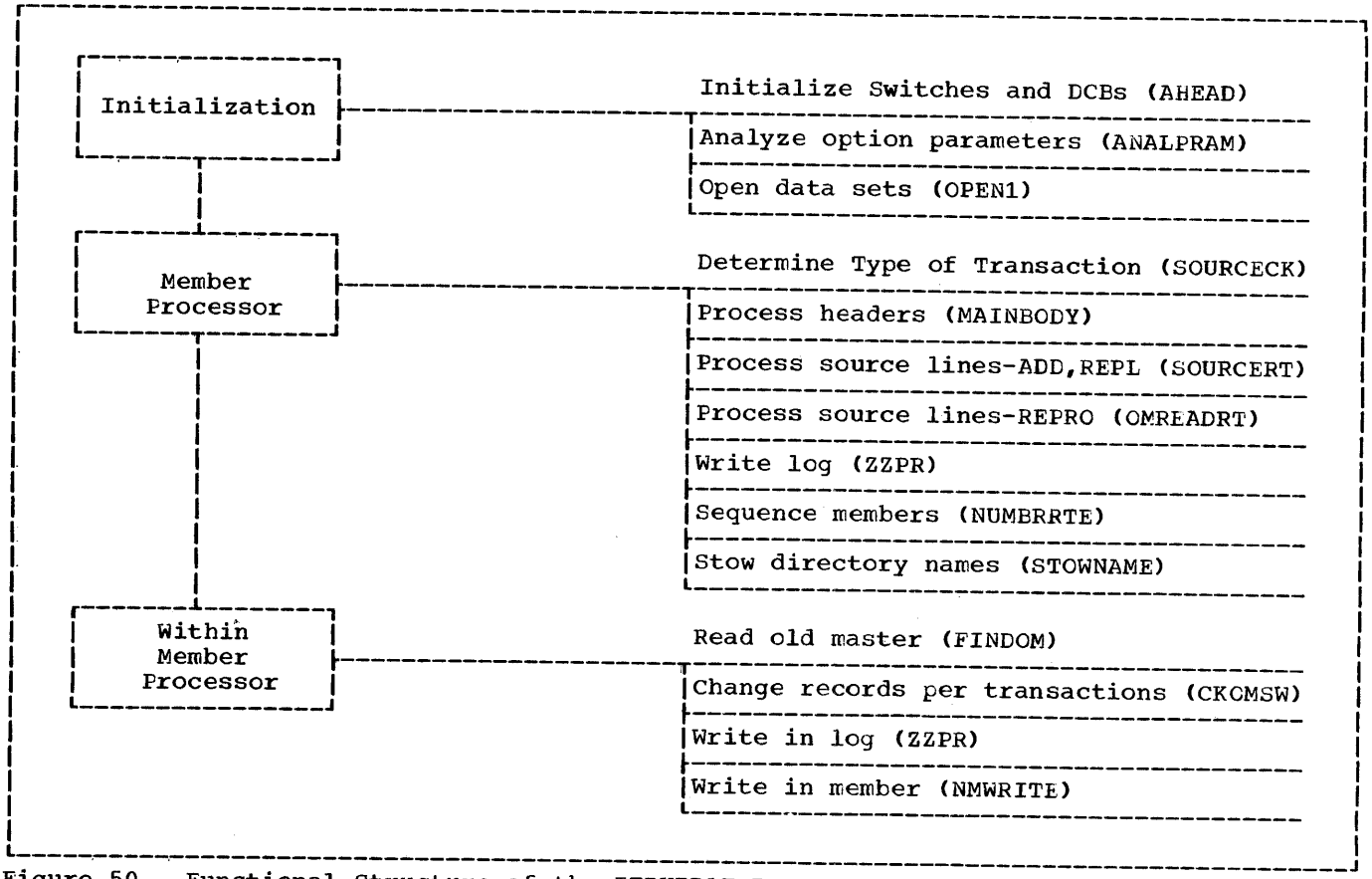


Figure 50. Functional Structure of the IEBUPDAT Program

RRSOURCE
compares the sequence numbers of the old master record and the source transaction record to determine whether the source record is an insertion or a replacement.

RRDELET
deletes old master records whose sequence numbers are within the range of numbers on the DELET transaction.

RRNUMBR
provides the sequence numbers for old master records and inserted source transaction records that follow a NUMBR transaction.

PROGRAM FLOW

Chart 57 shows the flow of control through the IEBUPDAT program. After the program is entered, it sets switches, assigns work areas, and checks DCBs for reusability. The output data sets SYSPRINT (log) and SYSUT2 (new master) are opened. The log header is written, using the optionally specified initial page number, and messages indicating error conditions found during ddname or initial page number interrogation are issued. Option parameters supplied by the user via the EXEC statement are analyzed.

Next, the current transaction data set (SYSIN) and the old master data set (SYSUT1) are opened. (the DCB exit is taken to determine the block size so that a buffer area can be dynamically obtained for the SYSIN data set. A user header label exit may be taken at this point to process user header labels.

The READCT routine is the starting point for the Member Processor part of the program, and is executed each time processing is completed on a current transaction and a new current transaction is needed. READCT passes control to the READCTA subroutine which reads and deblocks a record from SYSIN. The record can be one of the following: a header record, a source record, a NUMBR record, or an ALIAS record. A header record is processed by the HEADERCK routine; a source record is processed by the SOURCERT routine; a NUMBR record is processed by the NUMBRRTE routine; and an ALIAS record is processed by the STOWNAME routine.

The HEADERCK routine determines whether the header is valid and then sets appropriate switches depending on the type of header. If the header is not valid, an error message is logged and control is passed to the READCT routine. Valid headers are processed by MAINBODY.

The SOURCERT routine processes all source line transactions that are in a member whose header is either an ADD or REPL. A check is made to see if the source is in its proper place by checking the ALREPOSW switch which is turned on when either an ALIAS or REPRO is encountered. If the ALREPOSW switch is on, the source is out of sequence and a message is logged via the LOGROUTE. Control is then passed to the READCT routine. If the ALREPOSW is off, the CTINAREA area which contains the source image, is moved to the OMINAREA area. Then, if the NSW switch is off (no NUMBR preceding the source), the current transaction is written on the new master. The full list switch, FLLISTSW is checked and if it is on, the record is logged and control is returned to READCT.

The NUMBRRTE routine processes the NUMBR transaction which may have followed either a REPL or an ADD header. A check is made of the ADDREPSW switch, which will be on if the previous transaction was an ADD or REPL. If ADDREPSW is off, an error message is logged and control is passed to the READCT routine. If ADDREPSW is on, the NUMBR transaction is checked for its proper sequence within the stream of current transaction records referencing a member. Sequence numbers are converted and placed in the proper work areas. The NUMBR transaction is logged after which control is passed to the READCT routine.

The STOWNAME routine causes the previous member name or alias to be stored in the directory with the system status indicator (SSI) bytes (if any) via the STOWREPL subroutine. If the current transaction in CTINAREA is an alias, the alias, TTR, and user information are moved to STOWAREA. The alias is logged via the LOGROUTE routine and control is passed to the READCT routine which reads the next transaction. If the current transaction is not an alias, control is passed to the HEADERCK routine.

By reaching MAINBODY, it has been determined that the header is in proper sequence with a member. The member name, however, is compared with the previous member name to determine if the member is in sequence. If the member is out of sequence, an error message is logged and control is passed to the READCT routine. If the member is in sequence, the directories from the old master and the new master are compared. There should be entries in both directories for REPL, REPRO, or CHNGE headers but no entry in the old master for an ADD header. In the event of an error, an error message is logged, the entire member is rejected, and control returns to the READCT routine. If there are no errors, the header is logged.

If the header is a REPRO, the old master is read into OMINAREA; the record is logged if the full list switch (FLLISTSW) is on; and the record is then written on the new master. If the header is an ADD or REPL, control is passed to the READCT routine. If the header is a CHNGE, control is passed to the within member processor RRFINDOM.

Beginning at RRFINDOM, the within member processor handles the transactions following a CHNGE header; i.e., source, DELET, and NUMBER. The member being changed is located on the old master data set and the first record is read. The current transaction file is also read and checked for the type of transaction.

If the transaction is a source, the sequence number of the new master record and of the current transaction record are compared. When the old master is low, it is rewritten onto the new master and the next record on the old master is read and compared. When the old master is equal to the source transaction, the current transaction is written on the new master.

If the transaction is a DELET, the sequence number of the old master record is compared to the 'start' sequence number of the DELET transaction. When the old master is low, it is written onto the new master and the next record on the old master is read and compared. When the old master is equal to or greater than the 'start' sequence number, the old master records are read and deleted until a record is read whose sequence number is higher than the 'end' sequence number in the DELET transaction.

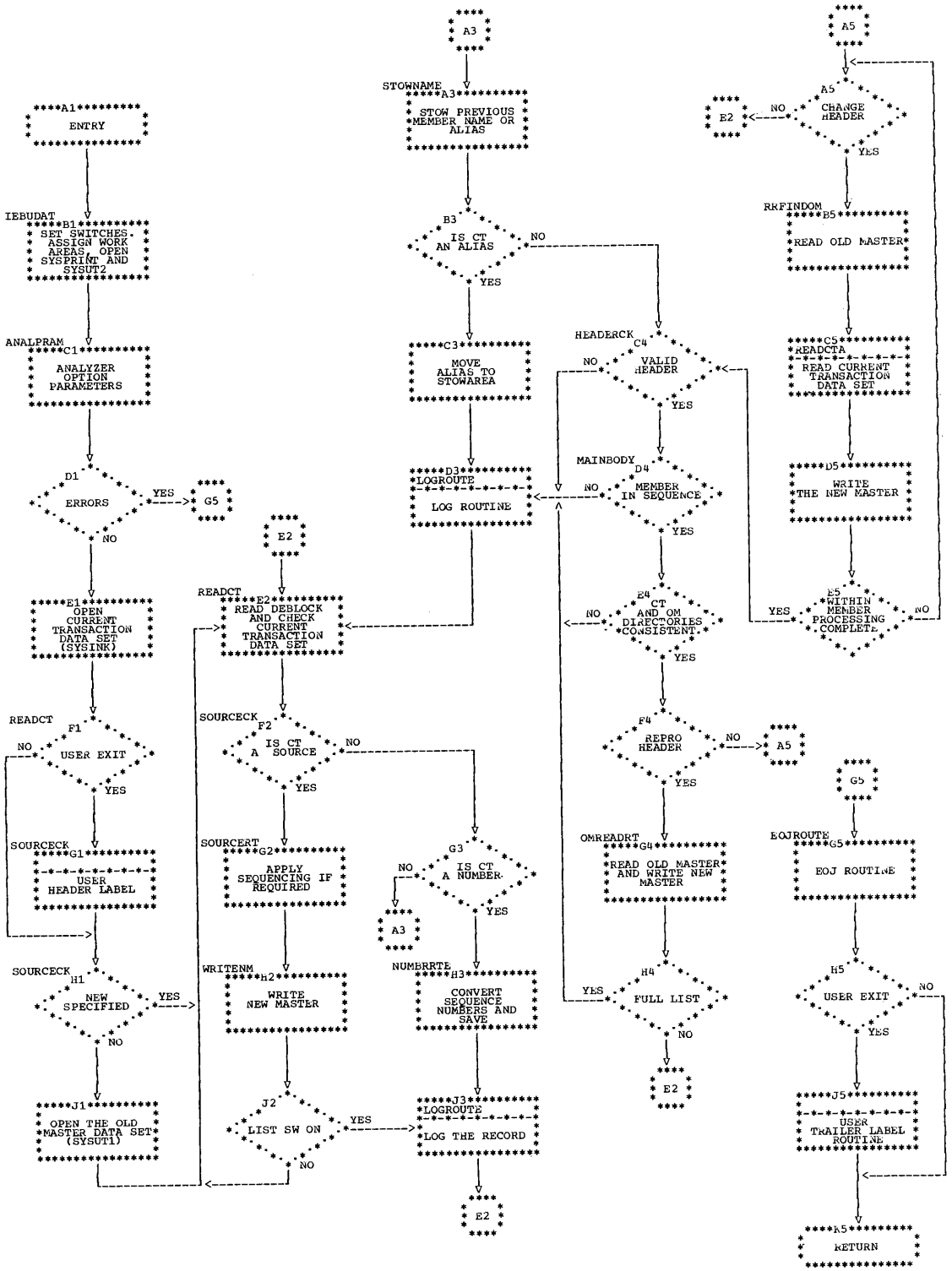
If the transaction is a NUMBR, the old master is read and resequenced according to the range of sequence numbers in the NUMBR transaction. The current transaction is also read and any DELET or source transaction is processed as described above. Source transaction may also be numbered sequentially.

As the current transactions are read and processed, each current transaction detail record is logged as is the record or records it referenced. If a complete log is requested, all records placed in the new master data set are logged. Any errors detected during processing are also logged.

Utilizing the EODAD exits, the end of member on the old master and the end of data on the current transaction data set are determined. Processing continues until the new master member is completed. All switches are reset and work areas are cleared before returning to the member level processor at STOWNAME.

After the last member is processed, as indicated by a /ENDUP or EOD exit on SYSIN, the old master, the new master, and SYSIN data sets are closed. A user trailer label exit, if one was specified by the user via the EXEC card may be taken at this point to process user trailer labels. When this is done, a final message is logged indicating the highest concode obtained in the program. The SYSPRINT data set is closed and control is returned to the invoker.

Chart 57. IEBUPDAT - Updating Symbolic Libraries



Creating a Modified Input Stream (IEBEDIT)

The IEBEDIT program creates a sequential data set containing Job Control Language (JCL) statements and system input data by extracting sets of statements representing jobs or job steps from a master file. The input to the program is in two data sets:

- SYSIN, which contains control statements that allow the user to control the editing of the master file of JCL statements and data.
- SYSUT1, which contains the master file of JCL statements and data.

The output of the program is in two data sets:

- SYSUT2, which is the primary output data set. It is composed of 80-character logical records containing the JCL statements and data records extracted from the master file.
- SYSPRINT, which contains a listing of the control statements, and (optionally) a listing of the contents of the SYSUT2 data set.

The IEBEDIT program is executed as a job step; the EXEC statement used to call it specifies the program IEBEDIT.

PROGRAM STRUCTURE

The IEBEDIT program is contained in one load module whose entry point name is IEBEDIT. The module contains three major program sections as well as a number of subroutines. The three major sections of the program are:

- The Initializing routine, which obtains main storage for tables and work areas, initializes them, and opens the program's data sets.
- The Main routine, which passes control among the subroutines to analyze control statements, to inspect master file records, to determine which records should be written out, and to write output records.
- The Post Processing routine, which stores condition codes, frees main storage, closes the program's data sets, and returns control to the supervisor.

The Initializing Routine

The entry point for the IEBEDIT program is the Initializing routine. When it is entered, the routine obtains main storage for an active save area and a work area, and opens the SYSPRINT, SYSUT1, SYSUT2, and SYSIN data sets.

The Initializing routine checks the block size specification of each data set except SYSPRINT to insure that it is a multiple of 80 characters. If the SYSUT2 blocksize specification is not a multiple of 80 characters, it is changed to match the SYSUT1 specification, and a message is written to SYSPRINT. If the SYSUT1 data set is not a multiple of 80 characters, a message is written to the SYSPRINT data set and the step is terminated.

If any data set cannot be opened, the Initializing routine passes control (via a branch instruction) to the Post Processing routine. Otherwise, it uses the GET macro instruction (locate mode) to obtain the first SYSUT1 record, and branches to the Main routine.

The Main Routine

The Main routine (Charts 58 and 59) passes control among subroutines that analyze control statements from the SYSIN data set and master file records from the SYSUT1 data set. Based on the specifications in the control statements, the Main routine determines which records are to be extracted from the master file, and uses the Update subroutine to write those records to the SYSUT2 and (optionally) to the SYSPRINT data sets.

When the Main routine is entered (via a branch from the Initializing routine), the first record from the SYSUT1 data set is in main storage. The Main routine uses the Scan subroutine to obtain a record from the SYSIN data set, and to analyze the record.

If there are no control statements in the SYSIN data set, the Scan subroutine encounters an end-of-data condition, indicating that a total copy of the master file is to be performed. Control is passed to the Update subroutine to write the record to the output data sets, then back to the Main routine to get the next master file record. When the master file has been completely copied, the Main routine passes control to the Post Processing routine.

If the Scan subroutine obtains a control statement, a selective copy is performed, based on the specifications in the control statement. The Main routine passes control to the Startjob subroutine, which gets master file records until it finds the proper JOB statement:

- If the parameter START=jobname was used in the control statement, the Startjob subroutine searches the master file for a JOB statement with the specified name.

- If no job name was specified, the Startjob subroutine searches the master file for the next JOB statement.

When the proper JOB statement has been found, the Startjob subroutine passes control to the Update subroutine, which writes the statement to the SYSUT2 and, optionally, to the SYSPRINT data set. When control is returned to it, the Startjob subroutine reads the next record and uses the Cardtype subroutine to determine whether the record is a JOBLIB DD statement.

If the record is a JOBLIB DD statement, the Update subroutine writes it to the output data sets. The Startjob subroutine then obtains another master file record from the SYSUT1 data set and returns control to the Main routine.

On the return from the Cardtype subroutine, the Main routine analyzes the switches set by the Cardtype subroutine and performs the processing indicated by the record type and control statement specifications.

If the record is an EXEC statement, its disposition depends on the use of the TYPE and STEPNAME parameters in the control statement.

If TYPE=POSITION, and no stepname was specified, the Main routine passes control to the Update subroutine, and the record is written to the output data sets. If a stepname was specified, and the corresponding EXEC statement is found, the Main routine passes control to the Update subroutine, and the record is written to the output data sets.

If TYPE=INCLUDE or EXCLUDE, the Main routine must determine whether the current record represents a step within an inclusive set, and if not, whether it represents a step whose name was specified singly. The routine does this with the aid of two tables (the inclusive stepnames table and the single stepnames table) and the inclusive/exclusive switches.

Each entry in the inclusive stepnames table contains the names of the first and last steps in a set as specified in the STEPNAME parameter; each entry in the single stepnames table contains the name of a step specified singly. The include/exclude switches indicate whether inclusive or exclusive processing is taking place.

The decisions made in the program, and the resultant processing, are shown in Figure 51. The upper section of the table shows the conditions that may exist; the lower section shows the action that is taken as a result of each set of conditions. The action "Write" means that the Main routine uses the Update routine to write the record containing the EXEC statement, and the remaining records representing that step, to the output data sets. The action, "No Write" means that the Main routine searches for the end of the current step, but does not write the records to the output data sets.

The end of the current step is indicated by the presence of a JCB statement, another EXEC statement, or an end-of-data condition. If a DD DATA statement is encountered, a switch is set; subsequent records, although they may appear to be JCL statements, are treated as data in the input stream. When a delimiter statement is encountered, the DD DATA switch is set off; and if the other records in the step were written out, so is the delimiter statement.

When a JOB statement is encountered, or when an end-of-data condition exists in the

Include/Exclude Switch is On	X	X	X	X						
Include/Exclude Switch is Off					X	X	X	X		
Match 1st Name in Inclusive Stepnames Table							X	X		
Match 2nd Name in Inclusive Stepnames Table	X	X								
No Match in Inclusive Stepnames Table			X	X	X	X			X	X
Match in Single Stepnames Table									X	X
No Match in Single Stepnames Table		X	X	X	X	X	X	X		
TYPE=INCLUDE	X		X		X		X		X	
TYPE=EXCLUDE		X		X		X		X		X
Write	X		X			X	X		X	
No Write		X		X	X			X		X
Set Include/Exclude Switch On							X	X		
Set Include/Exclude Switch Off	X	X								

Figure 51. EXEC Statement Include/Exclude Processing

SYSUT1 data set, the Main routine scans the list of step names constructed from the control statement. If any of the names in the list were not found, a message containing the step name is written to the SYSPRINT data set for each missing step. If a JOB statement was encountered, the Main routine then passes control to the Scan subroutine to analyze the next control statement; if there was an end-of-data condition, the Main routine passes control to the Post Processing routine.

The Post Processing Routine

The Post Processing routine is entered when no more processing is to be performed; at end-of-data in the SYSUT1 data set, when all SYSIN statements have been processed, or when an unrecoverable I/O error occurs. When it is entered, it determines whether an end-of-data condition exists for the SYSIN data set; if not, it uses the Scan subroutine to process the remaining control statements.

When all records in the SYSIN data set have been processed, the Post Processing routine uses the Update subroutine to write a terminal message (including the condition code) to the SYSPRINT data set. It then closes the program's data sets, frees the main storage that had been obtained, and returns control to the supervisor.

IEBEDIT Subroutines

The IEBEDIT program contains four major subroutines: Scan, Startjob, Cardtype, and Update. Linkage to each subroutine is via a BAL instruction; return is via a BR instruction.

The Scan Subroutine

The Scan subroutine is entered to obtain and analyze a complete control statement: the initial record and any continuation records. When the Main routine is first entered, the Scan subroutine determines whether a total copy is required; if not, and when a job has been processed, it determines the processing required for the next job; and when an end-of-data condition occurs on the SYSUT1 data set, it is entered to scan the remaining SYSIN records.

When the Scan subroutine is entered, it attempts to obtain a record from the SYSIN data set. If it obtains a record, it scans the record, converting the control statement parameters to switch settings that can be tested by the Main routine, and when it has processed the entire statement, it returns control to its caller. If it encounters an end-of-data condition, and no statements have previously been processed, the routine sets a switch indicating that a total copy is to be performed, and returns control to its caller. If it encounters an end-of-data condition, and statements have previously been processed, it passes control to the Post Processing routine.

When the routine is entered, it uses a search routine to set pointers to the fields in the statement, then scans the field. The Scan routine has five phases: Initialization, Name/Operator Handling, Operand Handling, Operand Value Handling, and Scan Post Processing.

The Initialization phase clears switches and resets pointers; the search routine finds the Name and Operator fields, and control passes to the Name/Operator Handling phase.

In the Name/Operator Handling phase of the Scan subroutine, the name field of the statement is checked for validity (it must be 8 characters long or less). Then, the contents of the Operator field is used as a search argument in a search of the Operation Code Table (see Figure 52). When a match is found, the Turn-On Box of the table is used to set the appropriate switches in the IEBEDIT work area, and pointers to the operator and to the appropriate Parameter Table (see Figure 53) are placed in the work area.

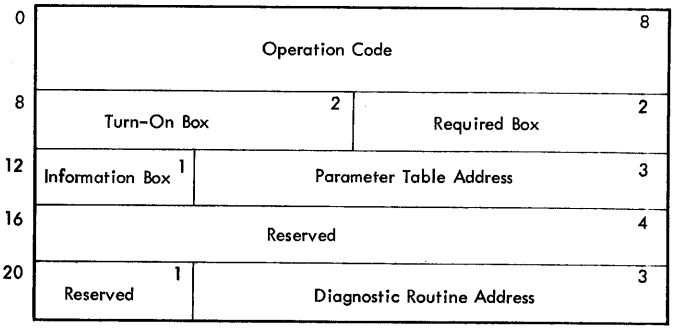


Figure 52. Scan Routine Operation Code Table Entry

Operation Code: This field contains the Operation Code, left justified, and padded with blanks.

Turn-On Box: This field contains the displacement (byte 1) in the IEBEDIT Work Area and the bit pattern (byte 2) to be set at that displacement.

Required Box: This field contains a displacement (byte 1) in the IEBEDIT Work Area, and a bit pattern (byte 2) to be found at that displacement. This bit pattern is required for processing of this statement.

Information Box: If bit 0 of this field is set to 1, this entry is the last entry in the table.

Parameter Table Address: This field contains the address of the Parameter Table that corresponds to this operation.

Diagnostic Routine Address: This field contains the address of a routine used to perform additional processing on the statement.

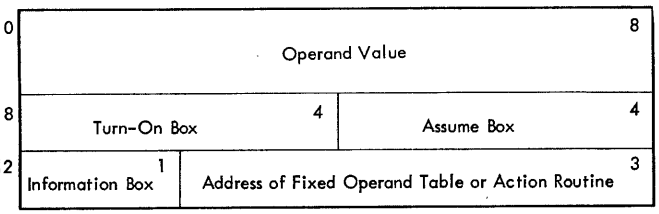


Figure 53. Scan Routine Parameter Table Entry

Operand Value: This field contains the value of the operand, left justified, and padded with blanks.

Turn-On Box: Byte 1 of this field contains a displacement in the IEBEDIT Work Area; byte 2 contains a bit pattern to be set at that displacement as a result of encountering this parameter.

Assume Box: Byte 1 of this field contains a displacement in the IEBEDIT Work Area; byte 2 contains a bit pattern to be set at that displacement if this parameter is omitted.

Address of Fixed Operand Table or Action Routine: If the operand is a fixed operand, this field contains the address of the appropriate Fixed Operand Table entry, if the operand is a variable operand, this field contains the address of the routine that is to process the operand.

Information Box: The bits in this field have the indicated meanings when set to 1:

Bit	Meaning
0	Last entry in table
1	Fixed operand
2	Variable operand
3	Reserved
4	Allow subparameters
5	Keyword-only operand
6-7	Reserved

Each operand in turn is used as a search argument, in the Operand Handling phase, to scan the Parameter Table. When a match is found, the Turn-On Box of the Parameter Table is used to set the appropriate switches in the IEBEDIT work area, and a pointer to the Parameter Table entry is placed in the work area. If the operand is a keyword-only operand, and there are additional operand fields, the routine processes the next field. If there are no additional operands, the routine passes control to the Scan Post Processing phase.

If there are parameters associated with the keyword, the routine passes control to the Operand Value Handling phase. In this phase, the Scan subroutine inspects the Parameter Table entry to determine whether the parameter has a fixed value, or whether the value may vary. If the parameter is a variable value parameter, the Action Routine Address field of the Parameter Table entry contains the address of the routine that is to process the parameter, and a branch is executed to give that routine control. If the parameter is a fixed value parameter, the routine uses the value specified as a search argument in a search of the Fixed Operand Table (see Figure 54). When a match is found, the Turn-On Box field of the table is used to set the appropriate switches in the work area.

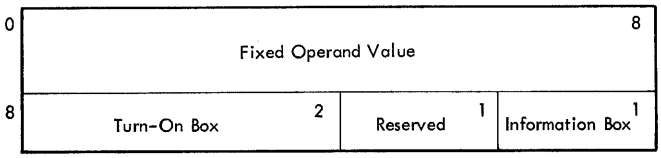


Figure 54. Scan Routine Fixed Operand Table Entry

Fixed Operand Value: This field contains the value of the operand, left justified, and padded with blanks.

Turn-On Box: Byte 1 of this field contains a displacement in the IEBEDIT Work Area; byte 2 contains the bit pattern to be set at that displacement when this operand is encountered.

Information Box: If bit 0 of this field is set on, it indicates that this entry is the last entry in the table.

When the parameters associated with a keyword have been processed, control is passed to the Operand Handling phase to process the next operand; if there are no more operands to process, control is passed to the Scan Post Processing phase.

When a complete statement has been processed, the Scan Post Processing phase scans the Parameter Table for the current operator, then sets the assumed (default) value switches for any parameters not supplied. The current Operation Code Table entry is then inspected to determine whether any diagnostic routine has been supp-

lied. If so, the diagnostic routine is given control, and when its processing is complete, the Scan routine returns control to its caller.

The Startjob Subroutine

The Startjob subroutine is entered from the Main routine with the first record of a master file statement in the buffer. It uses the Cardtype subroutine to determine the statement type, and it uses the Update subroutine to write a JOB statement and a JOBLIB DD statement to the output data sets.

If the first statement encountered by the Startjob subroutine is not a JOB statement, the routine gets records from the master file until it finds a JOB statement. The Startjob subroutine then determines whether the START=jobname parameter was used, and if not, it uses the Update subroutine to write the statement (including its continuations) to the output data sets.

If START=jobname was specified, the routine compares the specified job name to the name in the JOB statement. If they are not equal, the routine searches the master file until the proper JCB statement is found. In either case, the JOB statement having the specified name is written to the output data sets, and the Startjob subroutine reads the next master file record.

Once a JCB statement has been written out, the Startjob routine looks for a JOBLIB DD statement. If it encounters one, the routine uses the Update subroutine to write the statement to the output data sets; if the next statement is not a JOBLIB DD statement, the Startjob subroutine returns control to its caller.

The Cardtype Subroutine

The Cardtype subroutine classifies 80-character records by type. It stores a code for each type except system input data records, and if the record is a JOB or EXEC statement, it stores the statement name. When it has analyzed a record, it returns control to its caller.

The routine first examines the first two positions of the record. The characters // indicate that the record is a JCL statement, and the routine performs further analysis. The characters /* indicate that the record is a delimiter statement; the routine determines whether the statement is continued by checking for a nonblank character in position 72, then returns to its caller.

If the statement is a JCL statement, the routine classifies it as one of the following types:

- JOBLIB DD Statement: A statement is a JOBLIB DD statement if the name field contains JOBLIB and the operation field contains DD.
- JCB Statement: The statement is a JOB statement if the operation field contains JOB.
- EXEC Statement: The statement is an EXEC statement if the operation field contains EXEC.
- DD Statement: The statement is an DD statement if the operation field contains DD.
- DD DATA Statement: A statement is a DD DATA statement if it is a DD statement, and the first operand field contains DATA.
- Continued Statement: A statement is a continued statement if it is a JCL statement or a delimiter (/*) statement, and if it has a nonblank character in position 72.

The Update Subroutine

The Update subroutine is a control routine for the output functions of the IEBEDIT program. It contains the Put routine, which writes records to the SYSUT2 data set, and the Print routine, which writes records to the SYSPRINT data set. There are two entry points to the Update subroutine:

- UPDATE is the entry point used to write records to the SYSUT1 and, optionally, to the SYSPRINT data set.
- PRINT is the entry point used to write records to the SYSPRINT data set.

When it is entered at the UPDATE entry point, the routine inspects the first three positions of the record in the buffer. If it finds the characters period, period, asterisk (..*), it substitutes the characters /*; in either case, it branches to the Put routine.

The Put routine contains the PUT macro instruction, which causes the record to be written to the SYSUT2 data set. When the PUT macro instruction has been executed, the routine determines whether NOPRINT was specified, and if so, it returns control to the caller. If NOPRINT was not specified, the routine branches to the PRINT entry point of the routine.

When it is entered at the PRINT entry point, the routine is given the address of a record or a message code. It issues the PUT macro instruction to write the record or message to the SYSPRINT data set, then returns control to the caller.

Chart 58. IEBEDIT Main Routine (Part 1 of 2)

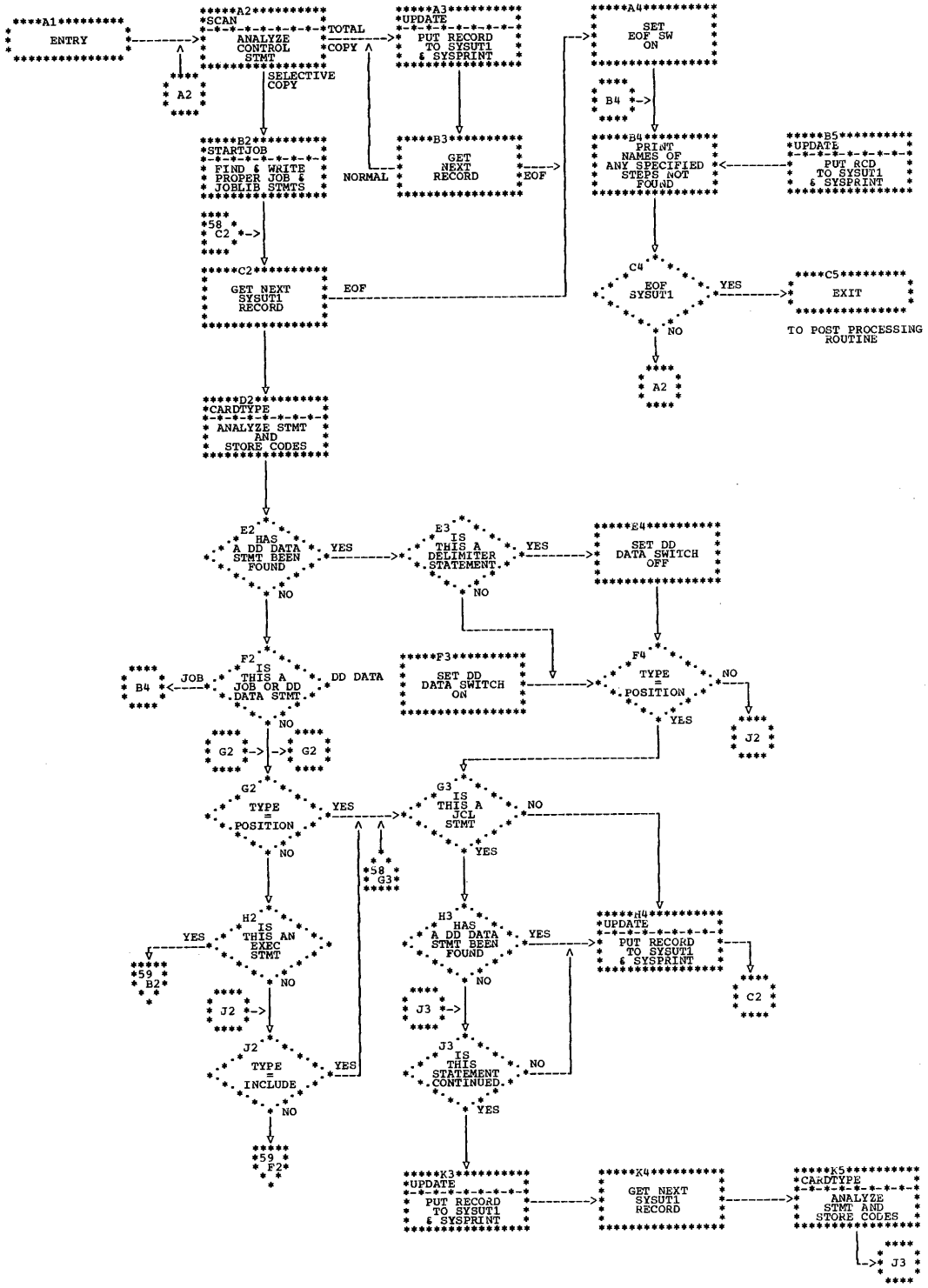
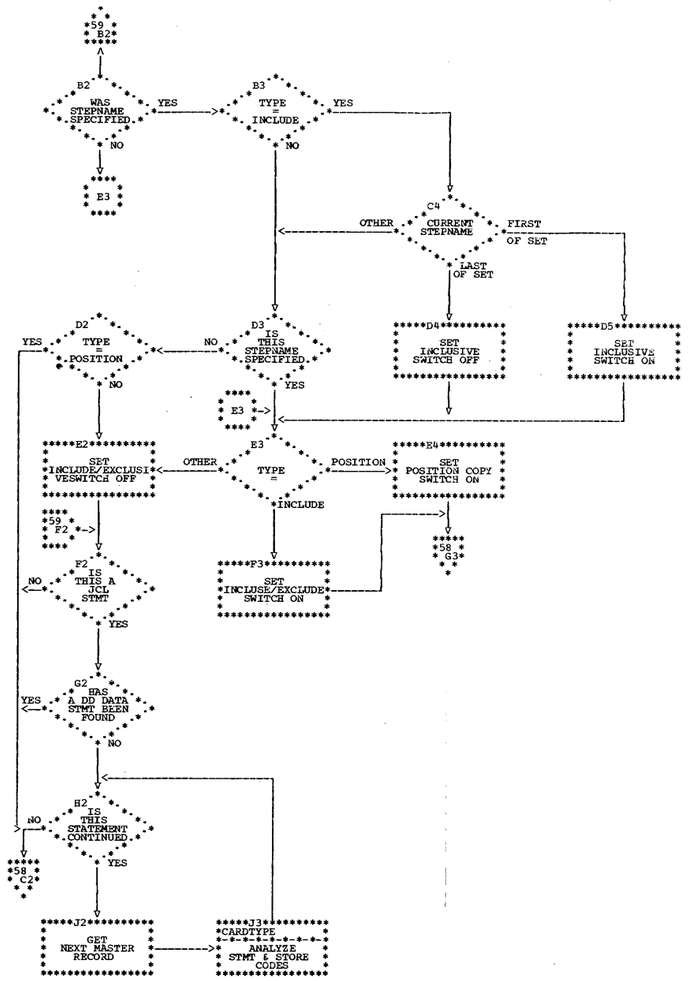
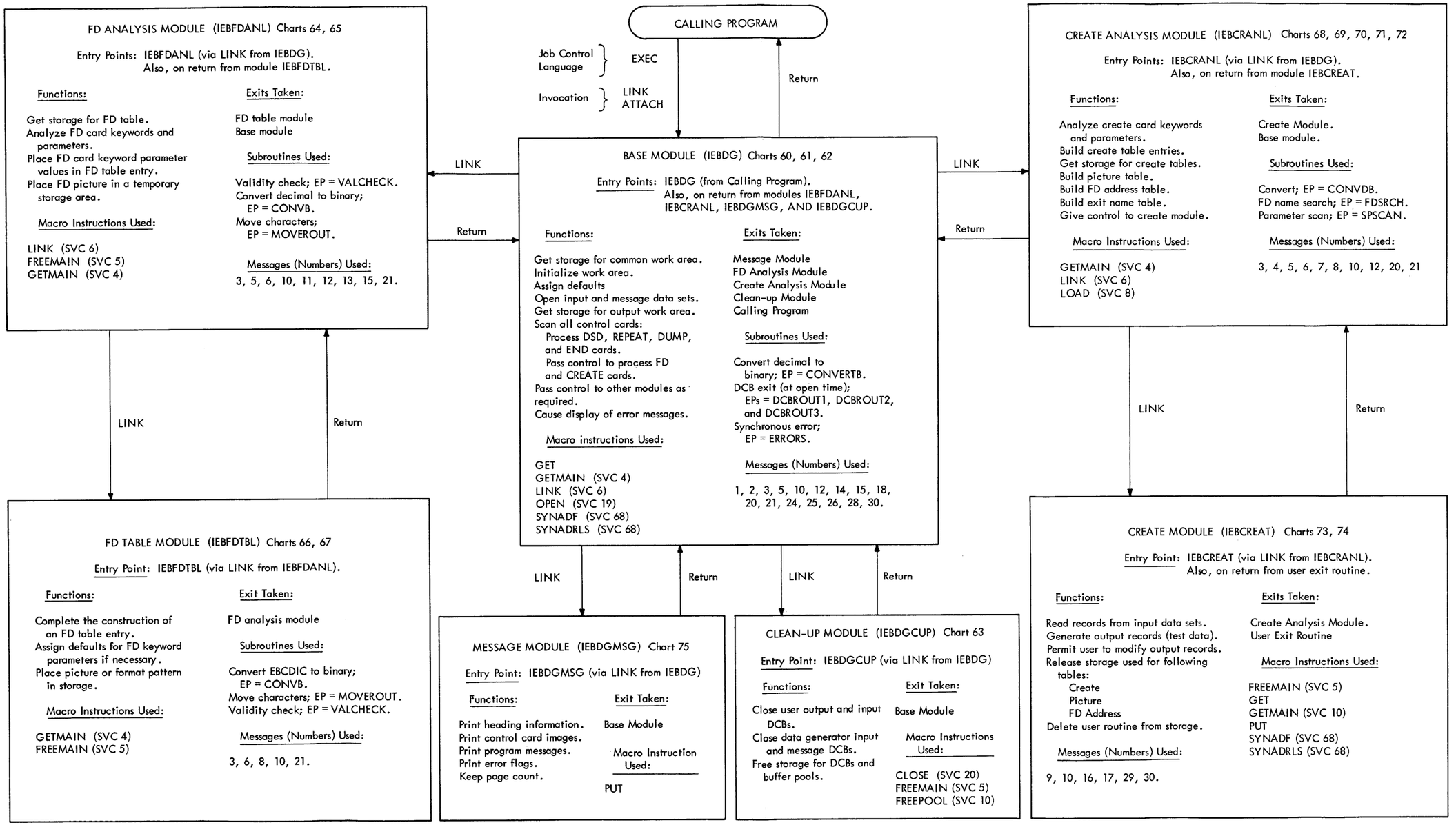


Chart 59. IEBEDIT Main routine (Part 2 of 2)





Note: EP = Entry Point

• **Figure 55. Information Summary and Overall Flow of Data Generator Program**

The Data Generator (IEBDG) Program

The data generator program (IEBDG) provides test data that can be used in program debugging procedures. The program will construct multiple data sets within a job that uses either the physical sequential or the partitioned access method. The records within these output data sets may consist of fields that are defined by any one of seven IBM character formats, each of which may be modified by any one of eight types of action. Alternatively, a user may elect to provide his own output pattern in the form of a 'picture' instead of an IBM format. If desired, a user may also inspect and/or modify the output records before the records are written in the test data set.

The IEBDG program acts as a problem program, which may be executed as a job step by use of the job control language, or which may be invoked by a calling program using either the LINK or the ATTACH macro instruction. Specification of either the program name (IEBDG) or the procedure name on the EXEC statement causes control to be given to the data generator program. In the case of invocation of the IEBDG program, the entry point (EP) parameter in the macro instruction operand specifies the program's symbolic name. Job control statements or parameter list information, and the IEBDG utility control statements, maintain control of the program and describe or specify the functions to be performed. They also describe or define the input and output data sets to be used. Depending on the specifications of the user, the records of the input data set may be either blocked or unblocked.

In the case of output records, the fields within a record may be repeated as desired, and the output records may be a part of a logical block, which may also be repeated. If an existing data set is used as the input data to the program, the fields within the individual records of the data set may be retained, modified, or replaced as desired. Also, the IEBDG program may generate output records that can be imbedded within the records of an existing (input) data set. The contents of the output (test data) records are defined by the utility program control statements.

Program Functions

The functions of the IEBDG program are performed by seven modules, which reside in the link library, SYS1.LINKLIB. At any given time, at least two modules (the control module and one or more other modules) will reside in the region assigned to the program. (If the region has enough space, all seven of the modules may be resident at the same time in the region.) The program contains a control module, a clean-up module, a message module, two analysis modules, a table-building module, and an output record generating module. Control passes within the modules of the program by means of the LINK macro instruction. If an exit to a user routine is specified, a branch-and-link procedure is used. Figure 55 indicates how control is passed between the modules of the IEBDG program.

The control (or base) module receives initial program control from the calling program and returns control to the calling program at the completion of the IEBDG program. This module scans the utility control cards for the function (e.g., FD card analysis, REPEAT card analysis) to be performed and passes control to the appropriate module that performs the function.

The clean-up module (IEBDGCUP) receives control from the base module to close the input and output data sets and to release the storage areas that were used by the program. This module returns control to the base module.

The message module (IEBDGMSG) has the prime function of putting out the images of the control cards, and of putting out the messages required as a result of program operation. It receives control from, and passes control to, the base module.

The message module places information about the operation of the data generator on the system output (SYSOUT) device. This information includes processed control cards, heading and paging information, and normal completion messages. Error messages caused by abnormal conditions encountered by the data generator program also appear on the SYSOUT device. Incorrect control card parameters cause messages that will be printed immediately below the printout of the control card. Messages begin at print position one, and the printout of control cards starts at print position ten.

Both the create analysis (IEBCRANL) and the FD analysis (IEBFDANL) modules analyze the parameters found on one of the two field- or record-defining control cards. Using the parameters found, these modules construct tables for use by subsequent modules that may require the information in the tables. In the case of the FD analysis module, control is given to the FD table building module, IEBFDTBL, to complete the construction of the table.

The output record generating module, or create module, IEBCREAT, controls the generation of the test (output) data records for the user. This module also passes

control to a user exit routine if output record modification is to be performed.

Control Card Scanning

Whenever any control card scanning is to be done, all modules within the IEBDG program employ the same general scanning techniques. The information to be scanned is placed in an input work area to which a register points. Information within this work area is scanned one byte at a time as the scan method looks for a non-blank character in a given column. If a nonblank character is encountered in column one of a card image, a control card name has been found. This name is of no significance to the program, and it may be up to 8 bytes in length; but it must be followed by a blank column. The card type (DSD, FD, END, etc.) is then determined. If the type is not valid, the program is terminated.

Following the card type and blank column, the finding of a nonblank indicates the presence of a keyword. As the scan encounters a keyword, an attempt is made to match the keyword with valid keywords in the program. If a match is made, a branch is made to the appropriate routine to process the keyword. If no match is made, or if incorrect parameters are associated with a given valid keyword, an error is indicated and a message is printed. A comma or a blank signifies the end of a parameter. A continuation card is to follow when either a nonblank character in column 72 of a card, or a comma followed by a blank column is encountered. The scan of a continuation card begins in column 4 of the card.

Except for the continuation of the scan of a PICTURE parameter, the first nonblank character in a continuation card indicates the presence of a keyword. In the case of the PICTURE parameter scan continuation, the character (or blank) in column 4 and any succeeding column(s) are recognized as belonging to the PICTURE parameter. This permits the presence of imbedded blanks and delimiter characters in the PICTURE parameter.

THE BASE MODULE (IEBDG) CHARTS 60,61,62

Module IEBDG is the first module of the data generator program to be placed in main storage. It is entered from a calling program and returns control to the calling program at the completion of the data generator program. Depending on the requirements encountered during the processing performed by this module, it will give control to (and receive control from) one of the following modules: the message module, the clean-up module, the FD analysis module, or the create analysis module.

The primary functions of the base module are:

- To get storage for a work area (the common communication area).
- To open the input, output, and message data sets.
- To read the utility control cards.
- To cause error messages to be displayed.
- To pass control to the appropriate module as required.

Initialization

Upon entry to this module, registers are saved for a later return to the caller. By use of an SVC 10 instruction, storage is obtained for a common communication area. This area is then given initial (default) values for ddnames, line count (for printer control), and paging information. To provide for the specification of a random binary number format for the output data set, an initial multiplier value is established for a random number generator and placed in the communication area.

If a calling program has invoked the IEBDG program by means of a LINK or an ATTACH macro instruction, the previously assigned default values in the communication area are replaced by the values specified in the parameter list for the invocation. The assigned names of SYSIN for the utility input control data set and SYSPRINT for the utility output control data set may be changed as a result of an invocation. If so, the changed names are effective for the duration of the job. After invocation, the input and output control data sets are then opened.

Opening Data Sets

If the IEBDG program is called by use of the job control language statements, the input (SYSIN) and message (SYSPRINT) data sets are opened and default values assigned as required.

Each time a data set is opened, a DCB exit routine in the base module is entered. The entry points to this routine are determined by the function (input, output, SYSIN, or SYSOUT) of the data set being opened. At each entry, the routine establishes default values for the record format, the logical record length, and the blocksize for the data set.

A common section of the DCB exit routine is then entered to inspect the actual

values of record format, logical record length, and blocksize. These values normally have already been placed in the respective fields of the DCB by the open routine.

For data sets having a fixed record format, the common routine determines if the block size is an integral multiple of the logical record length. An integral multiple is required; otherwise, default values are assigned (if not previously assigned) so that an integral multiple is assured.

As the DCB exit routine evaluates the preceding record parameters for input or output data sets, it sets the FLUSHSW switch (at COMMON + 572)[1] to one if default values are assigned. (If the switch is set, then, when the base module again receives control, it flushes the control cards and procedes to terminate the job.) The exit routine then returns control to the open routine to complete the opening of the data set.

In testing for a successful data set opening, only the input (SYSIN) data set is tested by the base module. Because a user may not desire any messages, or may not have enough space available for an output data set for messages, the testing for a successful opening of the output (SYSPRINT) data set is done by the message module when the module is first needed.

Messages

When messages are required during the processing by the base module, a linkage is made to the message module. Upon return from the message module, processing will continue or, depending on the severity of the situation causing the message, a return is made to the calling program. [When any of the modules of the data generator program require the printing of an error message, control is returned from the module in command to the base module, which will then link to the message module. Depending on the severity of the error causing the message, control may or may not be returned from the base module to the module that was in command.] A condition code, CONDCODE, (at the field COMMON+404), is set prior to giving control to the message module. Upon return from the message module, this code is checked to determine the severity of the situation. The base module returns control to the calling program by freeing the storage space for the common communication area, restoring the calling program's environment (registers), and issuing a BCR instruction.

After the input data set has been opened, a program heading message and an indication of any PARM field (on the EXEC card for the program) errors that may be present are placed on the SYSPRINT data set.

Reading Control Cards

The GET macro instruction is used to place a control card in the input work area. The card image is printed on the SYSPRINT data set, and tests are made to determine the type of card in the input area. For either an FD control card or a CREATE control card, the base module will give control to the appropriate processing module.

For any new group of data generator control cards, the first nonblank card must be a DSD control card. [If a blank card is present, it is merely flushed through and the next card is checked.] In order to indicate when a DSD control card is detected, a switch, DSDSW (at COMMON+550), is set to 1. This switch is tested for all but the DSD card in a group of control cards. If the first card in a group is not a DSD card, the syntax of the other control cards may be checked, but the program will not be executed. An error message will indicate the reason.

Following the test for a DSD card, the other utility control cards are checked for card types. The finding of a particular type causes the base module to give control to the proper module for processing of that control card. If a continuation card belonging to a given control card, is encountered, the base module gives control to the appropriate control card processing module to scan the card. Should a DSD control card have no CREATE control card(s) between it and either an END card or a /* card, the resulting output data set that is created will be a null data set (i.e., no picture or pattern will be produced).

As the base module continues its scan, a check is made for a blank following the card type (DSD, FD, etc.) as well as for improper control card names or name length. Errors in one of these areas will cause a message to be printed and the program will not be executed.

Included within the routines of the base module is a SYNAD routine for the SYSIN

---

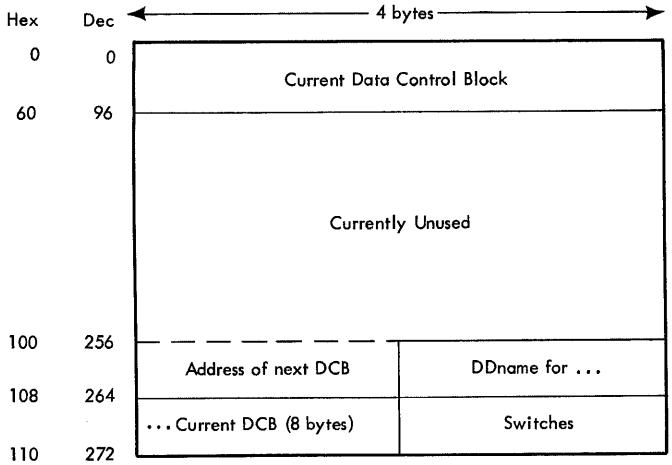
[1] In the discussion of the modules of the data generator program, references to locations in the common communication area are indicated by giving the decimal value of the displacement, or offset, from the start of the area. As an example, the offset of the field CONDCODE (condition code setting) would be given as COMMON + 404.

data set. The SYNAD routine obtains unrecoverable I/O error information that is to be printed on the SYSPRINT data set. (The message module contains a SYNAD routine for the SYSPRINT data set; the create module contains a SYNAD routine for input and output data sets.) After the information is printed, control is given to the clean-up module.

Base Module Card-Processing

The following data generator control cards are processed by the base module: The DSD card, the REPEAT card, the END card, and the DUMP card.

DSD CARD PROCESSING: In requesting storage for the user's DCB, allowance is made for future implementation to satisfy an indexed sequential data set. Figure 56 indicates the allocation of the storage area's 272 bytes for the current support.



• Figure 56. Storage Area Obtained by Base Module for Current DCB

A storage area is obtained as required for each of the data sets described by the DDnames on the DSD control card. In the case of storage for the last data set's DCB, the four-byte field beginning at location 256 (hex. 100) is zero.

REPEAT CARD PROCESSING: When the base module scans the parameters of the REPEAT card, it sets an indicator, QUANSW (at COMMON+576), to record the finding of the required keyword. After each valid keyword is found, the numerical value of its parameter is packed and converted to binary. Since 65,535 is the largest number that can be held in a 2-byte storage field, any parameter value that is greater than that results in a message to the programmer. Acceptable values for the QUANTITY and the CREATE parameters are stored for use by the create analysis module.

END CARD PROCESSING: When an END control card is encountered, the base module gives control directly to the clean-up module if all of the required number of entries specified on the REPEAT control card have been processed. Otherwise, a message is printed and then control is given to the clean-up module. Upon return from the clean-up module, the base module reads the next control card (which may be either a data generator control card or a /* delimiter card). There may be one or more additional groups of data generator control cards before a /* card.

DUMP CARD PROCESSING: The reading of a DUMP control card causes a printout of the user's program and/or storage areas assigned 0 to his program. When the DUMP control card is encountered, the base module places a zero in register 15 and forces an ABEND dump by branching to that register. Further description of the use of a DUMP control card is given in the section Service Aids.

THE CLEAN-UP MODULE (IEBDGCUP) CHART 63

When either an END control card, indicating the end of a group of data generator control cards, or a /* delimiter card, indicating the end of a job, is encountered, the base module gives control to the clean-up module.

All user input and output data control blocks (DCBs) that have been opened are closed. For each of these DCBs, any buffer pools that data management routines had obtained for use by the data generator program are released to the system. The 272-byte storage area(s) that the base module obtained for each of these DCBs are also released to the system.

If the entry to this module was the result of encountering an END control card, this module returns control to the base module for the purpose of checking for another group of control cards.

If the entry to this module resulted from encountering a /* delimiter card, this module will close both the system input (SYSIN) and the system output (SYSPRINT) DCBs of the data generator program, and free any related buffer pools for these data sets.

The storage area that was obtained for the data generator program's input and output DCBs (96 bytes each) was initially obtained as a part of the common communication area by the base module. Therefore, the base module will release this area after it receives control from the cleanup module.

THE FD ANALYSIS MODULE (IEBFDANL) CHARTS 64,65

This module scans and analyzes the parameters on the FD control card. Module IEBFDANL is initially entered from the base module. If module IEBFDANL does not encounter a condition that causes termination of the job, it will use the FD table module (described later on) as a subroutine. After the FD table module returns control to the FD analysis module, the latter module returns control to the base module.

The FD analysis module begins the assignment of information to a table called the FD table. This table is used by both the create analysis module and the create module. The FD table module completes the construction of the table.

An FD table entry has 64 bytes. Storage for the FD table is obtained in increments of 512 bytes (enough for eight table entries) by the FD analysis module. Each entry contains most of the parameter information (or a processed version of the information) from one FD control card. If a PICTURE keyword has been specified on the FD control card, the picture information is placed in another area of main storage. The FD table is shown in Figure 57.

Upon entry to the FD analysis module, tests are made to determine whether or not the entry is due to a continuation card. Such an entry may be due to the continuation of the parameter string on a card, or to the continuation of the PICTURE parameter on a card. If the entry is due to either a continuation card or a picture continuation, storage for an FD table entry may already be available as the result of processing a previous FD control card in the same set of data generator control cards. If the entry is not due to a continuation card, an FD table entry is to be constructed. A GETMAIN macro instruction is issued to obtain storage for an FD table.

FD Card Scanning

The scan of the actual FD card keywords and their associated parameters is then performed. As each keyword is encountered, its parameter is scanned, validated and/or converted if required, and then placed in a reserved spot in the FD table. If a keyword error or a parameter error is encountered, an appropriate message will be printed on the system output device. The severity of the error determines whether the program is terminated at that point or whether modified processing (e.g., syntax checking only) will continue. Control and storage tables are constructed even for syntax checking procedures.

A user may specify either the FORMAT or the PICTURE keyword, but not both, on the same control card. The FD analysis module sets a switch, either FDFMTSW (at COMMON+539) or FDPLSW (at COMMON+540), when it encounters one of these keywords. If the other keyword is then encountered in the scan of the same card, a test of the previously-mentioned switch for the keyword first encountered reveals the error.

Table 2 lists the keywords of the FD control card and indicates the processing done on the parameters of the keywords by the FD analysis module.

After the keyword parameters on the FD control card have been scanned and placed in the FD table, the FD analysis module gives control to the FD table module to complete the construction of the FD table entry. The FD analysis module assigned only the initially specified values of parameters to the FD table. If any keyword except LENGTH and NAME was omitted from the FD control card, the FD analysis module does not perform processing for the keyword and does not fill in the appropriate space in the FD table entry. Default values for keyword parameters are assigned by the FD table module.

THE FD TABLE MODULE (IEBFDTBL) CHARTS 66,67

This module completes the construction of the FD table, which was begun by the FD analysis module, At the time of entry into this module, an FD card has been completely scanned and initial values from the card have been placed in the FD table. The FD analysis module uses a LINK macro instruction to give control to the FD table module, which is then used as a subroutine by the FD analysis module. The FD table module returns control to the FD analysis module.

FD Pattern Construction

Initially, module IEBFDTBL determines the type of picture or format specified in an FD control card. (This field will be used by the create module when it constructs the output records.) If neither a picture nor a format is specified, the FD table module assigns a default value to the field.

Before further processing is done on a non-EBCDIC picture, the picture numbers are checked for validity by comparing the zone bits of the numbers against a hexadecimal "F". An incorrect value results in an error message indication, and control is returned to the calling module. [A picture

having a packed decimal specification must have a length specification that is less than or equal to 16 since the Pack instruction can handle up to 16 bytes.] Otherwise, the numbers are converted to the specified form, storage is obtained for the picture, and the picture is moved into the storage area (from the temporary storage area into which the picture had been placed by the FD analysis module). The temporary storage area is then released and the FD table module gives control back to the FD analysis module.

Except for the NAME and LENGTH keywords, the FD table module assigns default parameter values for each keyword that is omitted from an FD control card. The values assigned are shown in Table 2, and they are placed in the FD table.

For a pattern, which may be either a user EBCDIC picture specification or an IBM format specification, module IEBFDTBL determines the action that is specified on the FD control card. Based on this determination (including a possible default determination), the module makes an entry in the FDACTION field of the appropriate FD table and sets the appropriate bit in the FDSWITCH field of the table to one.

The module then determines the amount of storage required to hold the pattern. The amount of storage required depends on the action which the create module will later apply to the pattern. By means of the GETMAIN macro instruction, the FD table module obtains the necessary storage. To provide for a wave or a ripple type of action, the storage area must contain two contiguous copies of the pattern. If the action is a roll, three contiguous copies of the pattern must fit in the storage area. The create module requires the repeated patterns when it generates the output records.

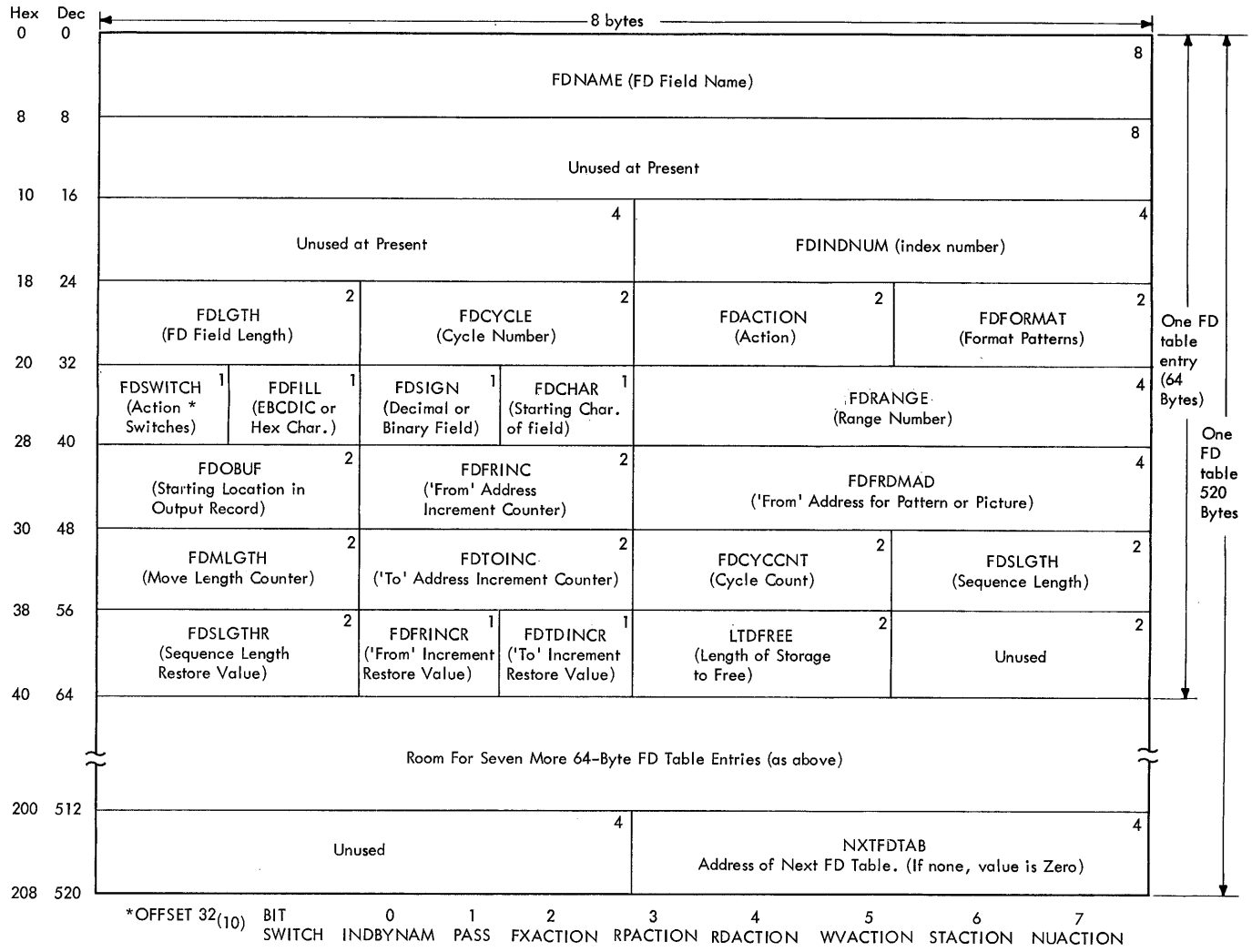


Figure 57. FD Table Constructed by FD Analysis Module and FD Table Module

• Table 2. FD Control Card Keyword Parameter Processing, and Default Values Assigned, if Required

FD Keyword	Processing Applied to Keyword Parameter			Default Value (If any) **
	Validity Checked	Value Converted	Other Processing	
NAME *	Yes	No	Length checked for maximum of eight characters.	None.
LENGTH *	Yes	Yes	None.	None.
STARTLOC	Yes	Yes	Subtract one from value.	First available byte in record.
PICTURE (length)	Yes	Yes	Check for occurrence of FORMAT keyword.	None.
PICTURE (field)	No	No	Get storage for picture. Determine type of picture. Move picture to storage. (Include continuation cards.)	Fill character.
FORMAT	No	No	Check for occurrence of PICTURE keyword. Check for two-character pattern.	Fill character.
ACTION	No	No	Check for two-character type.	FX (Fixed)
FILL	No	Yes (Hex only)	Check for EBCDIC or Hex type with two digits.	Binary zeros.
CYCLE	Yes	Yes	None.	One.
RANGE	Yes	Yes	None.	None.
CHARACTER (of FORMAT)	No	No	None.	A (for alphabetic and alphameric). 'blank' (for collating).
SIGN	No	No	Determine if sign value is valid.	Plus.
INDEX	Yes	Yes	None.	None.

* These keywords are required to be present in the FD control card. If not present, the program will be terminated.
** Default values assigned by FD table module.

After storage is obtained to accommodate the desired pattern action, the module places the specified fill character or a default fill character in each byte of the area. It then moves the pattern into the storage area the required number of times. Any leftover space (due to differences in field length and picture length specifications) contains the fill character.

If a PICTURE keyword had been specified, the temporary storage area that the FD analysis module had used to hold the picture is released before the FD table module returns control to the FD analysis module. If a FORMAT keyword had been specified, the starting character for an alphabetic, alphameric, or a collating sequence field is resolved before control is returned to the FD analysis module. For other formats, the storage field is initialized to a value that depends on the format specified. (For binary format, the value is a binary 1; for packed decimal, the value is a packed decimal 1; for zoned decimal, the value is a zoned decimal 1.)

For the FD table module, a 63-byte sequence of characters resides in storage at location COPAT. The 28th byte of this sequence is at location ALPAT. After resolving the starting character for an AL, AN, or CC format, the module fills the pattern field using the characters of this sequence. If the starting location value is a default value, a collating sequence pattern begins at location COPAT, and an AL or AN pattern begins at location ALPAT.

The pattern field is filled only in increments that are equal to or less than the length of the sequence that is being used for the format pattern. If the length of the field (given at decimal offset 24 in the FD table) to be moved is less than the indicated sequence length, the number of characters moved will be equal to the FDLGTH field value. If the length of the field to be moved is greater than but not an integral multiple of the indicated sequence length, the number of characters moved for all moves but the last will be equal to the sequence length. The last move will contain the characters remaining after an integral number of moves have been made, each move containing the number of characters in the given sequence. If the FDLGTH value is equal to an integral multiple of the sequence length, the number of moves is equal to the integral number.

THE CREATE ANALYSIS MODULE (IEBCRANL) CHARTS 68,69,70,71,72

This module scans and analyzes the parameters on the CREATE control card. The initial entry to module IEBCRANL is from base module when a CREATE card is encountered. Other entries to the module occur when create continuation cards or create card comments cards are encountered. If the create analysis module does not encounter a condition that suppresses the creation of output records, it will use the create module as a subroutine to generate output records. The create module will return control to the create analysis module, which will, in turn, return control to the base module.

Table Construction

The create analysis module constructs four types of tables that are used by the create module:

- The create table
- The picture table
- The FD address table
- The exit name table

The create table is the largest of the four. It may contain one or more create entries. Module IEBCRANL establishes a 28-byte create entry for each CREATE control card that it processes. (See Figure 58.) One create table may contain up to 18 create entries. These entries contain pointers to picture tables and FD address tables. More than one create table may be constructed.

The picture table contains information about, as well as the actual, picture string that may be specified on a CREATE card.

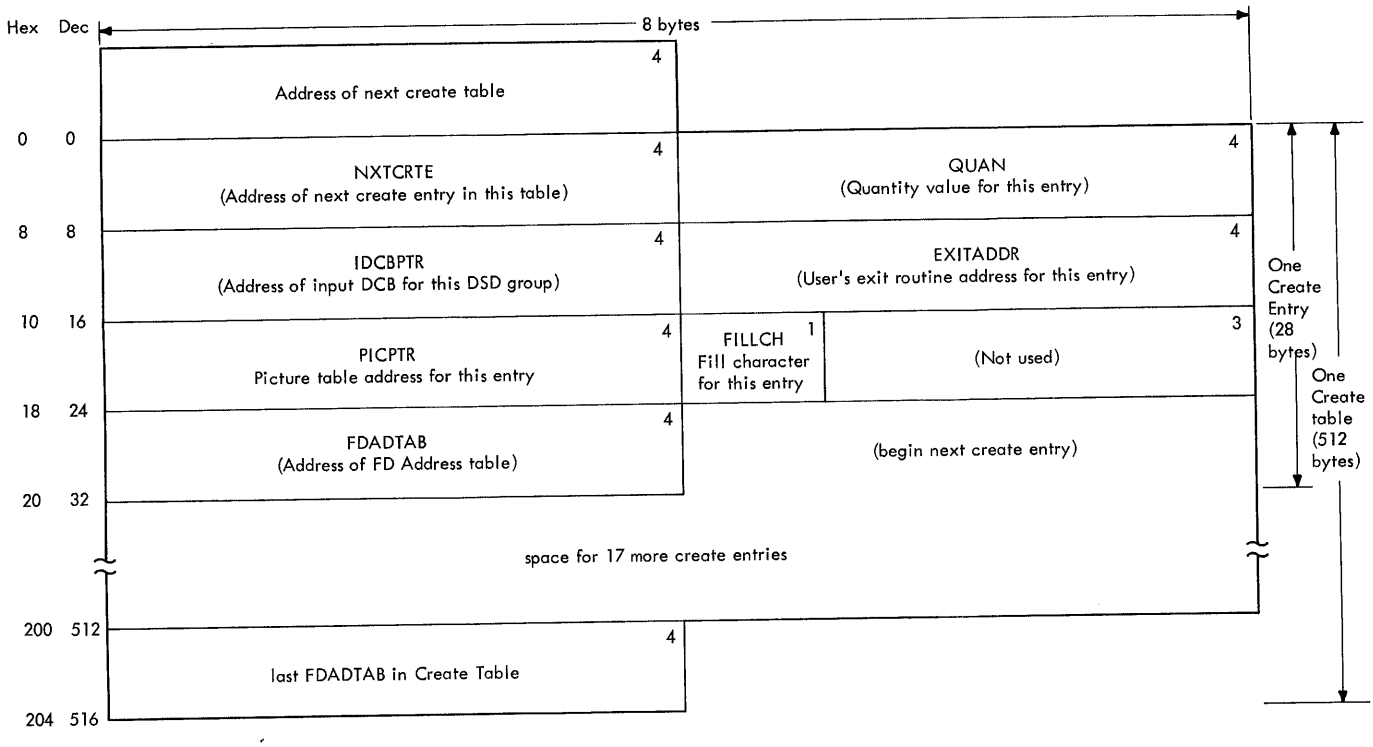
The FD address table contains the addresses of the FD table that have been constructed to contain information from FD cards.

The fourth type of table that the create analysis module constructs is the exit name table. This table contains the names of any user exit routines that have been specified. When a user's exit routine is loaded into main storage, the storage address of the routine is placed in the create table.

Module Entries

Since the create anaysis module may have been entered before in processing a given group data generator control cards, the initial analysis performed upon entry to the module consists of determining the cause for the module's receiving control. The create continue switch, CRCSW (at COMMON+564), is tested for this purpose. If the entry to the module is the first one for a given CREATE control card, storage for a create entry is obtained either from an existing create table or by the issuance of a GETMAIN macro instruction for space for another 512-byte create table. (As each new create table space is obtained, it is 'chained' to the previous space and initialized to all zeros.) Then, the module scans the control card keywords one at a time. If an invalid keyword is encountered, the create analysis module indicates a message, sets the NOGOSW switch (at COMMON+551) to suppress the creation of output records, and gives control back to the base module to continue the checking of syntex on other control cards.

If the entry to the create analysis module is due to a continuation of a CREATE control card, a check is made to determine if the parameters of either the NAME keyword or the PICTURE keyword were interrupted. (Except for the NAME and PICTURE keyword parameters of the CREATE card, all other CREATE card parameters must be on the same card as their associated keywords. The picture string parameter of the PICTURE keyword is the only one that may in itself be continued to another card.) The name continue switches, NAMCSW1 and NAMCSW2 (at COMMON + 561), and the picture continue switches, PICCSW1, PICCSW2, and PICCSW3 (at



• Figure 58. Create Table Constructed by Create Analysis Module

COMMCN + 562), are tested to determine if a parameter may be on a continuation card. For a continuation card, the scan begins in column 4.

If the entry to the create analysis module is because of a comments continution, control is given to the base module, IEBDG, to process the comments, and to read the next control card.

Module Subroutines

As the create analysis module processes each valid keyword, it may branch to subroutines within the module. These subroutines perform functions of parameter scanning (SPSCAN routine), packing and/or conversion (CONVDB routine), and table searching (FDSRCH routine). When the processing for a given keyword is completed, the keyword scanning section of the module scans the next keyword unless a continuation card has been indicated. If the latter action has occured, module IEBCRANL gives control to the base module to read the next control card.

THE SPSCAN SUBROUTINE: The function of the SPSCAN subroutine is to check the validity of a keyword parameter. The routine moves a pointer across each character in the parameter and checks for commas, blanks, and parentheses. It also determines whether a parameter has been extended into column 72 of a control card. If so, an error message is indicated and control is given back to the base module.

After a comma or a blank is encountered, the parameter length is determined and checked for an invalid length of zero, and the routine returns control to the calling section of the module. (An invalid parameter length causes the module to indicate a message and return control to the base module.)

THE CONVDB SUBROUTINE: The convert subroutine, CONVDB, performs two functions: it converts a decimal number to a packed decimal number, and it converts a decimal number to a binary number. The subroutine is used in processing the parameters of the QUANTITY, the NAME, and the PICTURE keywords. The convert subroutine processes a number that can be contained in 4 bytes or less. Therefore, a decimal value that is greater than 2,147,483,647 will not be processed, and a message will be indicated. Module IEBCRANL then returns control to the base module.

In order to determine if a parameter to be converted is numeric, the zones of its characters are compared against a hexadecimal 'F.' Valid numeric characters are then converted to a packed decimal format and placed in the storage area Q (at COMMON+ 216). For all cases except a packed decimal picture specification, the parameter value is then converted to a

binary value and placed in a general register.

THE FDSRCH SUBROUTINE: Module IEBCRANL contains and uses the FDSRCH subroutine in processing the NAME keyword parameter. The subroutine places a valid name from the CREATE control card into storage and then compares this name against the names of the FD tables (which have been established by the FD analysis module). If the list of FD tables does not contain a name that matches the name on the CREATE card, a message is indicated and the create analysis module returns control to the base module.

When an FD table name that compares with a CREATE card name parameter is found, the address of the FD table bearing that name is placed in an FD address table. (See Figure 59.) If there is no room in an existing FD address table, the FDSRCH subroutine will obtain storage for a new table. The current create entry, whose address is given at CURCRTE (COMMON + 316) contains the address of the first FD address table. All FD address tables are chained together by pointers in the tables themselves.

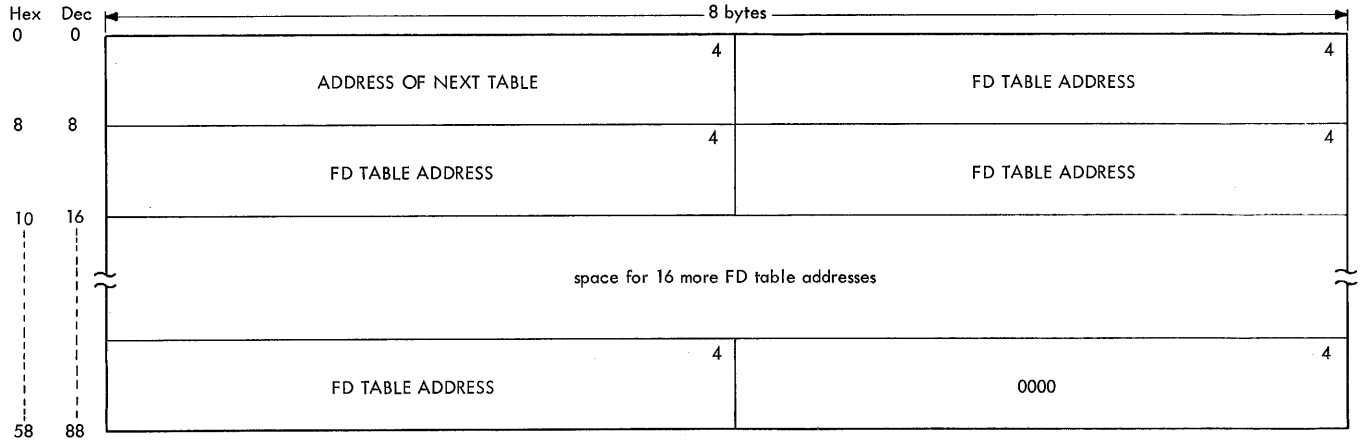
Keyword Processing

To determine if all keywords on a given CREATE card have been processed, module IEBCRANL tests the column after the last parameter of each keyword. If this column is blank and if column 72 of the control card is blank, the last parameter on the card has been processed. The module then establishes a default value for the QUANTITY keyword parameter if a value has not already been supplied on a CREATE control card.

If there are no more continuation cards for a given CREATE control card and if the create value (from a preceding REPEAT control card group) is equal to one, module IEBCRANL gives control to the create module, IEBCREAT. Otherwise, if the create value (determined by testing the field CREATENO at COMMON + 18) is greater than one, the create analysis module gives control to the base module to read the next CREATE control card. (The field CREATENO is initially set to one in case a CREATE card is not part of a REPEAT card group.)

If the column after the last parameter contains a comma, the next card column is checked. If this next column either is column 72 or contains a blank, the module gives control to the base module to read a continuation card. Otherwise, either a message would have been indicated and control returned to the base module or the subroutine for scanning the next keyword will be entered.

When the create analysis module processes the parameter of the DDNAME keyword, it tests the number of characters in the DDNAME and determines whether the parameter value is SYSIN. If the name length is valid and the name is SYSIN, the address of the SYSIN data control block (at COMMON + 116) is placed in the create entry (whose address is given at COMMON + 316) for the CREATE card being processed. If the name is not SYSIN, the input DCB(s) are scanned to find a name equal to the CREATE card's ddname. In doing the scanning, the address of the first input DCB is placed in the DCB pointer (at COMMON + 300). The name of the DCB (at DCBD + 260) is compared to the ddname given on the CREATE card. Unless an equal name is found, the process repeats with the next input DCB until there are no more input DCBs to check. If a successful DCB name comparison is made, the input DCB address is placed in the create entry. Otherwise, a message is indicated and the create analysis module gives control to the base module.



• Figure 59. FD Address Table Constructed by Create Analysis Module

If the ddname search was successful, either the given delimiter or a default delimiter is placed in the DELIM field (at COMMON + 344), or a message signifying an invalid delimiter is indicated and the create analysis module returns control to the base module.

When the EXIT keyword is encountered, the length of the user's exit routine name is checked for validity. If the length is valid, the name is placed in a table called the exit name table (see Figure 60). The user's routine is then placed (via a LOAD macro instruction) into main storage, and the storage address of the routine is placed in the create entry.

THE NAME KEYWORD: In processing the parameters of the NAME keyword, module IEBCRANL searches for multiple names, for 'copy' groups (based on the COPY keyword), and for breaks or interruptions in series of names within outer and/or inner parentheses. If the COPY keyword is not present and if multiple names have been indicated (by encountering a left parenthesis in the scan), a default value of one is assigned to the COPYVAL field (COMMON + 640).

The complete processing of the NAME keyword parameter(s) includes the use of the subroutines FDSRCH, SPSCAN, and, if the COPY keyword is present, the CONVDB subroutine. If multiple names are present, the SPSCAN and FDSRCH subroutines are used for each name that is encountered. The create analysis module indicates that there is a continuation in the parameters of the NAME keyword by setting switches in the NAMCSW field (CCMMON + 561) of the communication area. If the continuation occurs after a 'name' subparameter within only the outer set of parentheses, the high-order bit (bit 0) of the NAMCSW field is set to one. For a continuation indication that occurs after either the COPY keyword or a 'name' subparameter within the inner set of parentheses, the second bit (bit 1) of the NAMCSW field is set to one.

If an inner group of names is to be copied more than once, the create analysis module checks the current FD address table for enough space each time a name is to be copied. If space is not available, storage for a new 88-byte FD address table is obtained, and the new table is chained to the previous one by the first word in the current FD address table located at the address given in the CURFDGM field (COMMON + 632).

THE PICTURE KEYWORD: This section describes the processing of the PICTURE keyword parameters. The PICTURE length parameter is processed first; the start location of the picture string is then processed; and the actual picture processing is done last. In the case of the PICTURE keyword parameters, there are three ways in which a continuation card may be encountered.

1. The PICTURE parameter list may be interrupted after the length parameter. In this case, the first (high-order) bit of the PICCSW field (COMMON + 562) is set to one.
2. The PICTURE parameter list may be interrupted after the startloc parameter. In this case, the second bit of the PICCSW field is set to one.
3. The actual picture string may be interrupted. In this case, the third bit of the PICCSW field is set to one.

Chart 70 indicates the entry points to the section of the module in which processing for the continuation card relating to each of the above ways of interruption takes place.

In processing the length parameter, the module first scans the length and then coverts the length value to a binary equivalent. Based on this length, the module obtains a storage area called the picture table. (See Figure 61.) The binary length value is then placed in the field PICLGTH, which begins at the fifth byte of the associated picture table. The picture table is located at an address given in the PICBASE field (at COMMON + 664) of the communication area. This same picture table address is also placed in the create entry for the current CREATE card.

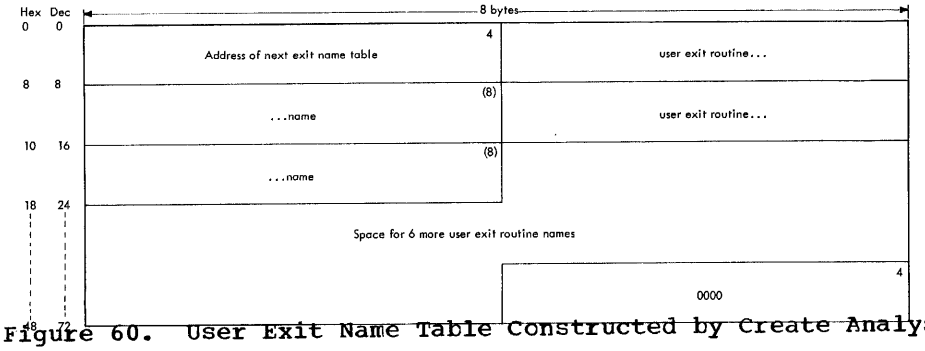
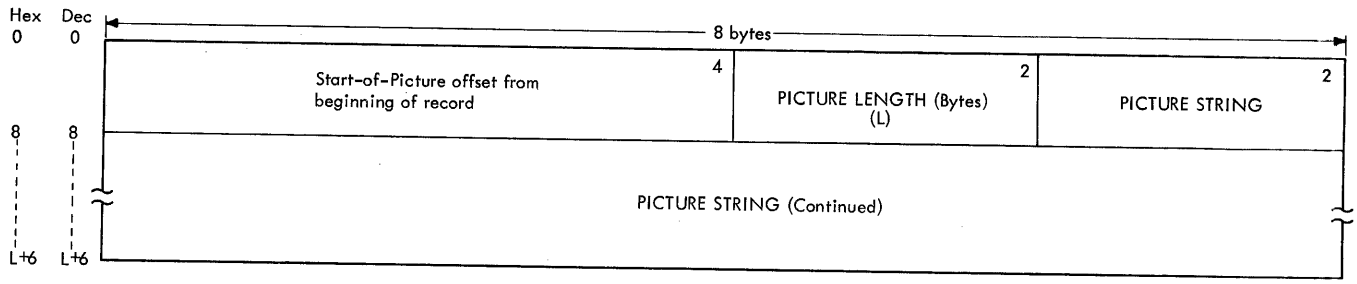


Figure 60. User Exit Name Table Constructed by Create Analysis Module



Note: L is equal to the value specified as the length subparameter of the PICTURE keyword on the related CREATE control card.

• Figure 61. Picture Table Constructed by Create Analysis Module

The start location (startloc) parameter is scanned; if valid, it is converted to a binary value; and the value is then placed in the associated create entry. As it does after the length parameter, the module then checks to see if the next parameter is on the same or a continuation card. If a continuation card is indicated, the module returns control to the base module to read the next card. Otherwise, the picture string will be processed.

If the picture string is specified as being in EBCDIC (character), the string characters are moved directly from the card to the picture table. If the picture string is to be continued, the continuation switch (second bit of the field PICCSW) is set to one, and control is returned to the base module to read the next card.

If the picture string is specified as being in either packed decimal or binary, the complete string must be on one card. The card format is checked, and if valid, the string value is converted either to a packed decimal value or to a binary value as specified on the control card. The converted value is then placed in the picture table for use by the create module.

After each parameter on the CREATE control card has been processed, the create analysis module checks for a valid delimiting character and for an indication of a continuation card.

THE CREATE MODULE (IEBCREAT) CHARTS 73,74

The create module maintains control over the generation of output records. This module receives control from the create analysis module after create entries (in one or more create tables) and related tables have been constructed. If, upon entry to this module, the switch, NOGOSW (at COMMON + 551), is on as the result of the action of a previous module, the generation of output records will be suppressed and the create module will perform only its clean-up functions. Module IEBCREAT always returns control to the create analysis module, which then returns control to the base module for the printing of messages and/or the reading of the next control card.

Output Record Modifications

Upon entry to the create module, the NOGOSW switch is tested to determine whether to continue processing or whether to immediately release storage areas and return control to the create analysis module. If processing is to continue, the record characteristics (length, format) are determined; a counter, RECREM (at COMMON + 348), is initialized with the quantity value from the create entry; and the input record size (if present) is compared to the output record size. The output record field is then filled with the create entry fill character prior to reading in a record from an input DCB.

FD TABLE MODIFICATION: If there is no input DCB, modifications based on values in the FD table(s) are made directly to the output area containing the fill character. Otherwise, the modifications are made to the input record that has overlaid the fill characters in the output area.

The modifications based on the FD table values involve the action, index, cycle, and range parameters from the FD control card. Initially, an FD pattern at the storage address given in the field FDFROMAD of an FD table is moved to the output area, which, at this point, contains either a fill character or an input record. The create module then inspects other FD tables that may have been indicated by the CREATE control card as belonging to the current create entry and moves the patterns from these tables into the output area. The output record starting location for each FD pattern is given in the field FDOBUF of the FD table. Note, that as each modification is made to the output record, it may overlap part or all of a previous modification depending on the starting location specifications involved.

PICTURE AND USER MODIFICATIONS: After the create module moves the FD pattern(s) to the output record area, the module moves in the picture string from the CREATE control card, if the PICTURE parameter has been specified. Otherwise, or after the picture string has been moved, module IEBCREAT determines if a user exit routine is present. If so, it indicates that a user may desire to inspect and/or modify the record before it is placed on the output device. An exit is taken to permit user modification if this is the case. After the user routine (if one is used) gives control back to the create module, the create module checks the return code that has been placed in register 6 by the user's routine. If the job is not to be terminated at this point, the create module places in the output data set the record that is in the output area. If termination is to take place, an indicating switch (FLUSHSW or FLUSHSW1 depending on the return code) is set, storage areas are released, and control is given to the create analysis module, which then gives control to the base module.

Updating the FD Table

After each record is placed in the output data set, certain values in each applicable FD table are updated to prepare for the next output record that is to be created. Multiple references, by the same create entry, to the same FD table are indicated by the setting of a 'pass' switch (bit one of the FDSWITCH field). If this occurs, the FD table is processed (and updated) only once.

For the binary, packed decimal, and zoned decimal patterns, the create module performs the following actions by using FD table values:

- The cycle count field (FDCYCCNT) value is incremented if the cycle value (FDCYCLE) is other than zero.
- The pattern values are then converted to a binary equivalent, if not already binary, and placed in a work area (register 4). The module then increments the binary-equivalent pattern value by using the index number (FDINDNUM).
- The incremented pattern value is then tested against the range value given in the field FDRANGE. If the range value has been exceeded, the current pattern value in the storage area to which the FD table refers is not changed. Otherwise, the indexed binary value is reconverted to a decimal form if necessary and placed in the storage area.

For a random number format, the random generator routine of the create module provides another value to be used for the next record and places the value in the pattern storage area to which the FD table refers.

For alphabetic or character patterns, the generation of the pattern to be placed in the output area for the next record requires that values of the 'from' address in the FD table and the 'to' (or output work area) address be changed. These addresses are used in moving the pattern (or a part of it) from the pattern storage area to the output record area. In the FD table, there are two fields (FDFRINC and FDTOINC) that contain the increment values used to modify the 'from' and the 'to' addresses. These fields initially contain a value of zero for the first output record. For subsequent records, the values may be incremented by values given in the increment restore fields (FDFRINCR and FDTOINCR) of the FD table.

The FD table module established the values of the increment-restore fields after the specified action had been determined. Table 3 lists the values of the increment-restore fields for the various actions that may be specified.

• Table 3. Values of Increment-Restore Fields in the FD Table

Pattern	FDFRINCR ('From' Increment Restore)	FDTOINCR ('To' Increment Restore)
Shift Left	1	0
Shift Right	0	1
Truncate Left	1	1
Truncate Right	0	0
Roll	+1 (*) -1	0
Ripple	1	0
Wave	1	0
Fixed	0	0

*This value will alternate between +1 and -1 as the roll pattern is developed in first one direction and then the other.

Depending on the action specified in the FD table, the create module may vary the values of the move length counter field, FDMLGTH, and the sequence length counter field, FDSLGTH, to prepare for the next output record. Table 4 summarizes the changes that may occur to values of fields in the FD table as the create module generates output records.

• Table 4. Changes Made to FD Table Values as Create Module Builds Output Records

Field	Format	Change
FDCYCCNT	Numeric	Increase by 1. When = FDCYCLE value, set to 0.
FDMLGTH	Alphabetic (Shift or Truncate)	Initially = FDLGTH value. If FDMLGTH > 1, decrease by 1. When FDMLGTH ≤ 1, set = FDLGTH value.
FDFRINC	Alphabetic (Shift or Truncate)	If FDMLGTH > 1, increase by value in FDFRINCR field. When FDMLGTH ≤ 1, set = 0.
FDTOINC	Alphabetic (Shift or Truncate)	If FDMLGTH > 1, increase by value in FDTOINCR field. When FDMLGTH ≤ 1, SET = 0.
FDSLGTH	Alphabetic (Ripple)	Initially = FDSLGTHR. If FDSLGTH > 1, decrease by 1. When FDSLGTH ≤ 1, set = FDSLGTHR value.
FDFRINC	Alphabetic (Ripple)	If FDSLGTH > 1, increase by 1. When FDSLGTH ≤ 1, set to 0.
FDFRINC	Alphabetic (Wave)	If FDSLGTH > 1, increase by 1. When FDSLGTH ≤ 1, restore to 0.
FDMLGTH	Alphabetic (Wave)	When FDSLGTH ≤ 1, restore to FDLGTH value.
FDFRINC	Alphabetic (Roll)	Increase by 1 for roll to left. Decrease by 1 for roll to right.

After all FD tables to which a given create entry refer have been updated, the create module inspects the NXTCRTE field (in the create table) to determine the address, if any, of the next create entry to be processed. (When there are no more create entries to be processed, the NXTCRTE field of the current create entry contains zeros.) If there is another to be processed, the create module will process the entry in the manner already described.

If there is a repeat function to perform, the entire list of create entries must be processed as many times as necessary to satisfy the repeat requirement. When that is done, the clean-up functions of the create module will be performed. If there is no REPEAT card function to perform for the current set of data generator control cards, the field REPEATNO (at COMMON + 16) contains zeros.

After the last create entry has been processed, the create module will release the storage areas that have been obtained for create tables, FD address tables, and the CREATE card picture string. The module then resets switches and communication area field values for an initial entry to the create analysis module, and returns control to the create analysis module.

THE MESSAGE MODULE (IEBDGMSG) CHART 75

Message module IEBDGMSG is entered from the base module whenever there is an indication of a message to be printed, or placed on the output device. To indicate the need for a message, the other modules of the data generator program set a message number in the MS field (at COMMON + 406) of the communication area.

This module places four types of messages on the output device: heading messages, control card images, error messages, and error flags (messages). The messages used for headings and errors exist as 121-byte entries within the message module. The location of each message within the module is contained in a 4-byte address entry in a message pointer table.

Before any messages are placed on the output device, module IEBDGMSG determines if the output data set has been successfully opened. If it has not been opened, control is returned to the base module and the job is terminated. If the data set is open, the value in the MS field is checked to determine the reason for requesting the module.

Initially, the module is requested to print a heading message (MS field value =

1). Thereafter, a heading message is printed when the module finds an indication of either a channel 12 printer carriage tape or the correct line count. All heading messages will begin at position one on the output device. After each heading message printout, the line count value is reset to either the user-specified value or the default value, and the page number value is incremented. Before printing a heading message, a printer will skip to channel one to set up a new page. When any other message is to be printed, a printer will space one line before printing the message.

If the MS field value is not 1, the module determines if the carriage control tape indication is 12 and, if the indication is not 12, if the line count value has reached its maximum value. If either situation has occurred, a heading message will be printed, the line count will be reset, and the page number will be incremented

Otherwise, the module tests the MS field to determine if a control card image is to be printed from the input buffer. If this is the case, the image is printed at position ten on the output device. If a card image is not printed, the module tests for a regular error message indication from the processing modules. These messages have message numbers from 2-28. For each message to be printed, the printer is placed at position one to receive the message.

The message module places a flag message (consisting of the word ERROR) in the message data set when one of the other modules of the data generator program requests an error flag. This message begins on the line below the corresponding control card image and in the column corresponding to the card column that is in error.

After a message has been placed on the output device, the message module increments the line count value, determines if a heading message has just been placed on the device, and either continues processing or returns control to the base module.

SERVICE AIDS

A customer or systems engineer can obtain useful information for use in debugging a (non-executed) run of the data generator program by re-running the program with a DUMP control card inserted in the group of control cards. When the base module recognizes a DUMP control card, it takes the action described in the microfiche copy of the base module code.

The publication IBM System/360 Operating System, Programmer's Guide to Debugging, Form C28-6670, describes both types of dumps that may be obtained when one uses a DUMP card. An indicative dump is a limited dump that results from an incorrect, or from a lack of a proper, SYSABEND DD statement. For a complete storage dump, a correct SYSABEND DD statement to define a dump data set must be included in the control cards for the program.

In using the contents of a dump, you will find that register 5 contains the address of the common communication area (common area). This area contains pointers to control blocks and to tables constructed by the data generator program; and it contains parameter values that the modules of the program (1) have obtained from control cards, (2) have assigned as default values, or (3) have arrived at through computation and/or conversion. Table 5 indicates the contents of the fields corresponding to the more frequently used labels in the common area.

Certain debugging information is available as the result of processing the control card(s) preceding a DUMP control card. In the following text, the information given for a specific location of the DUMP card is in addition to any information resulting from processing any control cards that may have preceded the specified location.

1. DUMP card preceding a DSD card:
   The common area contains values for printer action.
   The open list is initialized.
   The input DCB is open.
   Much of the common area contains zeros.

2. DUMP card follows a DSD card:
   Addresses of DCBs have been determined.
   Work area for output record has been established and the area's address is located in common area.

3. DUMP card follows an FD card:
   Storage area dump contains the FD table entry relating to that FD card. If other FD cards have been processed, the corresponding FD table entries are also included in the dump.

4. DUMP card follows a CREATE card:
   Storage area dump contains tables created by the corresponding CREATE card. An updated copy of any related FD tables is also included. The most recent create table is contained in the dump and the CURCRTE field of the common communication area contains the address of this table.

The following sections contain information that summarizes or further explains material appearing in the program listings for the data generator program. This information supplements the overall view of the program as supplied by Figure 55.

- Table 5 lists the entries in the Common Communication Area. This area is used by all modules of the data generator utility program.

Table 5. Common Communication Area

Offset From Start of Common Area			
Decimal	Hex.	Label	Notes
0	0	COMMON	
0	0	PAGENO	Number of next page to be printed.
4	4	LINECT	Number of lines to print on a page.
8	8	LINECTR	Number of lines already printed on current page.
12	C	PARM	Used during invocation. Also used by Create module to save SYNAD addresses.
16	10	REPEATNO	'Quantity' value from REPEAT card.
18	12	CREATENO	'Create' value from REPEAT card.
20	14	SYSP	Data Generator SYSPRINT DCB.
116	74	SYSI	Data Generator SYSIN DCB.
216	D8	Q	Work. Area
216	D8	QFILL	
223	DF	QSIGN	
224	E0	QFILL1	
231	E7	QSIGN1	
232	E8	COUNTER	Used in scanning for continuations.
236	EC	OPENLIST	Used during DCB open processing.
236	EC	OPTBYTE1	
240	F0	OPTBYTE2	
244	F4	EXLST	
244	F4	INHDR	
245	F5	INHDR1	
248	F8	OUTHDR	
249	F9	OUTHDR1	
252	FC	INTRL	
253	FD	INTRL1	
256	100	OUTTRL	
257	101	OUTTRL1	
260	104	EXITDCB	
261	105	EXITDCB1	
264	108	TOTAL	
265	109	TOTAL1	
268	10C	EXLST1	
268	10C	EDCB1	
269	10D	EDCB2	
272	110	EXLST2	
272	110	EDCB3	
273	111	EDCB4	
276	114	EXLST3	
276	114	EDCB5	
277	115	EDCB6	
280	118	DLRECL	Default value of record length for DCB opening.
282	11A	DBLKSI	Default value of block size for DCB opening.
284	11C	DRECFM	Default value of record format for DCB opening.
288	120	LEFTOVER	
292	124	OFFSET	
296	128	LPTR	
300	12C	DCBPTR	Address of current DCB.
304	130	COMMON1	
304	130	SAVEMS	Save area for message number if more than one message.
306	132	CONCODE	Condition code to be returned to caller.

(Continued)

Offset From Start of Common Area			
Deci-mal	Hex.	Label	Notes
308	134	OUTREC	Address of output work area.
312	138	CRTABPT	Address of first create table.
316	13C	CURCRTE	Address of current create entry.
320	140	CURCRGM	Address of current create table.
324	144	CURPIC	Address where next portion of picture is to be placed in picture table.
328	148	PICCTR	Counter to keep track of length of picture remaining to be moved.
332	14C	EXITTAB	Address of first exit name table.
336	150	EXITGM	Address of current exit name table.
340	154	CUREXIT	Address of current exit name in exit name table.
344	158	DELIM	Delimiter for SYSIN records.
348	15C	RECREM	Counter for 'Record' quantity.
352	160	CURFD	Address of current FD table (in FD address table).
356	164	CUROUT	Current location in output record being constructed.
360	168	SAVE14	Contents of register 14 saved here.
364	16C	GETMLIST	Parameter list for GETMAIN macro instruction.
364	16C	GLENGTH	
368	170	ADRLIST	
372	174	IND	
372	174	GCODE	
373	175	SPOOL	
374	176	CCODE	
376	178	GCADDR	Address of last storage obtained by a GETMAIN macro instruction.
380	17C	FIRSTGMO	Address of first output DCB storage area.
384	180	CURRGMO	Address of current output DCB storage area.
388	184	LASTGMO	Address of last output DCB storage area.
392	188	FIRSTGMI	Address of first input DCB storage area.
396	18C	CURRGMI	Address of current input DCB storage area.
400	190	LASTGMI	Address of last input DCB storage area.
404	194	CONCODE	
406	196	MS	Current message number.
408	198	INBUFA1	Starting address of input work area (121 bytes).
408	198	INFILL	(10 Bytes)
418	1A2	INBUFA	Control card is read into this (111-byte) section of INBUFA1.
532	214	DDPTR	
536	218	COMMON2	
536	218	SWITCH	Start of 52-switch area.
536	218	FDCSW	FD-card continuation switch.
537	219	FDNAMESW	
538	21A	FDPCSW	FD picture-continuation switch.
539	21B	FDFMTSW	FD format switch. } only one of these
540	21C	FDPLSW	FD picture switch. } should be on.
541	21D	RANGESW	
542	21E	FILLSW	} FD card keyword indicator switches.
543	21F	REPSW	
544	220	INDEXSW	
545	221	INDNMSW	
546	222	BQUOTESW	Binary picture indicator.
547	223	PQUOTESW	Packed decimal picture indicator.
548	224	EQUCTESW	EBCDIC (character-string) picture indicator.
549	225	FDSW	
550	226	DSDSW	
551	227	NOGOSW	'No-execution' switch. (Indicates syntax checking only.)
552	228	CREATESW	First CREATE-card switch.
553	229	DSDCSW	DSD continuation card switch.
554	22A	CRCSW	CREATE continuation card switch.
555	22B	EXITSW	Indicator that an initial exit-name table exists.
556	22C	EODSTOP	Switch to stop generation on input end-of-data.

(Continued)

Offset From Start of Common Area			
Deci-mal	Hex.	Label	Notes
557	22D	DSDNULSW	Not used.
558	22E	DSDORGSW	
559	22F	DSDDDSW	
560	230	CRTBLK	Indicator for a blank CREATE card.
561	231	NAMECSW	Name continuation switch.
562	232	PICCSW	CREATE card picture-continuation switch.
563	233	BUFPSW	
564	234	ENDSW	
565	235	COMCSW	Comments continue switch.
566	236	FLAGSW	
567	237	PAGESW	
568	238	EPSW	
569	239	SYSISW	
570	23A	SYSPSW	
571	23B	OLDNEWSW	Input/output data set indicator.
572	23C	FLUSHSW	
573	23D	FLUSHSW1	
574	23E	DSDOSW	
575	23F	DSDISW	
576	240	QUANSW	CREATE card 'quantity' switch.
577	241	PARENSW	Indicates detection of a left parenthesis.
578	242	REPEATSW	Used to test if a REPEAT card remains to be processed.
579	243	SYSINEOD	Address of end of SYSIN data.
588	24C	FDPLGTH	FD-picture length.
592	250	SGCADDR	Save Area for address of storage obtained by GETMAIN macro instruction.
596	254	FDPTR	Address of current FD-table entry.
600	258	FDPTR1	Address of first FD table.
604	25C	FDPTR2	Address of current FD table.
608	260	COMMON3	
608	260	FDCTR	Count of number of entries in current FD table.
612	264	LREMAIN	Length of FD picture remaining at end of scanning an FD card.
616	268	COMPCTR	
620	26C	LMOVED	
624	270	U	Current random number value.
628	274	PICEND	Location of end of picture in output record.
632	278	CURFDGM	Address of current FD-address table.
636	27C	SWTCH	
636	27C	FIRSTSW	
637	27D	FRSTSW	
638	27E	STOPSW	
640	280	COPYVAL	Value of COPY parameter from a CREATE card.
644	284	COPYFD	Pointer to FD address used in copying a 'name' group.
648	288	COPYFDGM	Address of FD-address table.
652	28C	NAMCTR	Number of FD addresses to be copied for a 'name' group.
654	28E	NAMCTR1	Counter used in copying a 'name' group.
656	290	INRECSZ	Logical record length of an input record.
658	292	OUTRECSZ	Logical record length of an output record.
660	294	INRECFM	Input record format.
661	295	RECOFFST	Offset to start of data in output record.
662	296	OUTRECFM	Ouput record format.
664	298	PICBASE	Address of start of picture table.
668	29C	MESSAGE	

Table 6 lists the fields of the Data Control Block (DCB). The labels, as given in this dummy section, may vary in name from the levels for the DCB fields as given in the System Control Blocks publication. However, the offsets from zero correspond in meaning with those given in the System Control Blocks publication.

• Table 6. Data Control Block

OFFSET FROM START OF DCB		
DECIMAL	HEX	LABEL
0	0	DCBD
0	0	FILL
17	11	DEVT
18	12	FILL1
26	1A	DSORG1
26	1A	DSORG
28	1C	FILLER
28	1C	IOBAD
32	20	BFTEK
33	21	EODAD
36	24	RECFM
37	25	EXLIST
40	28	DDNAME
40	28	DEBAD
40	28	IFLGS
48	30	GETAD
48	30	OFLGS
49	31	OFLGS1
50	32	MACRF
52	34	FILL2
56	38	SYNAD
60	3C	CIND
62	3E	BLKSI
64	40	FILL3
82	52	LRECL
84	54	FILL4
256	100	NEXTDCB
260	104	DDNAME1
268	10C	EODSW
269	10D	DCBSW1
270	10E	DCBSW2
271	10F	DCBSW3

Table 7 lists the defined constants (DCs) that are used by the various modules of the data generator program.

• Table 7. Defined Constants for Modules of the Data Generator Program

Label	Base	Clean-up	FD Analysis	FD Table	Create Analysis	Create
'C1				C'FX'		
'C2				C'SL'		
'C3				C'TL'		
'C4				C'SR'		
'C5			C'NAME='	C'TR'	C'QUANTITY='	
'C6	C'LINECT='		C'LENGTH='		C'NAME='	C' (see 'C30 for base)
'C7			C'STARTLOC='		C'PICTURE='	
'C8			C'PICTURE='	C'RP'	C'FILL='	C'RA'
'C9			C'FORMAT='	C'RO'	C'INPUT='	C'ZD'
'C10			C'ACTION='	C'WV'	C'EXIT='	C'PD'
'C11			C'FILL='			C'BI'
'C12			C'CYCLE='	C'BI'		
'C13	C'END'		C'RANGE='	C'PD'		
'C14	C'FD'		C'CHARACTER='	C'AL'	C'COPY='	
'C15	C'CREATE'		C'SIGN='	C'AN'		
'C16	C'REPEAT'		C'INDEX='	C'CO'	C'P'''	
'C17	C'DUMP'		C'REPLACE='		C'B'''	
'C18	C'OUTPUT=('					
'C19	C'INPUT=('					
'C20				C'RA'		
'C22				C'ZD'		
'C24					C'SYSIN	
'C25	C'QUANTITY='				C'$$$E'	
'C26	C'CREATE='					
'C28			C'B'''			
'C29			C'P'''			
'C30	C' IEB729I PERMANENT I/O ERROR'					
'D1	H'256'		H'-64'	H'-1'	H'-1'	F'0'
'D2	F'123456'		H'32767'	H'256'	F'0'	H'-1'
'D3	F'65535'		H'256'	H'2'	H'256'	H'8'
'D4				H'-2'	H'1'	H'4'
'D5				H'1'	H'16'	F'524291'
'D6				H'-4'	F'1'	H'-4'
'D7				H'-3'		H'256'
'D8				H'32767'		
'X2	X'F0F0F0F1'					
'X5	X'000B'					X'0000'
'X7	X'0030'			X'0000'		
'X8	X'0000000000000000'					
'X9	X'0028'					
'X17				X'001A'		
'X18				X'0024'		
'X19				X'003F'		
'X23	X5000'					
'X26					X'02147483647F'	
'X30			X'0000'			
'X31			X'0000000000000000'	X'0004'		
'X32			X'000002147483647F'	X'0003'		
'X36				X'000002147483647F'		
FOXZEROS		X'F0F0F0F0F0F0F0F0'			X'F0F0F0F0F0F0F0F0'	
NO						X'00'
OFF		X'00'	X'00'	X'00'	X'00'	X'00'
ON		X'FF'	X'FF'	X'FF'	X'FF'	X'FF'
ONE						FL4'0'
'SIZ001'	A('DATEND-'DATD)					
'T1					F'0'	F'0'
'TEMP4			F'0'	F'0'	F'0'	F'0'
YES						X'FF'

Table 8 lists the equated symbols (EQUs) that are used by the various modules of the data generator program.

• Table 8. Equated Symbols for Modules of the Data Generator Program

Label	Module					
	Base	Clean-up	FD Analysis	FD Table	Create Analysis	Create
A2J7 F6D3 'L RET '9CE		'EL01 1	1	F8B4 1	1 'EL01	1 'EL01
'9D4 '9D7 '9EC '9EF '9E2	'EL01 A2B3		F4G11		CARDSCAN	A7A12
'9E9 '9FC '9FD '9F0 '9F2	PDDNAMER PDDNAMER		'EL02	'EL02	A6C5	A7A18
'9F3 '9F4 '9F5 '9F7	LABEL1 SCAN1			F9E5	A6A11 KEYSCAN	

Tables and Work Areas Used by Modules of Data Generator Program

Table 9 is a grid indicating the modules that establish, use, and modify the major work areas and information tables of the data generator program. Mnemonic names for the tables or area are placed in parentheses and correspond with the names given in the module cross-references on microfiche listings.

• **Table 9. Data Generator Modules Information Tables and Areas**

Modules	BASE	FDANL	FDTAB	CRANL	CREAT	MSG	CLNUP	
Table/Area	Module Action Code: B = Module gets storage for, and/or enters data into. U = Module uses or modifies the area.							
Common Comm. Area (COMMON)	B	U	U	U	U	U	U	
Create Table (CRTAB)				B				
Exit Name Table (EXITTAB)				B	U			
FD Table (FDTBL)		B	B		U			
FD Address Table (FDADTAB)				B	U			
Input Buffer (INBUF)	B, U	U		U				
Input/Output DCB	B				U		U*	
Message Table (MESSAGE)						U		
Message Pointer Table (MSGPTR)	B					U		
Create Picture Table (CRPICT)				B	U			

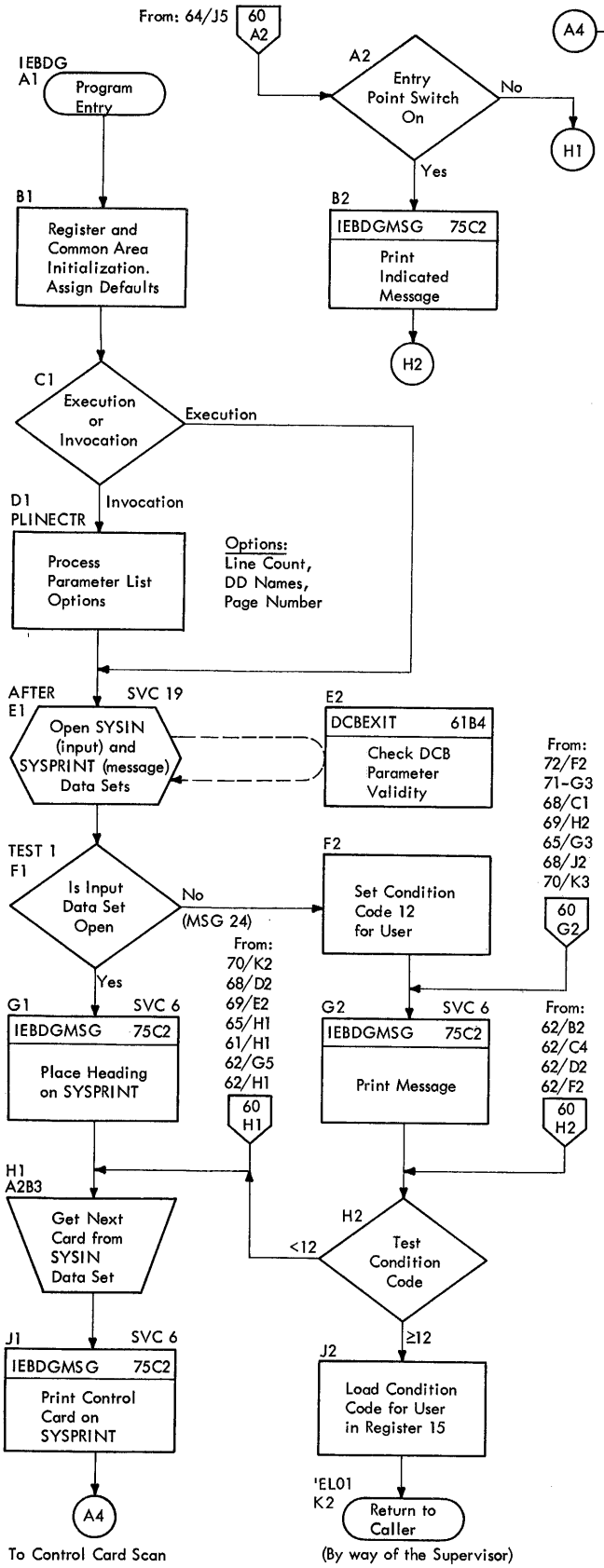
* Closes and releases storage for:

Table 10 contains a summary of the input and output information to be found on microfiche listings of the data generator program.

• Table 10. Module Inputs and Outputs

INPUTS	OUTPUTS
BASE MODULE	
Data Generator Control Cards. DD cards for all data sets used. Either: Parameter List Address for invocation or Job Control Language EXEC card parameters.	Reg. 5, pointing to a common communication area. Message indicator in the common area. Reg. 9, pointing to a data generator control card operation field in an input Buffer--this is for output to an analysis module.
CLEAN-UP MODULE	
Reg. 5 and other pointers (in the communication area) indicating, respectively, the addresses of the communication area and control tables.	DCB storage areas and associated buffer areas are released to the system.
MESSAGE MODULE	
Reg. 5, pointing to the communication area. Message number indicator. Actual or default values for linecount and page number. Output DCB name. Indicators for: Output DCB open or not. Channel 9 carriage control. Channel 12 carriage control.	Heading and paging information. Program messages. Error return codes. Control card images.
FD ANALYSIS MODULE	
Reg. 9, pointing to a control card image in an input buffer.	One or more FD tables...520 bytes each. FD table entry with some parameter values. Temporary storage with a picture or format pattern.
FD TABLE MODULE	
FD table entry. Reg. 5, pointing to communication area. SGCADDR pointer to picture temporary storage. Switch indicating picture type, if any.	Completed FD table entry...64 bytes.
CREATE ANALYSIS MODULE	
Reg. 5, pointing to communication area. Reg. 9, pointing to control card image in input buffer.	One or more create tables..512 bytes each. Create table entry...28 bytes. Picture table...(L + 6)* bytes. FD address table(s)...88 bytes each. Exit name table(s)...72 bytes each. *Note: See Figure 61 for definition of L.
CREATE MODULE	
Reg. 5, pointing to communication area. Create tables(s). Picture table(s). FD address table(s). Exit name table(s).	Records written on an output device as specified by DD name on a DSD control card.

• Chart 60. IEBDG Base Module (Part 1 of 3)



A4 → CONTROL CARD SCAN

For a given control card type, check for the initial card, a continuation card, and a comments card.

The first control card of a set of control cards for this utility program must be a DSD card. In the following table, the indicated switches are tested, or the indicated tests are preformed. The action taken depends on whether a switch is "on" (= 1) or "off" (= 0), or whether a test result is "yes" or "no".

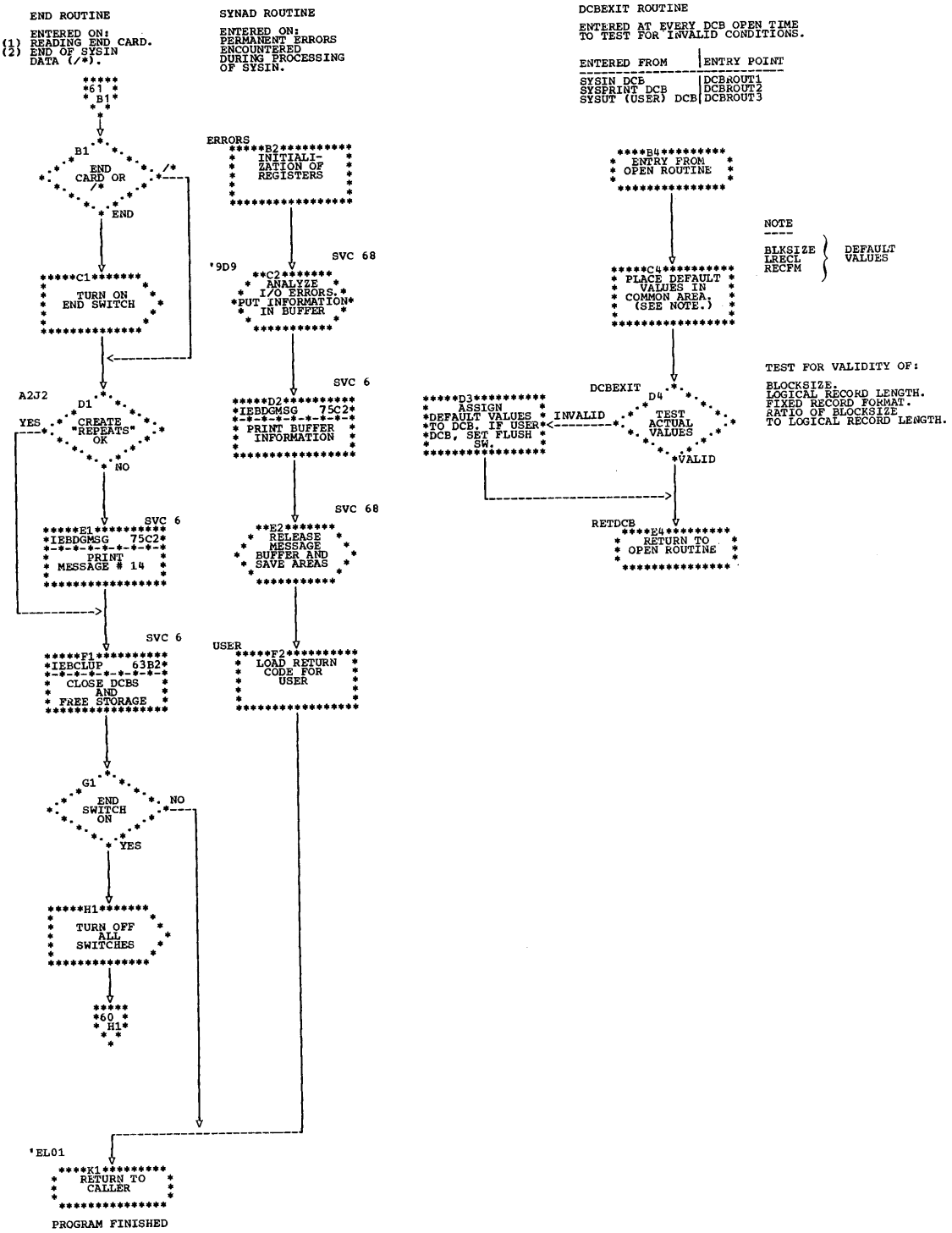
Switch or Test No.	Switch or Test Name	"On" or "Yes" Action (*)	"Off" or "No" Action
SW 1	Comments Continue SW	Test SW 2 or go to Chart 60, Box H1.	Test SW 2
SW 2	FD Continue SW	Go to Chart 64, Box A1	Test SW 3
SW 3	Create Continue SW	Go to Chart 68, Box A2	Test Sw 4
SW 4	DSD Continue SW	Go to Chart 62, Box C3	Test SW 5
SW 5	FD Picture Continue SW	Go to Chart 64, Box A1	Test SW 6
SW 6	Create Picture Continue SW	Go to Chart 68, Box A2	Test for DSD Control Card
Test 1	DSD Control Card	Go to Chart 62, Box B3	Test for FD Control Card
Test 2	FD Control Card	Go to Chart 64, Box A1.	Test for CREATE Control Card
Test 3	CREATE Control Card	Go to Chart 68, Box A1	Test for REPEAT Control Card
Test 4	REPEAT Control Card	Go to Chart 62, Box B1	Test for END Control Card
Test 5	END Control Card	Go to Chart 61, Box B1	Test for DUMP Control Card
Test 6	DUMP Control Card	Terminate the Job	Return to Supervisor

To Point Indicated in Above Table.

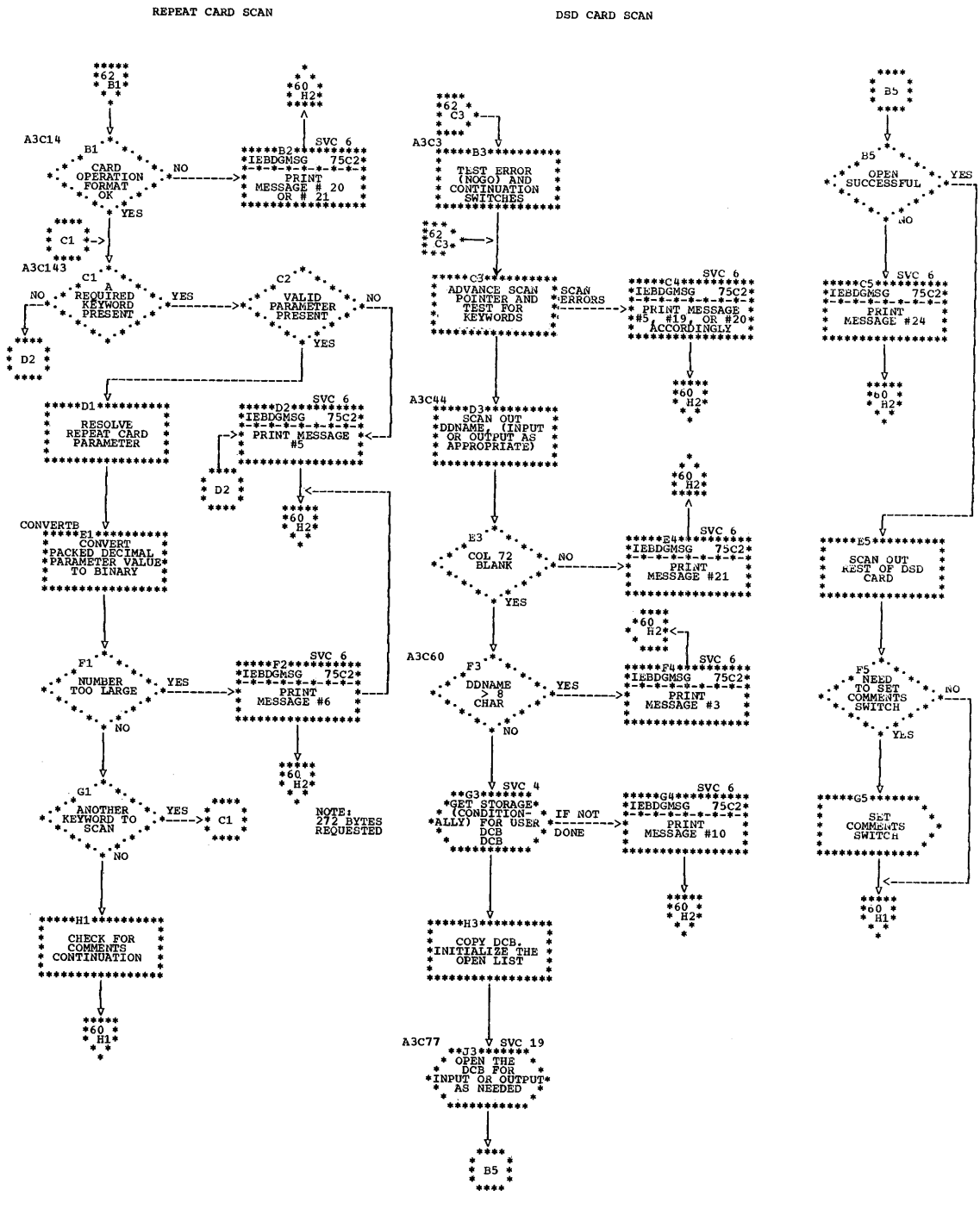
(*) Chart Designations:

68-72 Create Analysis Module
60-62 Base Module
64,65 FD Analysis Module
75 Message Module

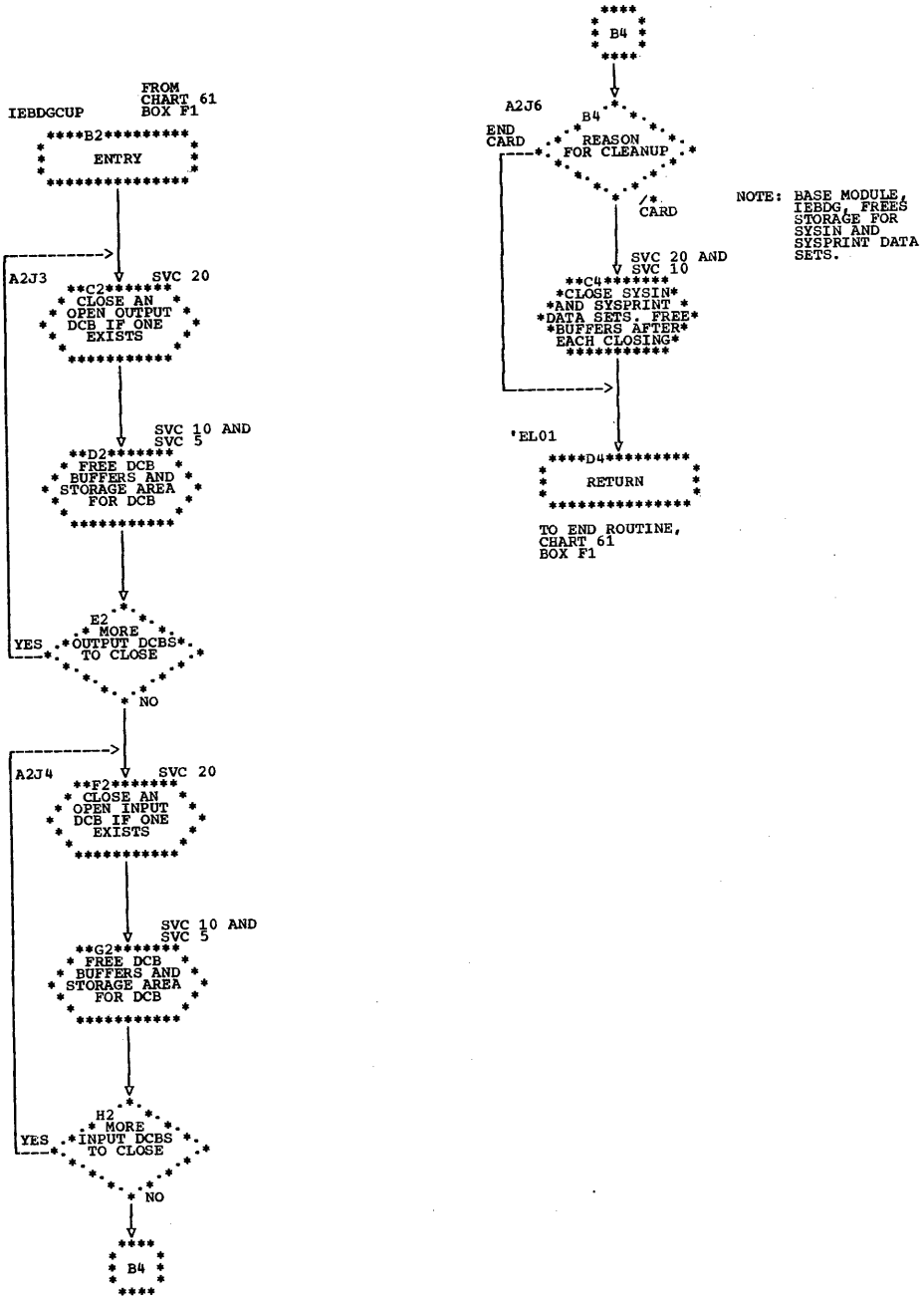
• Chart 61. IEBDG Base Module (Part 2 of 3)



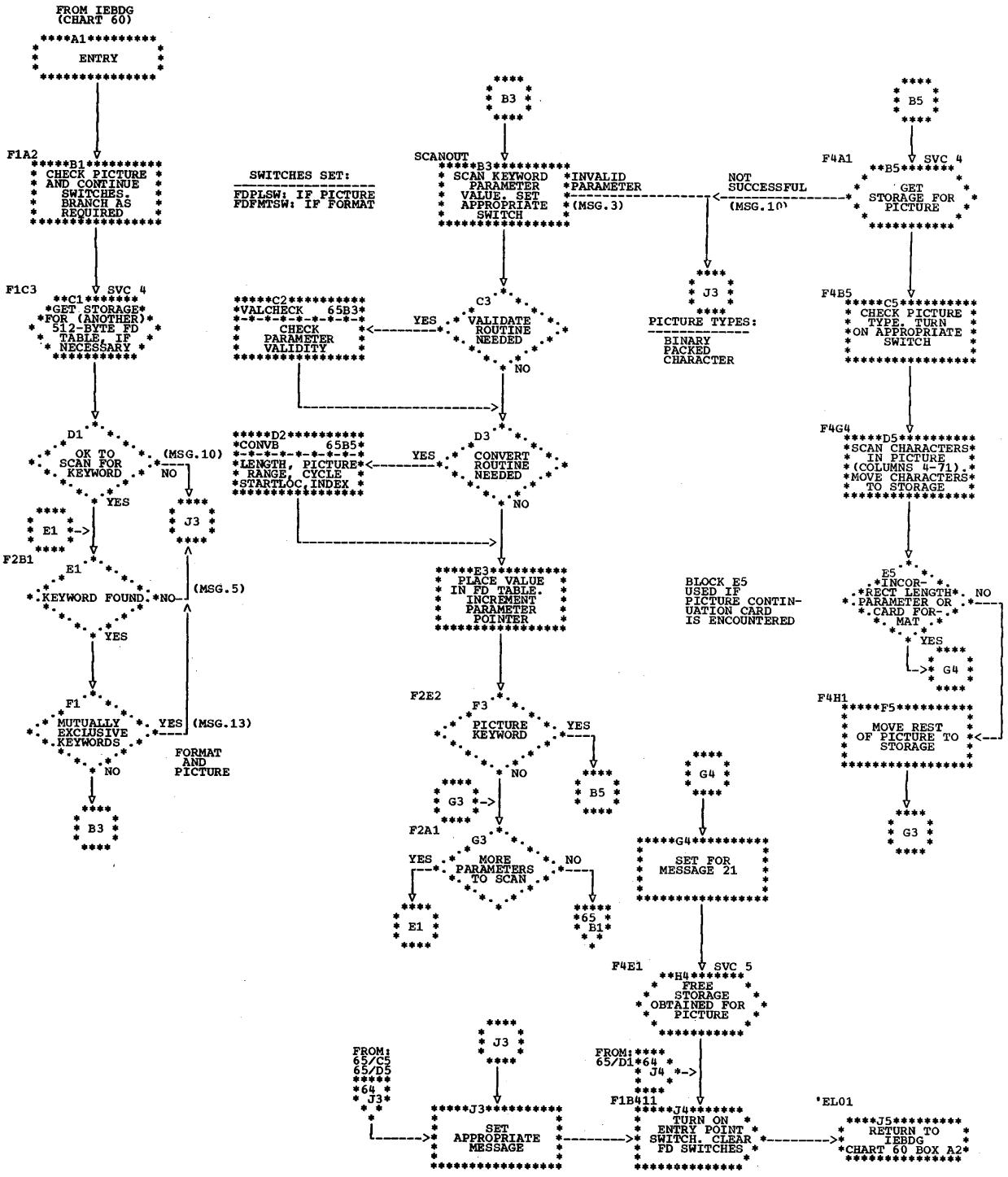
• Chart 62. IEBDG Base Module (Part 3 of 3)



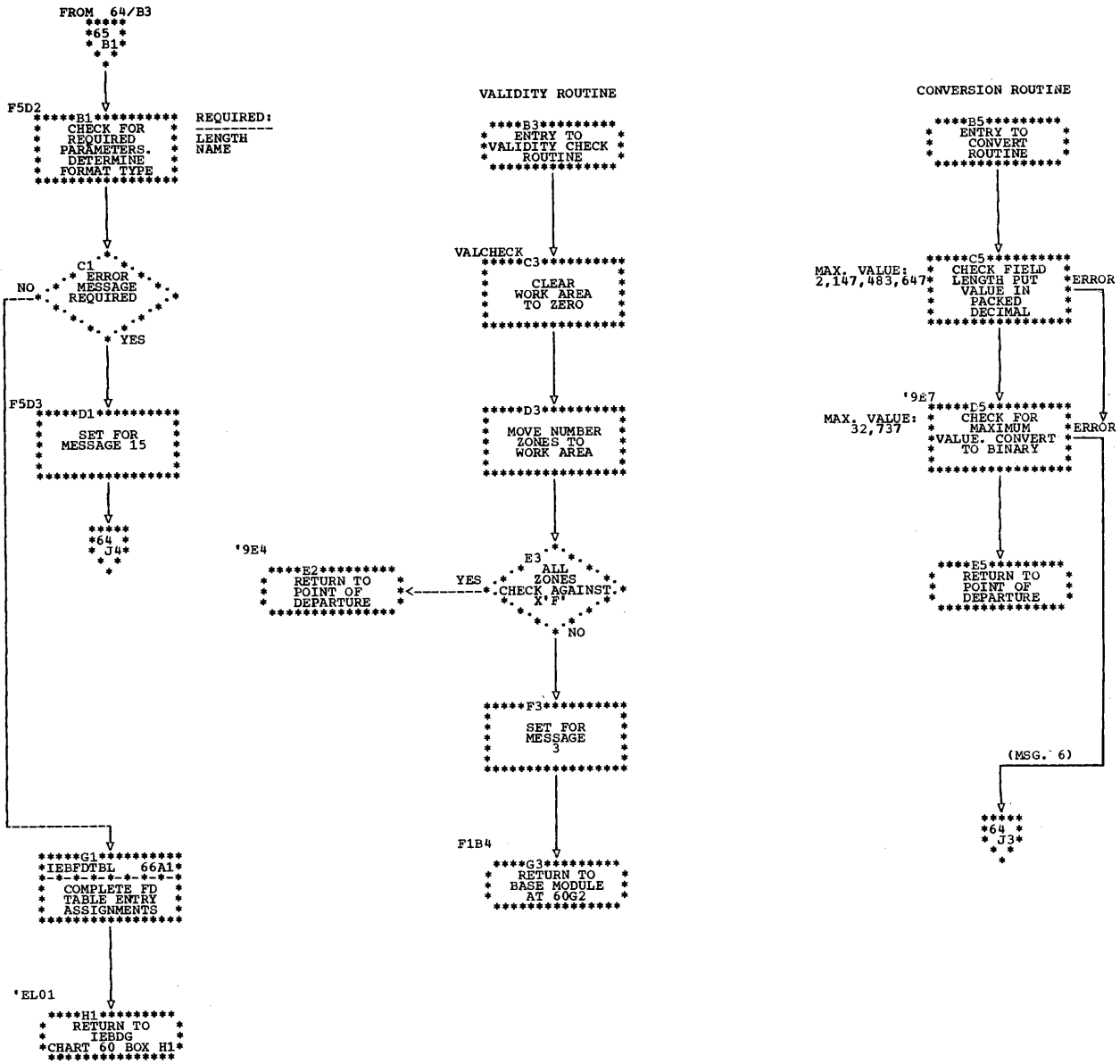
• Chart 63. IEBDG Clean-Up Module, IEEDGCUP



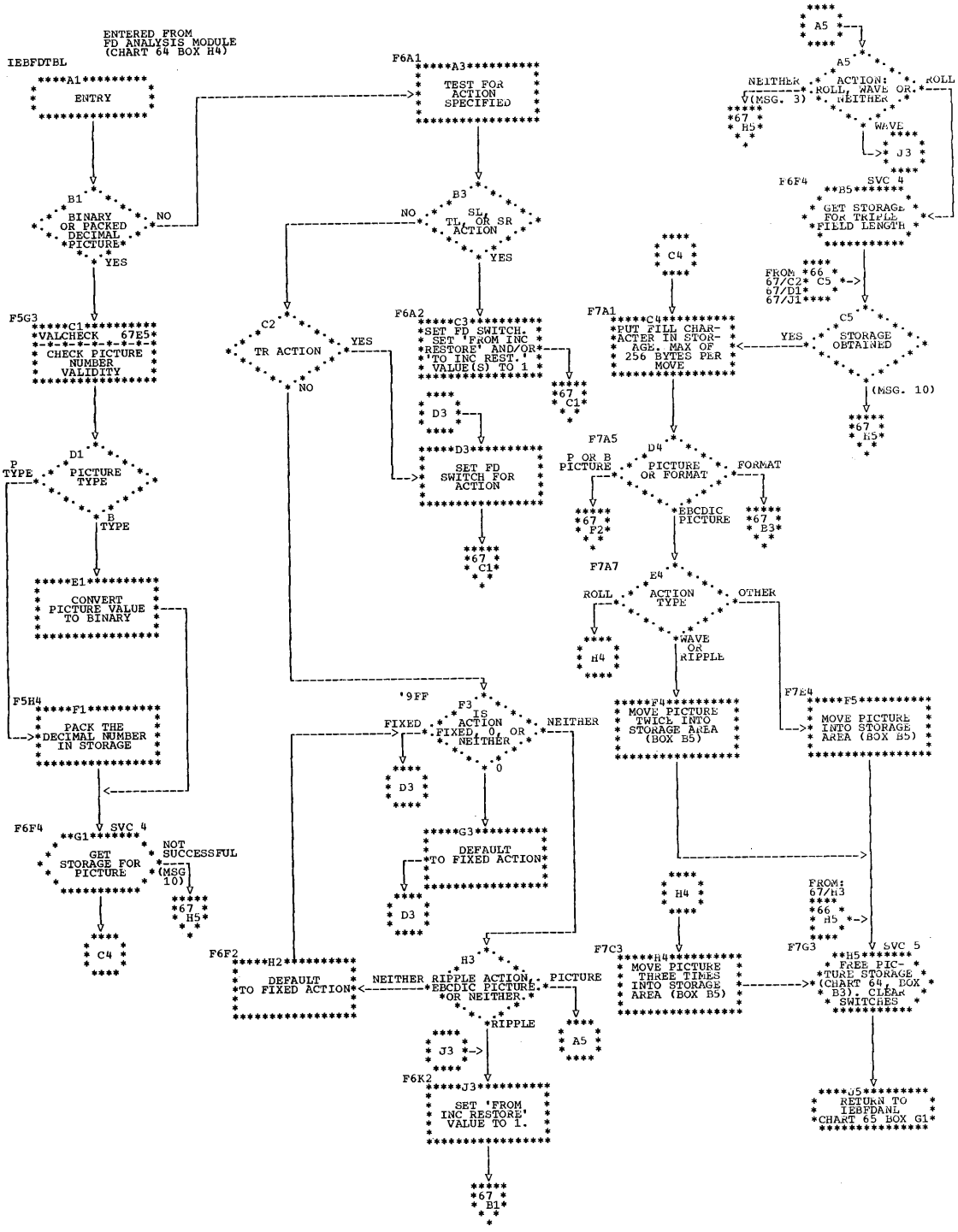
• Chart 64. IEBDG FD-Analysis Module, IEBFDANL (Part 1 of 2)



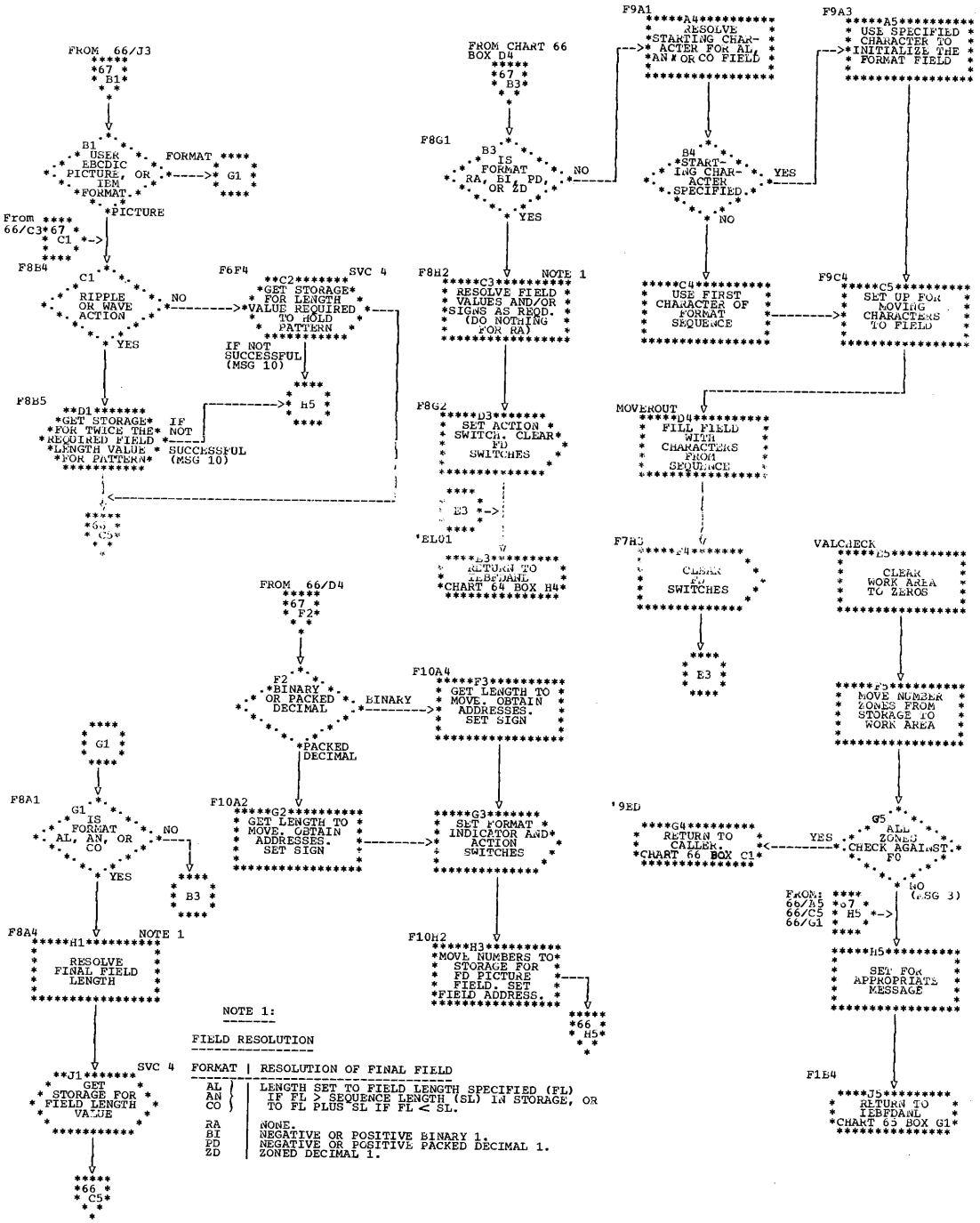
• Chart 65. IEBDG FD-Analysis Module, IEBFDANL (Part 2 of 2)



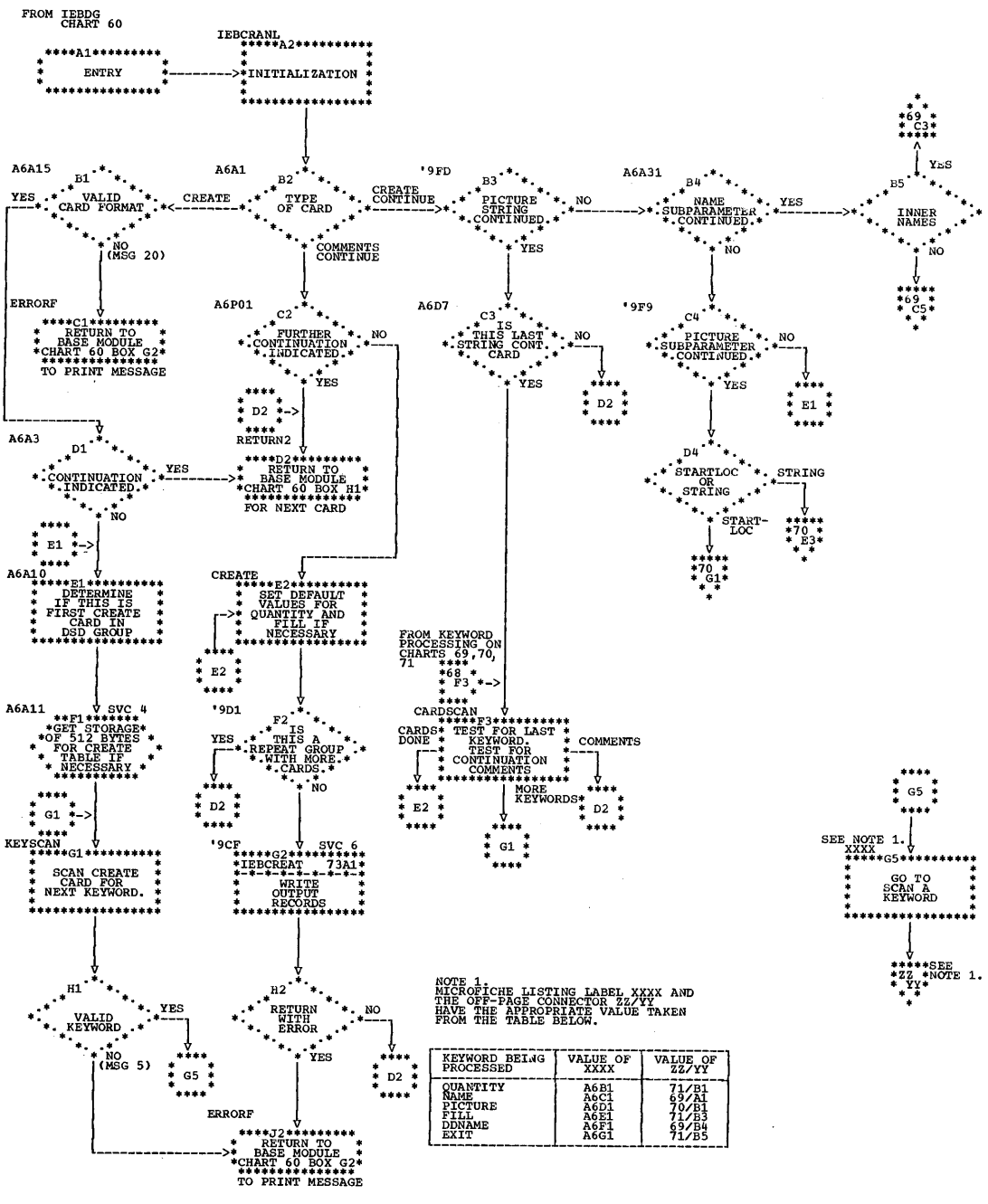
• Chart 66. IEBDG FD-Table Module, IEBFDTBL (Part 1 of 2)



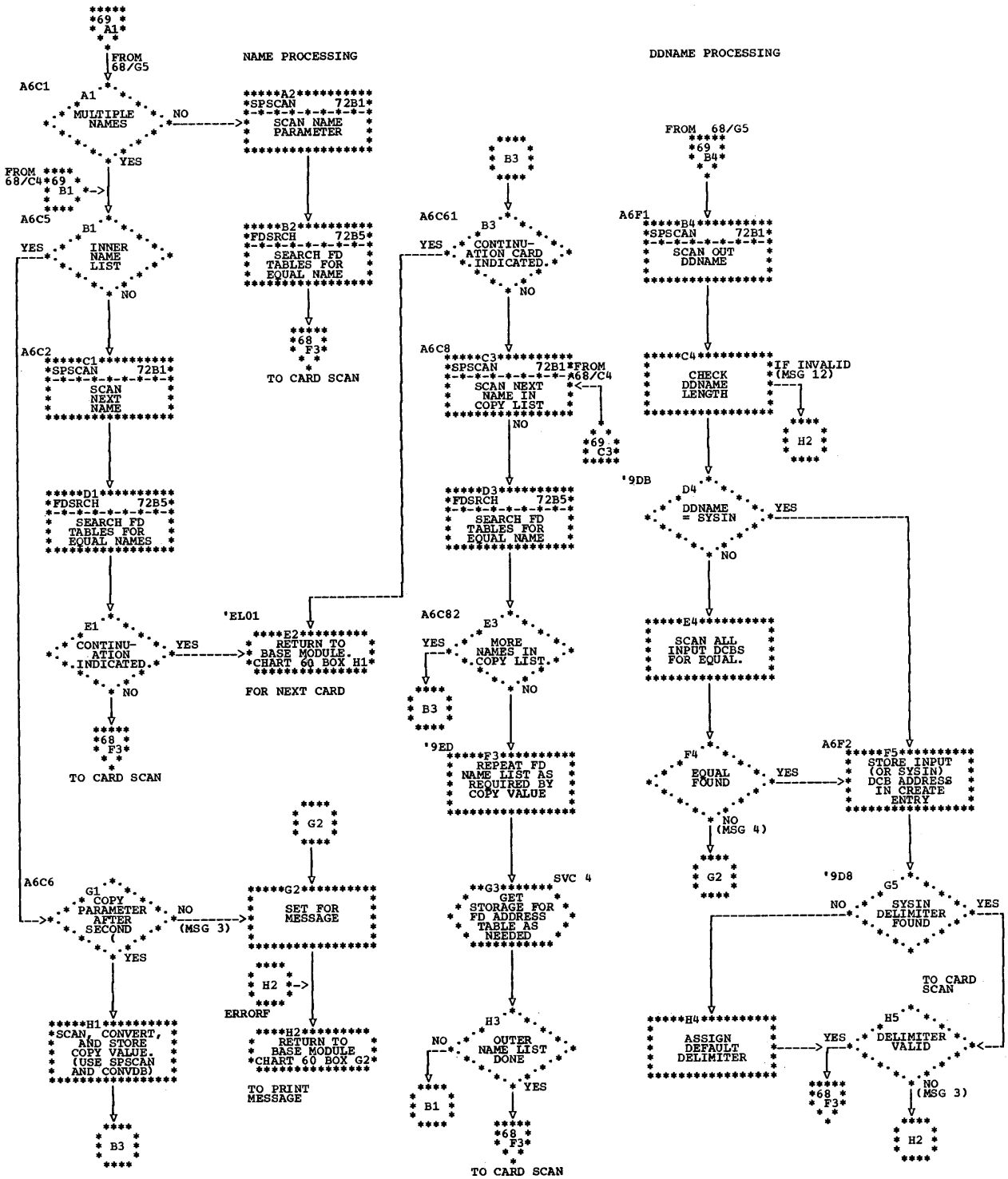
• Chart 67. IEBDG FD-Table Module, IEBFDTBL (Part 2 of 2)



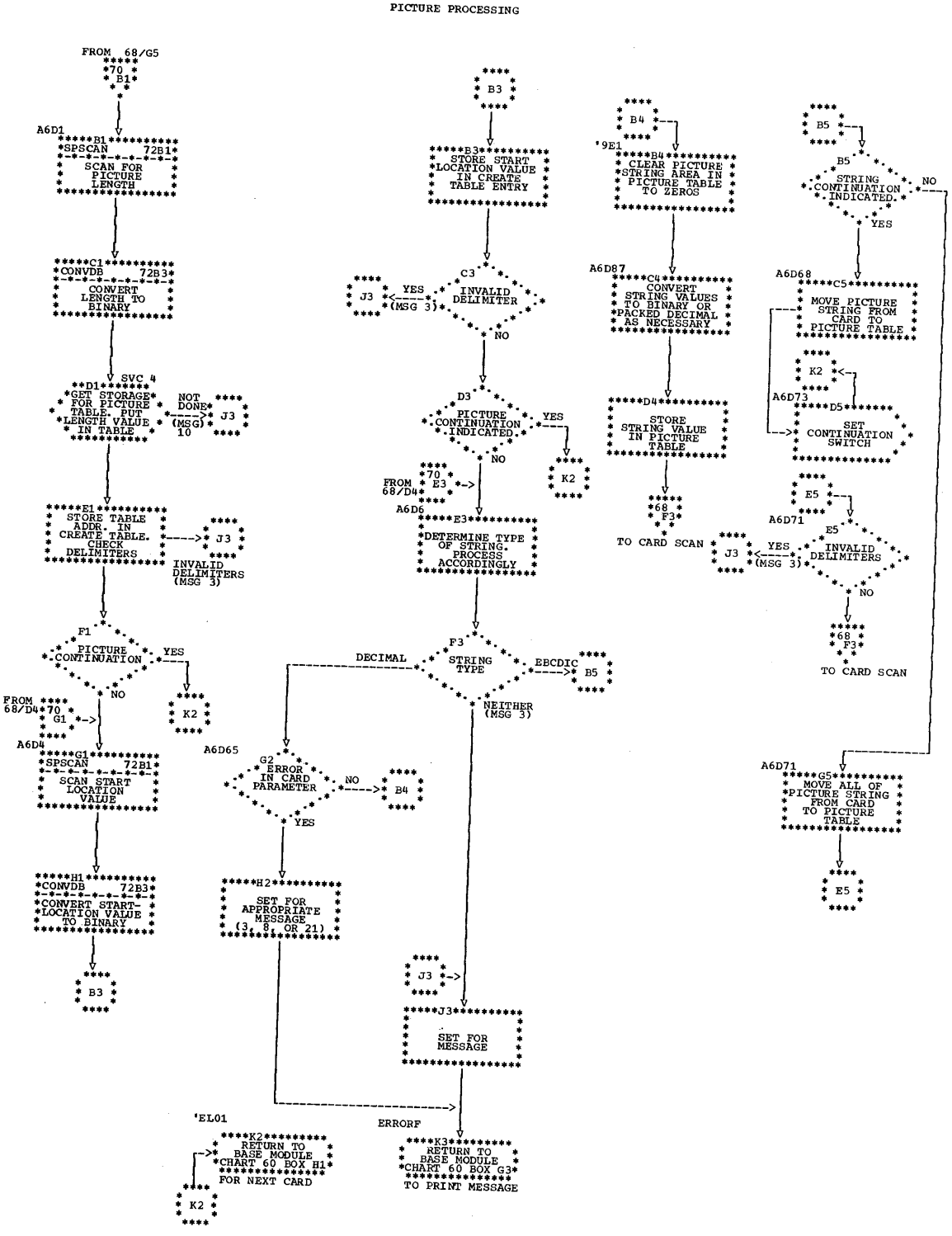
•Chart 68. IEBDG Create Analysis Module, IEBCRANL (Part 1 of 5)



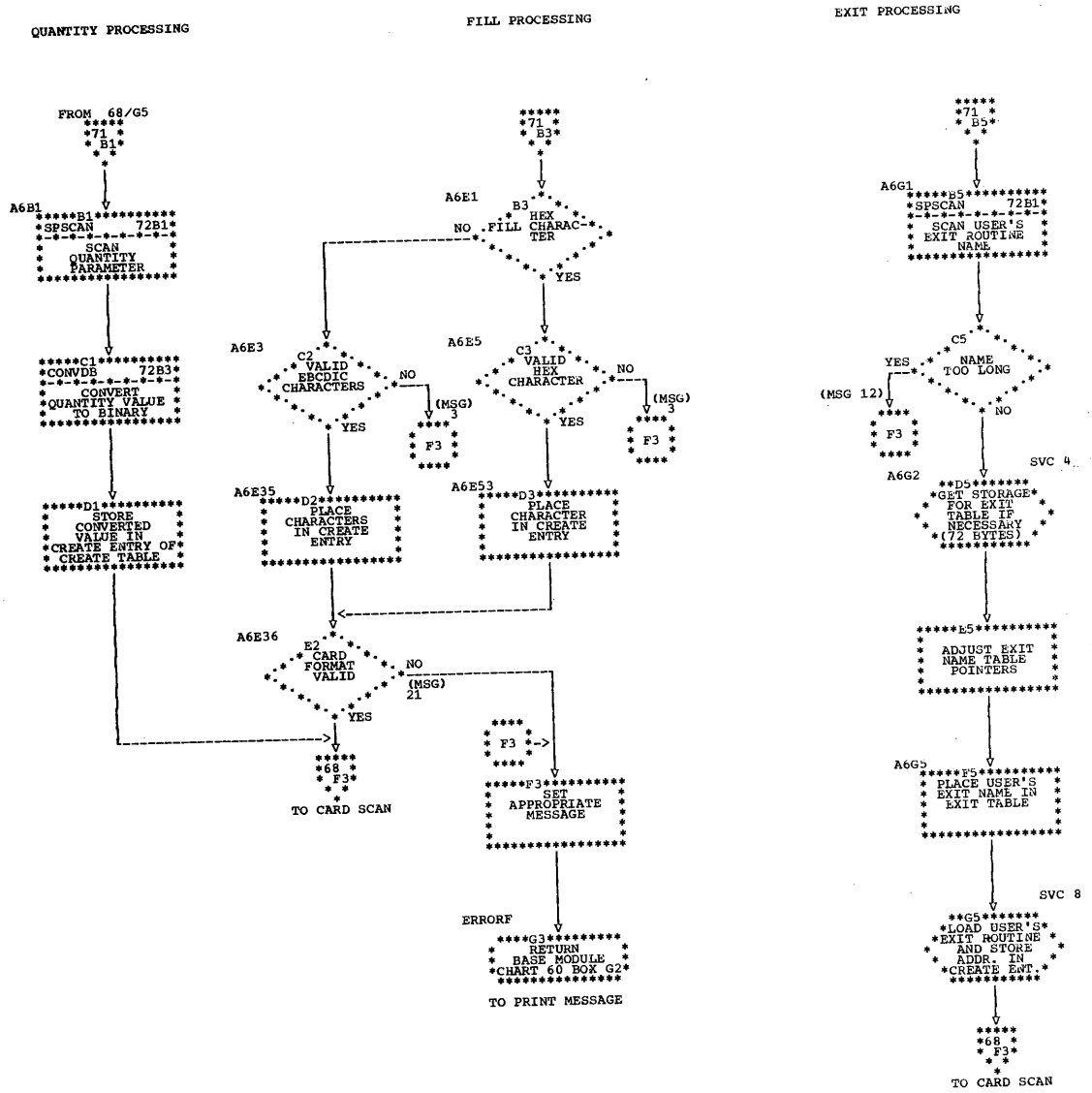
•Chart 69. IEBDG Create Analysis Module, IEBCRANL (Part 2 of 5)



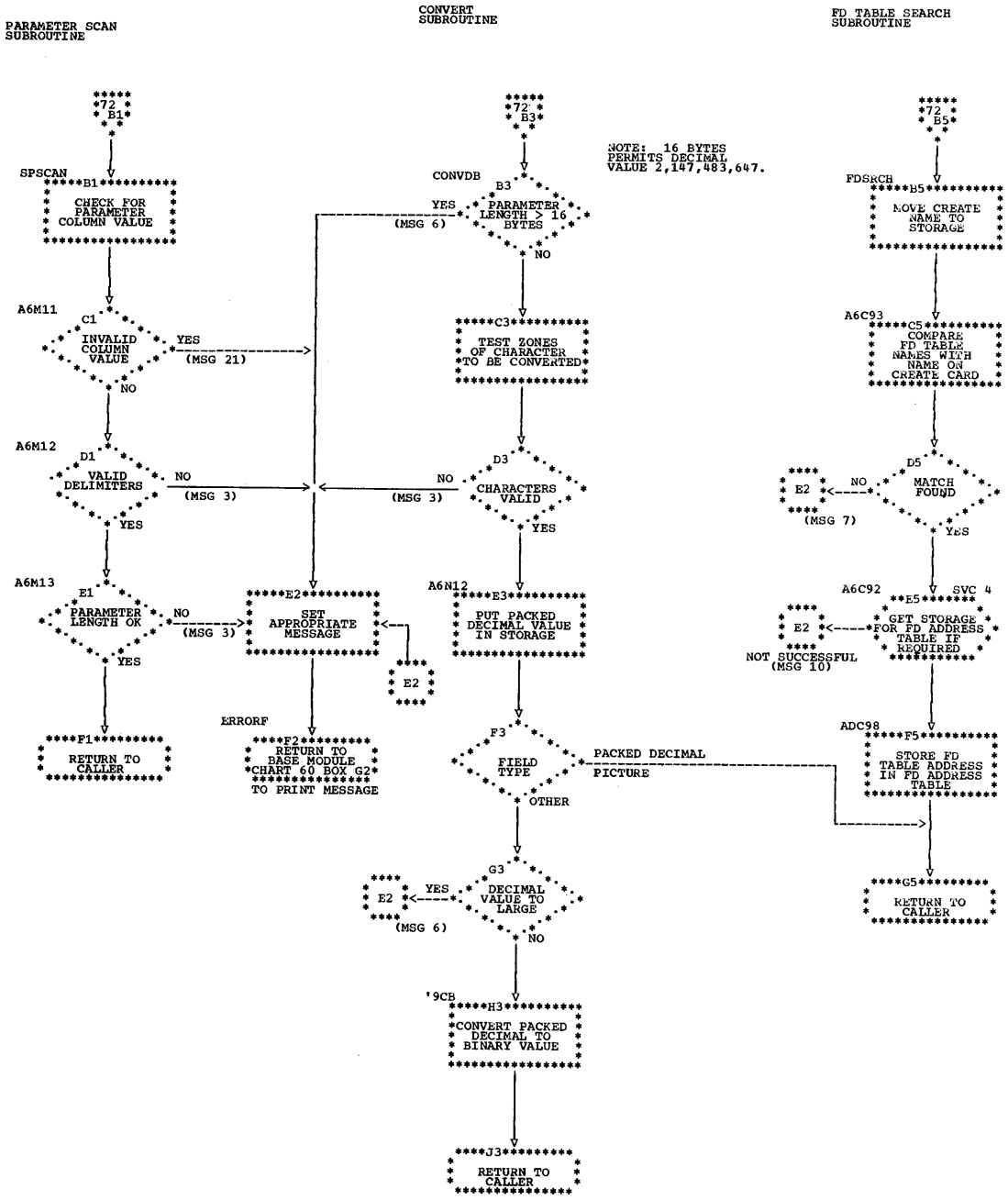
• Chart 70. IEBDG Create Analysis Module, IEBCRANL (Part 3 of 5)



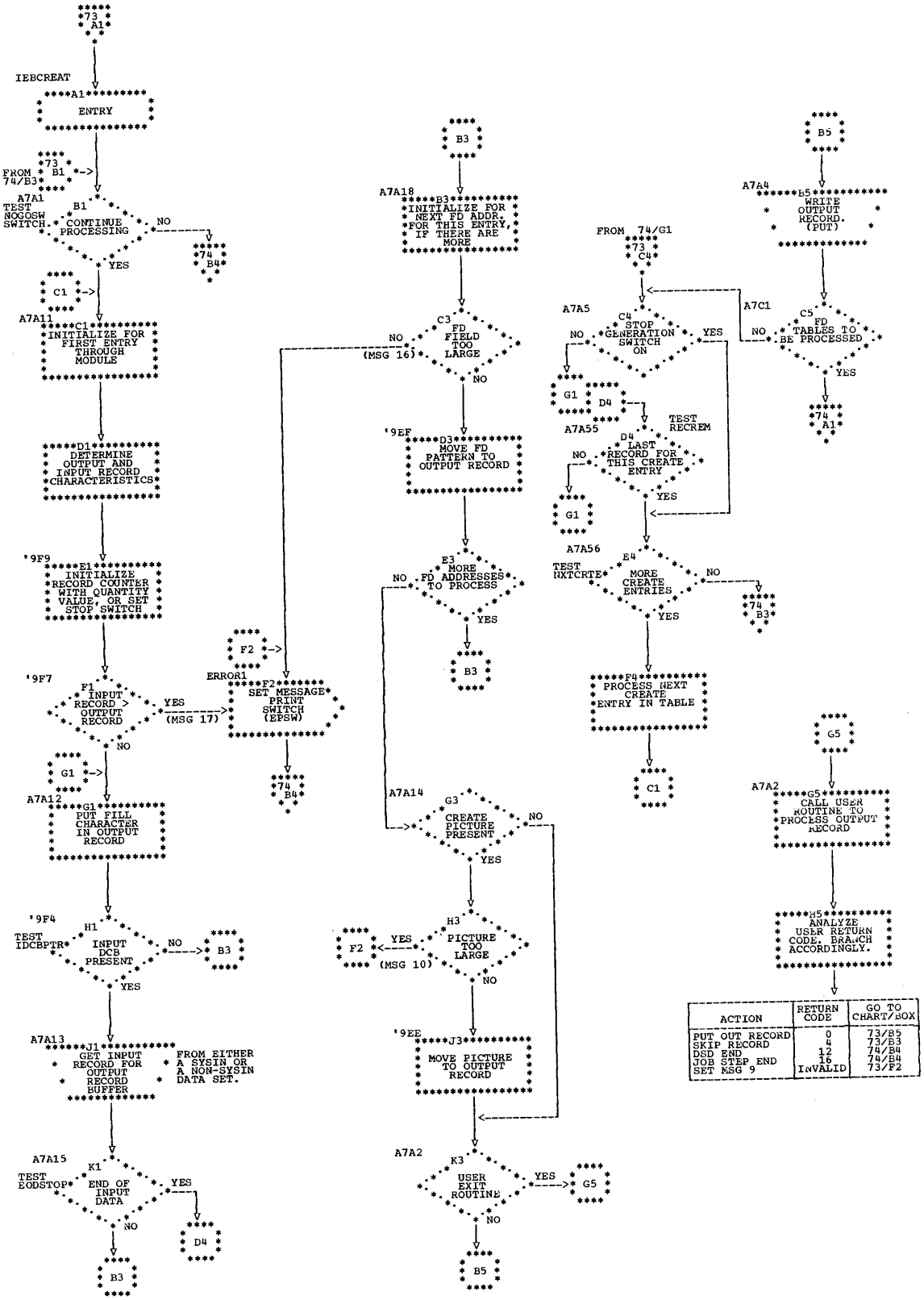
• Chart 71. IEBDG Create Analysis Module, IEBCRANL (Part 4 of 5)



• Chart 72. IEBDG Create Analysis Module, IEBCRANL (Part 5 of 5)

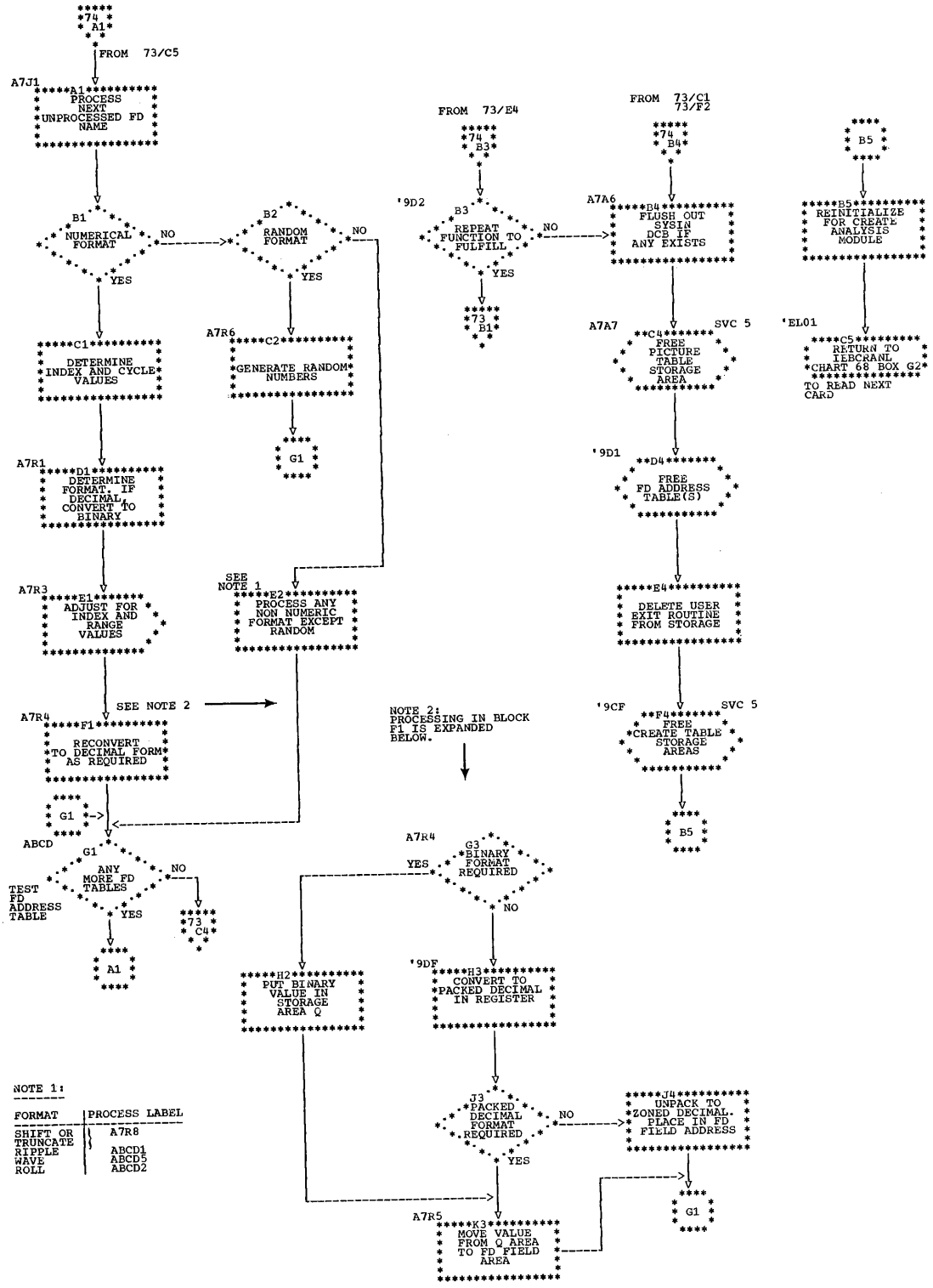


• Chart 73. IEBDG Create Module, IEBCREAT (Part 1 of 2)

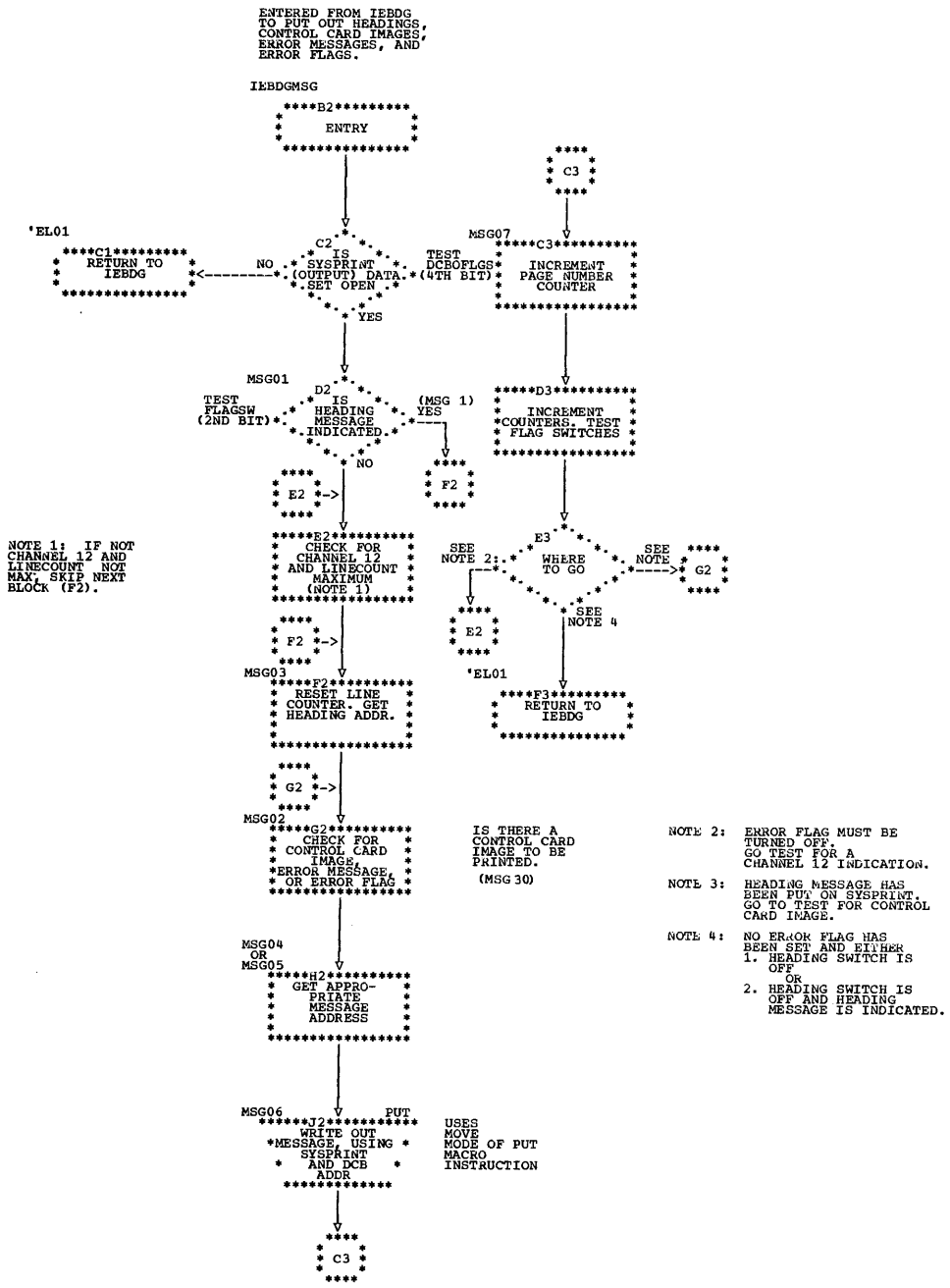


ACTION	RETURN CODE	GO TO CHART/BOX
PUT OUT RECORD	0	73/B5
SKIP RECORD	4	73/B3
DSD END	12	74/B4
JOB STEP END	16	74/B4
SET MSG 9	INVALID	73/F2

• Chart 74. IEBDG Create Module, IEBCREAT (Part 2 of 2)



• Chart 75. IEBDG Message Module, IEBDGMSG



Independent utility programs are executed outside and in support of IBM System/360 Operating System. They are:

- IBCDASDI, which initializes a direct access volume and obtains alternate tracks on initialized disk storage.
- IBCDMPRS (dump-restore), which dumps and restores the data contents of a direct access volume.
- IBCRCVRP (recover-replace), which recovers data from a track on direct access storage, replaces defective records with data supplied by the user, and writes the composite data on an operative track of the original volume.

Independent utilities are discussed in four parts:

- Supervisory Routines of the Independent Utilities
- IBCDASDI
- IBCDMPRS
- IBCRCVRP

Supervisory Routines of the Independent Utilities

The independent utility programs contain copies of supervisory routines to check the input device, read control statements, analyze control statements, check volume labels, print diagnostic messages, type diagnostic messages to the operator, control I/O, and analyze I/O interruptions.

CHECKING THE INPUT DEVICE

The entry point to this routine is CKINPUT. The routine is entered immediately after IBCDASDI, IBCDMPRS, or IBCRCVRP is loaded. The program assumes a WAIT state (by means of LPSW) until the input device is defined by the operator. The operator then enters a code by means of typewriter or console. This routine then checks the code to verify that the input device is 1442, 1402, 2400, or 2540 (or 1052 for IBCRCVRP) and that the channel number is not greater than six. If these conditions are satisfied, the appropriate UCB is selected and control is given to the control statement analysis routine at location CLRSCAN. If an error is detected in the coded information, an error message is printed or displayed and the WAIT state is entered with E's displayed on the console lights.

DATA INPUT ROUTINE

The entry point to this routine is SYSIN. Linkage to the routine is by a BAL LINK15, SYSIN. Register GR2 contains the address of the calling routine's buffer. This subroutine stores the buffer address in the channel command word SYICCW, sets a read command and links to subroutine STARTIO via a BAL LINK9, STARTIO. Reading is then performed by the defined input device. When control is returned to this routine, it in turn returns control to the calling routine via a BR LINK15.

CONTROL STATEMENT ANALYSIS

The entry point to this routine is CLRSCAN. Housekeeping functions are first performed on program switches and buffer areas required by the routine. This routine then links to the control statement scan routine at RDCARD. RDCARD returns a pointer to a field and the length of the field in registers SCANADR and LENGTH, respectively, and an indication of the field type in location SWITCHRD. SWITCHRD is a one-byte switch with the following settings:

Bit	Value	Meaning
0	1	control statement error
1	1	bypass
3	1	first control statement has been read
4	1	operator found
5	1	keyword found
6	1	parameter found

Validity checks are then performed on the scanned data. If an error is detected in the input data, an attempt is made to print a message on the defined message output device. If the message output device is not defined, an attempt is made to issue the message using the Write to Operator routine. If neither device is defined, the WAIT state is entered. If the message is successfully issued, the WAIT state is entered, and the program must be reinitiated and the corrected statement submitted.

Following completion of control statement analysis, control is given to the appropriate routine in IBCDASDI, IBCDMPRS, or IBCRCVRP.

VOLUME LABEL CHECKING

The IBCDASDI program compares the volume serial number of the object volume to that specified by the VOLID parameter, if both numbers are present. If the VOLID parameter specifies SCRATCH, no comparison occurs. If a serial number is specified, and it is not equal to that in the volume label, or if the volume label is not present, this routine causes an appropriate message to be printed and terminates the program.

The IBCDMPRS program compares the volume serial number of the TO volume to that specified by the VOLID parameter, if both numbers are present. If the TO device is tape, and there is no volume label present, there must be a tape mark at load point, or SCRATCH must be specified, in order for the program to continue. If the TO device is tape and a volume label is present and VOLID does not specify SCRATCH, the volume serial number in the label must equal that specified by VOLID in order for the program to continue. If the TO device is direct access storage, VOLID must be specified and an equal comparison of serial numbers must occur in order for the program to continue.

The IBCRCVRP program compares the serial number of the direct access volume to that specified by the VOLID parameter. If there is no volume label, or if the serial numbers are not equal, a message is written and the request is aborted.

Entry point to the volume label checking routine in all three of the independent utility programs is at location CKVOLLBL.

MESSAGE OUTPUT ROUTINE

The entry point to this routine is SYSOUT. This routine writes messages using the message output device as defined by the MSG control statement. The address of the fixed-length message to be printed is passed to this routine in register GR2. The appropriate CCW is then constructed, and its address is passed in register GR2 to routine STARTIO. Upon regaining control, this routine returns to the calling routine.

WRITE TO OPERATOR ROUTINE

The entry point to this routine is OPPRNT. This routine writes messages which need to be brought to the immediate attention of the operator. The message is given on the console typewriter if one is available.

I/O CONTROL ROUTINE

This routine controls every I/O operation performed by the independent utility programs. It is entered at STARTIO, at which time register UCBREG contains the address of the appropriate UCB, and register CSR3 contains the address of the CCW to be executed. The channel-unit number is loaded into register CSR4. This routine stores the CCW address in the CAW and issues the SIO instruction. If the unit is unavailable, the WAIT state is entered and the program is terminated. If the unit is busy, the SIO is issued until the command is accepted, at which time the TIO instruction is issued repeatedly until the unit is not busy. At this time control is given to CKCSW, the entry point to the I/O interruption analysis routine. The IBCDMPRS program returns control to the calling routine, however, to continue processing as soon as the I/O is started.

UNIT CONTROL BLOCKS: The independent utility programs each contain one unit control block (UCB) for each device in use. Figure 62 lists the UCBs and their uses. UCBs for the independent utilities have the following format:

Byte	Function
00	unit reference number
01	used only by IBCRCVRP; set to X'FF' if the UCB is for a tape drive, set to zero when label is checked
02-03	channel-unit
04	CAW protect
05-07	CAW
08-15	interruption PSW
16-23	interruption CSW
24-31	sense bytes

UCB Label	Use in IBCDASDI	Use in IBCDMPRS	Use in IBCRCVRP
UCBTO	'TO' device	'TO' device[1]	'TO' device[1]
UCBFRM	unused	'FROM' device[1]	'FROM' device[1]
UCBSYI	control statement input device	control statement input device	control statement input device
UCBSYO	message output device	message output device	message output device
UCBOPR	operator message device	operator message device	operator message device
UCBLIST	unused	unused	record data listing device
UCBSERT	unused	unused	'DATA' replace statements input device (REPLACE only)

[1]'TO' and 'FROM' are relative to the operation being performed by the programs. For a dump from 2311 disk storage to tape, for example, 'TO' refers to tape and 'FROM' refers to 2311; whereas for the companion restore, 'TO' refers to 2311 and 'FROM' refers to tape. A parallel situation exists for recovering and replacing.

Figure 62. The Use of UCBs in the Independent Utilities

I/O INTERRUPTION ANALYSIS

All I/O interruptions cause control to be given to the I/O interruption analysis routine, whose entry point is CKCSW. Register UCBREG contains the address of the applicable UCB. This routine checks the nature of the I/O interruption:

1. Error: control is given to IOERR.
2. Attention: control is given to ATTN.
3. Busy: the SIO is reissued.
4. Device end: control is given to IORTRN.
5. Unit end: the SIO is reissued.
6. Channel end: the TIO is reissued for device end.

IOERR: The CSW, PSW, and CAW are saved, and control is given to SENCHK (in case of a unit check) or TYPECHK (otherwise).

ATTN: The request is honored.

IORTRN: If a surface check is indicated, control is given to the appropriate (device-dependent) surface check routine; otherwise, control is returned to the routine which first issued the call to STARTIO. In the case of IBCDMPRS, the UCB is posted complete and control is returned to the routine which first issued the call to STARTIO.

SENCHK: The device address is entered in SIO and TIO instructions, a sense CCW address is stored in the CAW, and the SIO is issued until it is accepted, at which time the TIO is issued. The TIO is reissued until it is accepted, at which time control is given to TYPECHK.

TYPECHK: The device type causing the interruption is determined by interrogating the UCB, whose address is in register UCBREG. Control is then given to one of the following locations:

Device Type	Location
2302,2303,2311,2314	ERR100
1442	ERR200
2400 series tape units	ERR300
1403	ERR400
1052,2150	ERR500
1402	ERR600
2301	ERR700
1443	ERR800
2321	ERR900

At each of the locations - ERR100, ERR200, ... IER900 - is the instrction

```
BAL ERRLINK,ERRTEST
```

followed by a table of two-byte entries. The instruction loads the address of the table into register ERRLINK and then gives control to routine ERRTEST, which uses the indicated table to interrogate status and/or sense bits.

Each two-byte entry in the indicated table consists of a one-byte relative pointer to a status or sense bit and a one-byte relative pointer to a routine. Routine ERRTEST successively interrogates the bit indicated by the first byte of the table entry; if the bit is on, ERRTEST directs control to the routine indicated by the second byte of the table entry; if not, ERRTEST processes the next entry in the table.

The settings of the first byte of each table entry are as follows:

	Bits	Setting	Meaning
Case 1:	0-3	X'1'	The bit to be tested is a status bit.
	4-7	X'y'	y = the bit position (hexadecimal) of the bit to be tested, relative to bit 32 of the CSW.
Case 2:	0-3	X'0'	The bit to be tested is a sense bit.
	4-7	X'y'	y = the bit position (hexadecimal) of the bit to be tested, relative to bit 0 of sense byte 0.

If the tested half-byte is found to be on, ERRTEST directs control to location A+B,

where:

A = the address of the first byte of the current table entry;

B = the value of the second byte of the current table entry.

Initializing and Assigning Alternate Tracks on Direct Access Volumes (IBCDASDI)

The direct access storage device initialization (IBCDASDI) program performs one of two functions during a single execution:

- Initializes a direct access volume to conform to Operating System/360 specifications.
- Obtains alternate tracks for specified defective tracks on an already initialized disk storage volume.

The current version of this program initializes a volume on:

- 2301 drum storage
- 2302 disk storage
- 2303 drum storage
- 2311 disk storage
- 2314 disk storage
- 2321 data cell storage

The program obtains alternate tracks for a volume on:

- 2302 disk storage
- 2311 disk storage
- 2314 disk storage
- 2321 data cell storage

Initializing a direct access volume consists of the following:

- Detecting defective tracks.
- Assigning alternates to defective primary tracks (on disk storage only).
- Writing the standard home address and record zero on each track.
- Writing track zero, consisting of two IPL records, a standard volume label, and space for seven additional volume labels (see Figure 63).
- Writing a standard volume table of contents (VTOC) at a user-specified location.
- Optionally writing the IPL initialization program.

Obtaining an alternate track for a user-specified defective primary (i.e., nonalternate) track on disk storage consists of the following:

1. Selecting the first available operative alternate track from those indicated in the VTOC of the specified volume.
2. Writing the address (CCHHR) of the primary track in the count field of the selected alternate track, and writing the address (CCHHR) of the alternate track in the count field of the primary track.
3. Modifying fields five and six of the VTOC DSCB to reflect the new status of available alternate tracks.

PROGRAM FLOW

Chart 76 shows the logical flow of the DASDI program. This section describes the operations performed by the IBCDASDI program relative to its functions: initializing a volume and obtaining alternate tracks.

Descriptions of the following supervisory routines of the IBCDASDI program may be found in this publication in the section entitled "Supervisory Routines of the Independent Utilities."

- Input Device Check (CKINPUT)
- Data Input (SYSIN)
- Control Statement Analysis (CLRSCAN)
- Message Output (SYSOUT)
- Write to Operator (OPPRNT)
- I/O Control (STARTIO)
- I/O Interruption Analysis (CKCSW)

After the input device has been defined by the operator and checked for validity by the IBCDASDI program (see "Checking the Input Device"), control statements are read and analyzed (see "Control Statement Analysis") and control is given to the appropriate initialization or GETALT section of the program.

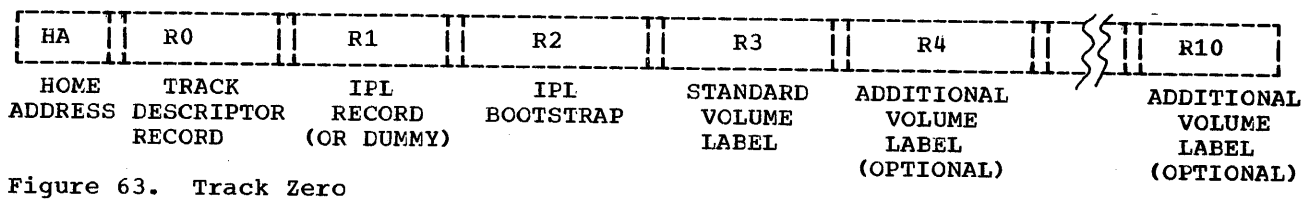


Figure 63. Track Zero

Initializing a Volume

The following routines are executed to initialize a volume:

- INTALT, which initializes a track for disk and drum devices.
- WRITECT1, which initializes a track for data cell storage.
- CONSTR1, which builds an image of track zero in main storage.
- YESUSER, which places additional volume labels in the track zero format.
- CONSTR2, which writes track zero.
- WRTIPL, which writes the IPL initialization program, if requested.
- FMTVTOC, which builds the VTOC.
- WRTVTOC, which writes the VTOC.

Following execution of WRTVTOC, the program initiates normal end-of-job and the CPU assumes the WAIT state.

INTALT
initializes a track for disk and drum devices. When the device is disk, INTALT first checks the track for having been previously flagged as defective. (This test can be suppressed for the first initialization on that volume.) Alternate tracks are immediately assigned for tracks flagged as defective.

Disk and drum track initialization may or may not include surface analysis. When the recording surface is to be checked, the alternate tracks are checked first. (The alternate track concept is not defined for drum storage.) If an alternate track is found to be defective, it is flagged as such (later, FMTVTOC adjusts field six of the VTOC DSCB to indicate the number of available alternate tracks). If a primary track is found to be defective, it is assigned an alternate by ASGNALT, which is the same routine used to assign alternate tracks for a GETALT execution of IBCDASDI. After the track is assigned by ASGNALT and a message printed, control is returned to the initialization section of the program, at which time the next track is checked, or, if all tracks have been checked, track zero is constructed. Tracks are checked for a good recording surface in the following way:

1. When the flag test has been suppressed, the home address (HA) is written followed by a maximum-length record zero consisting of data field of identical bytes of hexadecimal 55.
2. The track is read and checked.
3. A maximum-length record zero is again written, this time consisting of data field of identical bytes of hexadecimal 00.
4. The track is read and checked.
5. If no data error has occurred in steps 2 to 4 and no additional passes are requested, record zero is rewritten (see step 8). If additional passes are requested on this track, repeat steps 1 to 4.
6. If either step 2 or step 4 have indicated a data error, steps 1 to 4 are repeated ten more times, unless a data error occurs.
7. If any other data error occurs during step 6, the track is flagged as defective. An alternate track is assigned when the device is disk. For drum devices, a message is given indicating the address of the defective track. If the HA-R0 area is defective on a 2314 disk storage volume, an attempt is made to move the HA-R0 fields down the track approximately 800 bytes.
8. A track descriptor record (R0) is then written and verified as an 8-byte count field followed by an 8-byte data field of zeros.
9. When all tracks have been initialized, control is given to CONSTR1. Otherwise, the sequence is repeated for each track. (When initialization without surface analysis is requested, only steps 8 and 9, are repeated for each track.)

WRITECT1
performs data cell track analysis in the following way:

1. A home address (HA), track descriptor record (R0), and a maximum length record one (R1) are written on each of 20 tracks of a cylinder. The data field of R1 consists of identical bytes, containing hexadecimal E5.

2. An address compare is made on each of the tracks written in step 1, and record one is verified for each track.

3. Record one is erased for each track written above.

4. If no errors occur in step 2, steps 1 to 3 are repeated for each cylinder with additional address compares made after the completion of each strip, subcell, and cell.

5. If an error (i.e., data check or missing address marker) has occurred during step 2, the track is rewritten and reread until either a successful pass is obtained or 113 errors have occurred. If this track is in the alternate area, it is flagged to prevent its future use. Otherwise, an alternate track is assigned by ASGNALT, and a message is printed.

6. When all tracks have been initialized, control is given to CONSTR1.

CONSTR1
: constructs track zero. If the IPL function is selected, records one and two are written as an IPL bootstrap program and a program to load the IPL initialization program. If the IPL function is not selected, record one is written as a program to set the WAIT state in the CPU in case the volume is loaded for execution.

Regardless of whether the IPL function is selected, record two is written as an IPL bootstrap. (Since record one will set the WAIT state in the CPU in case a non-IPL volume is loaded for execution, there is no danger of executing record two.)

YESUSER
: writes up to seven user-supplied additional volume labels as records 4-10. Space is allocated for those volume labels not supplied.

CONSTR2
: writes track zero, consisting of two IPL records (or a dummy IPL record), a standard volume label and up to seven additional labels.

WRTIPL
: writes the user-supplied IPL initialization program, if requested. The program is written on the first track preceding the alternate track area (track 1999 on 2311), or, if that track is defective, on its assigned alternate.

FMTVTOC
: constructs the DSCBs needed for the VTOC. They are the VTOC DSCB (format 4) and the DADSM DSCB (format 5).

WRTVTOC
: writes at the user-specified location of the VTOC the DSCBs constructed by FMTVTOC.

Obtaining Alternate Tracks

If the IBCDASDI program is executed under the GETALT option, control is given to location GETALT following control card analysis. Routine GETALT performs a track check on the user-specified track if the track check bypass is not selected. If the track is found to be operative, a message to that effect is printed (or displayed) and the next GETALT request is processed. If the track check bypass is selected, or if the track is found to be defective, the following routines are executed in the order in which they appear.

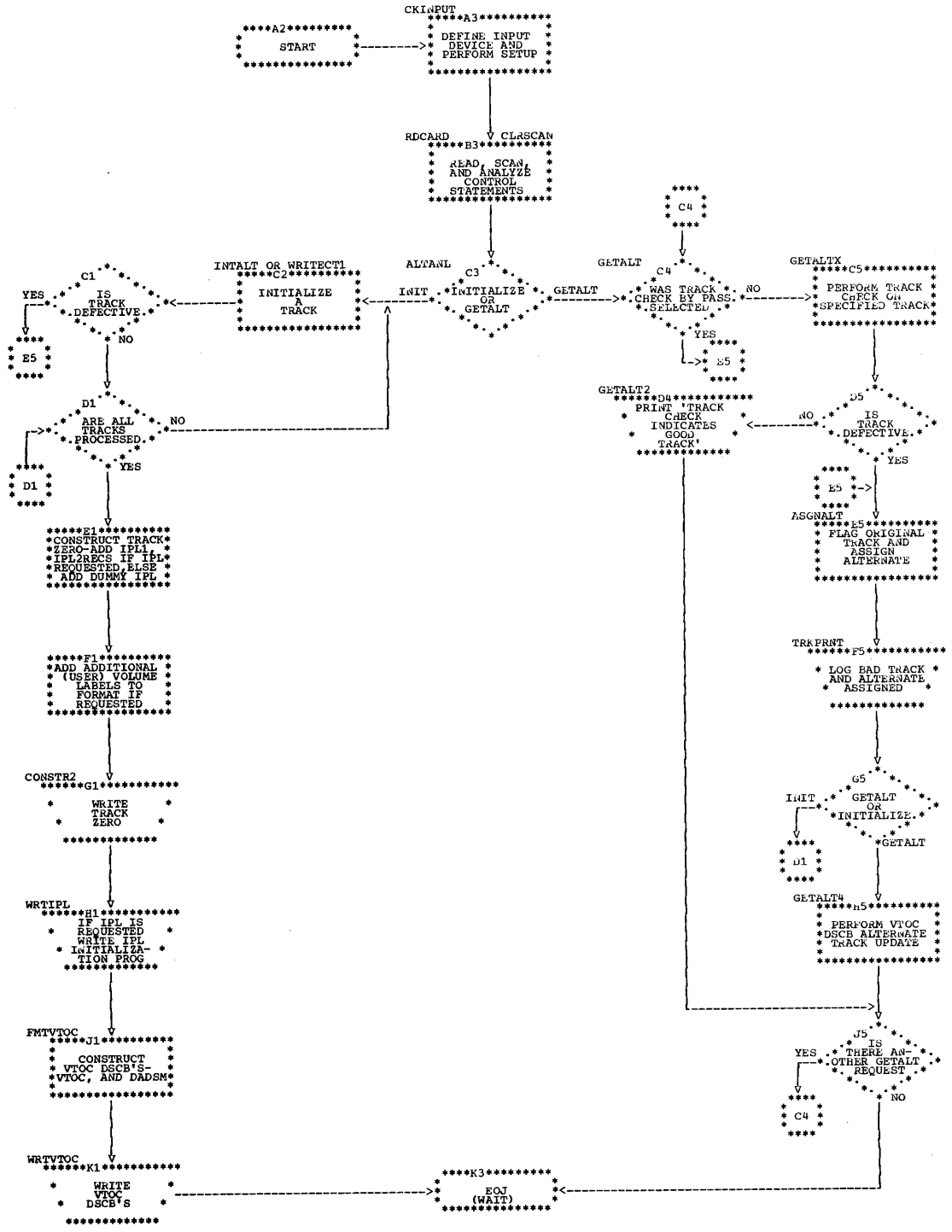
ASGNALT
: flags the given track as defective and assigns it an alternate as described, if it is a primary track. If the given track is an alternate, it is flagged as defective; if the given alternate track had been assigned to a primary, an operative alternate is assigned to the primary.

TRKPRNT
: causes a message to be printed stating the addresses of the defective track and its assigned alternate.

GETALT4
: decrements field six of the VTOC to reflect the fact that one less alternate track is available, and increments field five to point to the next available alternate track.

Control is then given to location GETALT to repeat the process for the next user-specified track, or, if none exists, initiates normal end-of-job and sets the CPU to the WAIT state.

Chart 76. IBCDASDI - Initializing and Assigning Alternate Tracks on Direct Access Volumes



Dumping and Restoring a Direct Access Volume (IBCDMPRS)

The direct access storage device dump-restore program performs one of two functions during a single execution:

- Dumping (copying) data from a direct access volume to 2311 or 2314 disk storage or magnetic tape, in a format recognizable to the restore portion of the program.
- Restoring (recopying) data which has been dumped by this program. Data is restored only to a volume residing on a device of the same model number from which it was dumped.

There is no provision to restore from 2311 to 2311 or from 2314 to 2314. Instead, another dump of the same type may be performed.

A dump may be either partial (a set of contiguous tracks is dumped) or entire (the entire volume is dumped).

The current version of this program dumps the data contents of a volume from:

- 2301 drum storage to magnetic tape or 2311 disk storage or 2314 disk storage.
- 2302 disk storage to magnetic tape or 2311 disk storage or 2314 disk storage.
- 2303 drum storage to magnetic tape or 2311 disk storage or 2314 disk storage.
- 2311 disk storage to magnetic tape or 2311 disk storage or 2314 disk storage.
- 2314 disk storage to magnetic tape or 2311 disk storage or 2314 disk storage.
- 2321 data cell storage to magnetic tape or 2311 disk storage or 2314 disk storage.

DUMPED DATA FORMAT

The format of dumped data depends on the device configuration of the dump: 2311 to 2311 (or 2314 to 2314), direct access to tape, or non-2311 direct access to 2311 (or non-2314 to 2314).

2311 TO 2311 (OR 2314 TO 2314): Data from the input 2311 (or 2314) is copied record-for-record and track-for-track. For this reason a restore from 2311 to 2311 (or 2314 to 2314) is not provided, but can be effected by another dump.

DIRECT ACCESS TO TAPE: The following records are written on tape for a direct access-to-tape dump (see Figure 64):

- A limits record is written as the first record (following any labels) on each volume of tape. This record contains the addresses of the first track dumped, the last track dumped, and the the first track dumped on this volume of tape.
- A control record is written for each track dumped, immediately preceding the dumped data from the track. The control record contains a channel program to be used by a subsequent restore to write one track.
- A dumped track image is written as a maximum-length physical record. A track image is not split between tapes.
- A trailer label is written at the end of each tape volume, immediately following the tape mark. During a restore, successive oring of trailer labels indicates whether another FROM volume is to be mounted. The mounting of FROM volumes during a restore is thus order-independent.

NCN-2311 TO 2311 (OR NON-2314 TO 2314): The records written as record one of track one of each 2311 (or 2314) volume needed for the dump are similar to those for tape, but with the following differences:

- The limits record is written as record one of track one of each 2311 (or 2314) volume needed for the dump. The limits record contains (as with tape) the addresses of the first track dumped, the last track dumped, and the first track dumped onto this 2311 (or 2314) volume.
- The control record is written immediately preceding each dumped track image. The first control record on a volume is written as record one of track two; subsequent control records are each written as record one of the first track following the image of the last track dumped. The control record consists of two subsets: (1) eight two-byte fields, each containing the number of bytes of the original (dumped) track written on a track of the 2311 (or 2314) and (2) a channel program to be used by a subsequent restore to write one track.
- A dumped track image is written in maximum-length physical records on as many 2311 (or 2314) tracks as are necessary. The number of bytes of the

dumped non-2311 (or non-2314) track written on each 2311 (or 2314) track is recorded in the control record for the track image. A dumped track image is not split between disk packs.

- The trailer label is written as record one on the last available track of each 2311 (or 2314) disk pack used. The contents of the trailer label for 2311 (or 2314) are identical to those for tape.

PROGRAM FLOW

The flow of the direct access storage device dump/restore program is shown in Chart 77. Descriptions of the following supervisory routines of the direct access storage device dump/restore program may be found in this publication in the section entitled "Supervisory Routines of the Support Utilities."

- Input Device Check (CKINPUT)
- Control Statement Analysis (CLRSCAN)
- Message Output (SYSOUT)
- Write to Operator (OPPRNT)
- I/O Control (STARTIO)
- I/O Interruption Analysis (CKCSW).

After the input device has been defined by the operator and checked for validity by this program, control statements are read and analyzed and control is given to the appropriate dump or restore section of the program.

Dumping

If the program is dumping, the following routines are executed in the order listed.

TOTAPE
ensures that the TO volume is mounted, whether tape or not. If the dump is not from 2311 to 2311 (or not from 2314 to 2314), this routine also writes the limits record.

MODTKADF
reads the count fields on one track of the FROM volume and at the same time, if two channels are used, writes header or data records on tape from location DTABUFF.

ANALSENS
uses the information obtained from reading the count field of one track to construct a channel program capable of reading the count, key, and data fields of the track.

READCCWs
moves the channel program to a higher area in main storage and executes the channel program constructed by ANALSENS, reading one track of the FROM volume into the buffer DTABUFF. (In the buffer, record images are blocked.)

TSTWRTSP
converts the channel program at location DTALENG to a channel program capable of writing the buffer (with read-back check) onto a track of the same device from which it was read in its original format.

If the dump is 2311-2311 (or 2314-2314), the channel program is executed, thus writing one track on to the 2311 (or 2314).

If the dump is not 2311-2311 (or not 2314-2314), the converted channel program is not executed during dumping, but will be executed during a future restore. After converting the channel program, this routine gives control to DMPDASD if the TO device is tape, or to STRTDSK if the TO device is 2311 (or 2314).

DMPDASD
writes the control record, consisting of the channel program at location DTALENG on the tape. Control is then given to MODTKADF, EOJ1, EOJAA, or the program terminates (see Chart 77).

STRTDSK
writes the control record and the buffer on 2311 (or 2314) disk storage. The function performed is similar to that of DMPDASD (writing on tape), but with the following exceptions (see Figure 64).

- The control record for dumping from non-2311 to 2311 (or non-2314 to 2314) consists of a 16-byte field beginning at DTALENG prefixed to the channel program at location CCWLIST.

- Several 2311 (or 2314) tracks may be needed to contain the data in the buffer at DTABUFF. If so, the buffer is written in maximum-length physical records on as many tracks as are needed. A buffer image is not split between disk packs. Any remaining space on the last track needed to contain the buffer image is not used. (The next control record begins on the next available track.)

Control is then given to MODTKADF, EOJ1, EOJAA, or the program is terminated (see Chart 77).

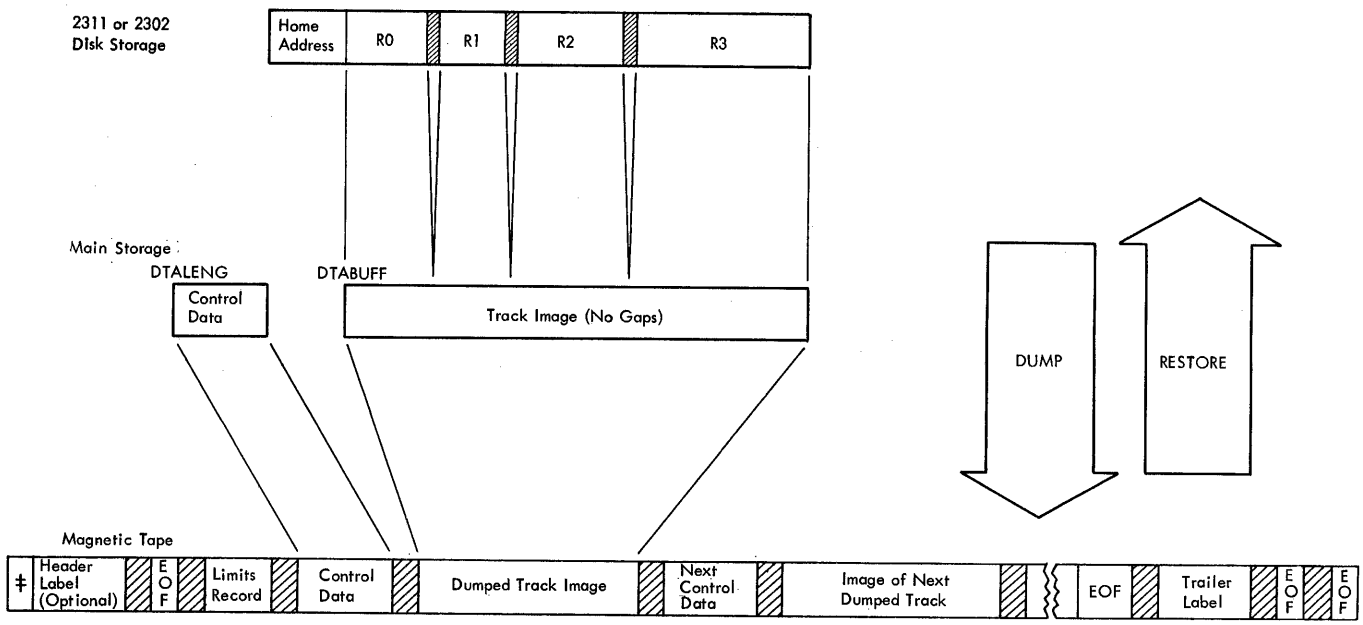


Figure 64. Dumping and Restoring a Direct Access Track

EOJ1
is given control when a new TO volume is needed. EOJ1 writes the trailer label on the current TO volume and then gives control to routine TOTAPE to insure that a new volume is mounted. (See "Dumped Data Format" for a description of the trailer label and its location for tape or disk.)

EOJAA
is given control at the conclusion of an entire 2311-2311 (or 2314-2314) dump. EOJAA updates field six of the VTOC DSCB to reflect any alternate track assignments necessitated during the dump. A WAIT state is then set in the CPU and the program terminates.

Restoring

After the input device has been verified and control statements have been analyzed (see "Supervisory Routines of the Independent Utilities"), control is given to the restore section of the program, consisting of the following routines, which are executed in the order indicated.

FRMTAPE
ensures that a FROM volume is mounted, whether tape or disk. The order of volume mounting is immaterial. After a FROM volume is mounted, this routine reads the limits record (record one). Control is then given to RSTRTAPE, if the FROM device is tape, and to STRTDSK, if the FROM device is 2311 (or 2314) disk storage. When the device is not the 2301 drum and if there is at least 64K of main storage, buffers are built in upper storage for the data records and the channel programs.

RSTRTAPE
reads the control record into location DTALENG1, when storage is available. (The control record consists of a channel program capable of restoring the dumped track.) From DTALENG1, the record is moved to DTALENG. The image of the dumped track (in blocked record format) is read into location DTABUFF1, when storage is available, and then is moved to DTABUFF. Control is then given to MODTKADT.

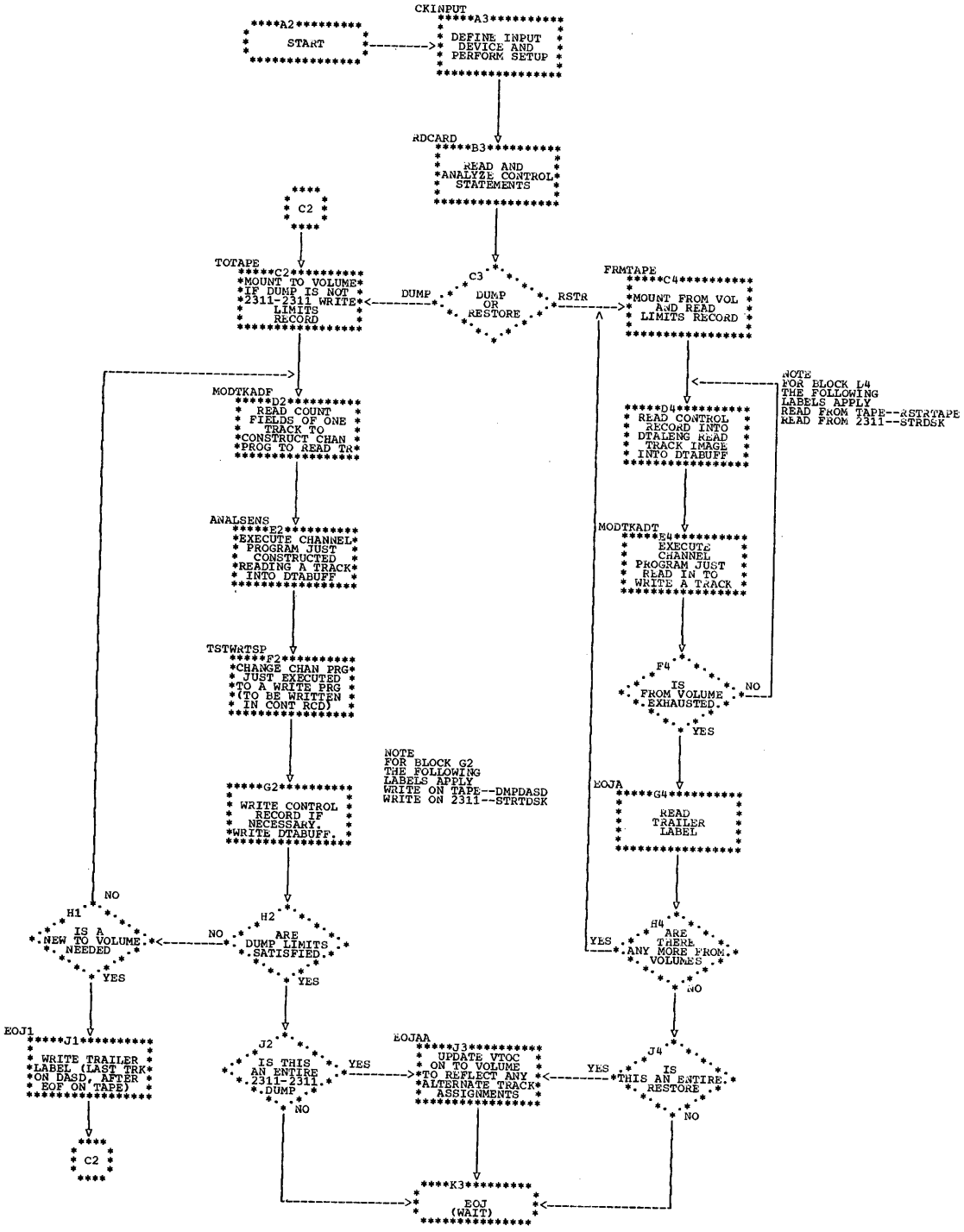
STRTDSK
performs the same logical function as RSTRTAPE, but reads instead from 2311 (or 2314) disk storage. The control record is first read into location DTALENG (also causing the channel program, the second field of the control record, to be read into location CCWLIST). The first field of the control record is then used to read as many tracks as are necessary to "fill" the buffer DTABUFF, that is, to complete one dumped track image in the buffer. Control is then given to routine MODTKADT.

MODTKADT
executes the channel program at location DTALENG, thus restoring one track in its original format. If the FROM volume is not exhausted, control is given to RSTRTAPE or STRTDSK, depending on whether the FROM device is tape or disk, respectively. When the FROM volume is exhausted, control is given to EOJA to read the trailer label.

EOJA
reads the trailer label (for a description of the trailer label and its location, see "Dumped Data Format"). Successive oring of trailer labels by this routine controls FROM volume mounting. If another FROM volume is to be processed, control is given to FRMTAPE to insure that it is mounted, whether tape or disk. If no more FROM volumes are to be processed, control is given to EOJAA (if the restore is entire), or the program is terminated (if the restore is partial). Note: a restore is entire or partial depending only on the limits of the companion dump.

EOJAA
updates field six of the VTOC DSCB to reflect any alternate track assignments necessitated during the (entire) restore. No such update is provided for a partial restore.

Chart 77. IBCDMPRS - Dumping and Restoring a Direct Access Volume



Recovering and Replacing a Track (IBCRCVRP)

The recover-replace program performs one of two functions during a single execution:

- Recovering (reading) data from a track on an initialized direct access volume; listing defective records, or all records, if specified; and writing the good data on a recovery output tape for use by the replace portion of the program during a future execution.

- Replacing a good track image on an operative track by merging data from the recovery output tape with replacement data supplied by the user.

Requests may be stacked, but all must specify the same function -- recover or replace.

The current version of the program supports recovery and replacement of data on:

- 2302 disk storage
- 2303 drum storage
- 2311 disk storage
- 2314 disk storage
- 2321 data cell storage

As a stand-alone program, recover-replace contains the following supervisory routines, described under the heading, "Supervisory Routines of the Independent Utilities":

- Input Device Check (CKINPUT)
- Data Input Routine (SYSIN)
- Control Statement Analysis (CLRSCAN)
- Volume Label Check (CKVOLLBL)
- Message Output Routine (SYSOUT)
- Write to Operator (OPPRNT)
- I/O Control (STARTIO)
- I/O Interruption Analysis (CKCSW)

The logic of the recover and replace portions of the program is shown in the following charts:

Chart 78.	Overall Logic
Chart 79.	Recover Logic
Chart 80.	Recover Data Check Routine
Chart 81.	Recover Count Check and End-of-track Routines
Chart 82.	Replace Logic

Overall Flow

When the program gains control, it waits for the operator to define the input device from which utility control statements are to be read. The program then verifies that the input device definition is valid, and begins to read, scan, and analyze utility control statements.

Figure 65 suggests how main storage is managed by the program. The space occupied by the replace portion of the program after initial loading is used as buffer for reading the track to be recovered or replaced. A recover run causes the replace coding to be overlaid by the track image; for a replace run the replace coding is first moved to overlay the recover portion of the program.

Depending on the request, the appropriate recover or replace coding is then executed. Following this, listing is performed: for a recover run, if the LIST option is specified all records on the track are listed, or otherwise only the defective records; for a replace run, if the LIST option is specified all records on the replacement track are listed, or otherwise only the replacement records. When all requests have been serviced, the program issues an end-of-job message, rewinds and unloads the tapes, and sets the wait state in the CPU with D's displayed on the console lights.

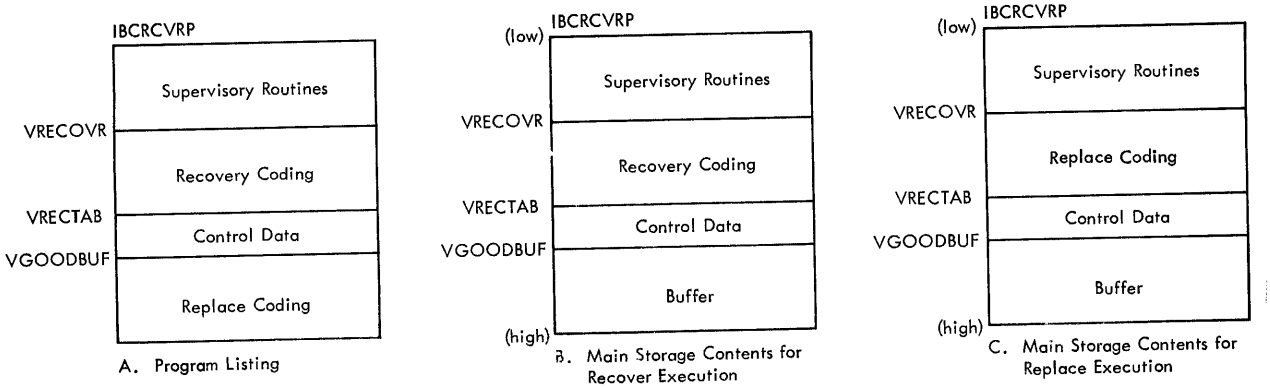


Figure 65. Main Storage Management for Recover Replace

Recovering

The recover portion of the program reads the specified track of the direct access volume, gathers control data to be used by a future replace run, and records the control data and the successfully recovered portion of the track on a recovery output tape. Figure 66 shows the tape format.

Records are read into VGOODBUF. If a data check is detected in the count field of a record or an address marker is missing from a record, the remaining bytes on the track, including records and gaps, are read into VGOODBUF using the space count command and are immediately listed on the message device. After listing, the records and gaps are cleared from VGOODBUF and the next record is read into VGOODBUF immediately following an 8-byte entry left in place of the record which had the bad count or missing address marker. If the count field is good and the address marker is present, any key and/or data fields read, whether good or defective, will remain in VGOODBUF as read. (See Figure 67.)

As each record is read into the buffer, an entry is built in the record control table VRECTAB. Each entry consists of a 1-byte flag and a 3-byte pointer to the record image. The settings of the flag byte in VRECTAB are as follows:

Bit=1	Meaning
0	Bad count field
1	Bad key field
2	Bad data field
3	Missing address marker
4	Last record flag
5	Recovery was aborted
6-7	EOF with key
7	EOF (with or without key)

After reading the track, recover builds at location CCWLIST a channel program which will be completed and executed by replace in writing and read-back checking the data put on the alternate track. Recover then stores into VALTBUF the address of the first doubleword boundary following the recovered data in VGOODBUF. This establishes the area replace uses to receive data for records with bad counts or missing address markers. Recover then writes the recovery output tape.

Replacing

The replace portion of the program, which is moved to overlay the recover portion, reads the recovery output tape, reads replacement data supplied by the user, assigns an alternate track (if the volume resides on disk storage), and writes the merged data on the track.

The header record on the recovery output tape is first read and the serial number of the direct access volume is checked. The next two records (control record and recovered data) are then read into the same absolute storage locations they occupied during the companion recover run (VRECTAB and VGOODBUF). Flag bytes in VRECTAB are then interrogated, and replacement data is read as needed. Replacement data is read into the alternative buffer (pointed to by VALTBUF) if the record to be replaced had a missing address marker or a bad count field; otherwise replacement data is read into VGOODBUF, overlaying the corresponding defective recovered key and/or data portions. When all replacement data has been read, an alternate track is obtained on the volume (if it is non-drum storage), and the merged recovery and replacement data are written on the track using the channel program at location CCWLIST. If the HA-R0 fields are defective on 2314 disk or 2321 data cell storage, the program attempts to move these fields approximately 800 bytes down the track.

Example: Figure 67 illustrates a complete cycle (two executions of the program) for recovering and replacing a track.

ID = 4-byte constant "RECV"
VOL = 6-byte volume ID of direct-access device
TRACK = 12-byte BBBBCCCCHHHH of recovered track
DATE = 8-byte date of assembly MM/DD/YY
PAD = 50 bytes of zeros

CCWLIST = Channel program to be used to replace data on volume
VALTBUF = Pointer to buffer for replacement data
VRECTAB = Table of control data for track

VGOODBUF = Buffer containing recovered data

Figure 66. Format of Recovery Output Tape

(1) During a recover execution, the track containing defective records is read into VGOODBUF; for each record, a flag and pointer are set in VRECTAB. In this example, the given track is found to be in the following condition:
- HA - Good
- R0 - Good
- R1 - Bad count
- R2 - Bad key
- R3 - Bad data
- R4 - Missing address marker
- R5 - Last record, good

(2) The recovery output tape is written, consisting of a header record, a control record, and recovered data. The recover execution terminates.

(3) During a subsequent replace execution, the recovery output tape is read into the same absolute storage locations from which it was written.

(4) Using control data from VRECTAB, replacement data is read into (a) VGOODBUF, or (b) the buffer pointed to by VALTBUF, in case of bad count or missing address marker.

(5) Using the channel program read from the recovery output tape, the merged (old and new) data is written on an alternate track.

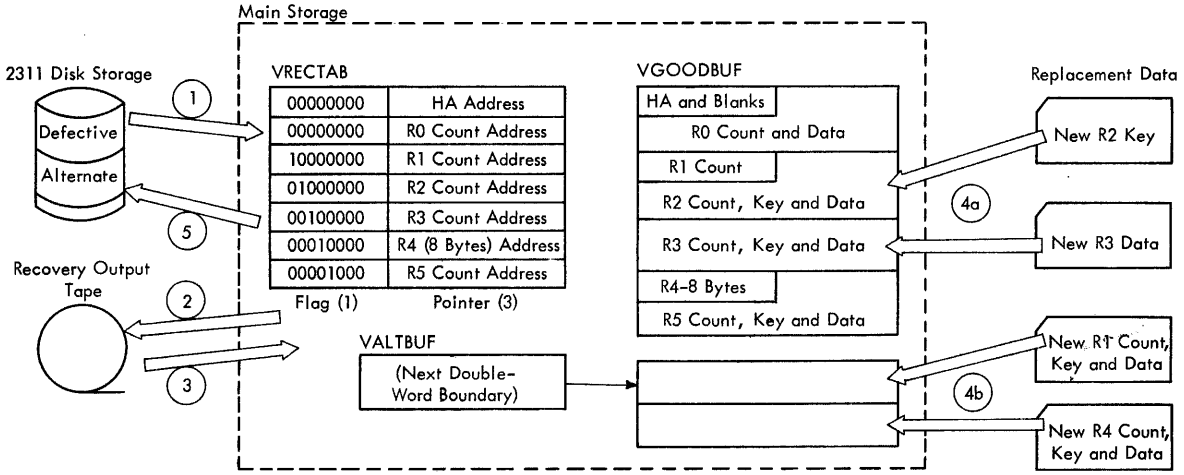


Figure 67. An Example of the Recover-Replace Cycle

Chart 78. IBCRCVRP Overall Logic

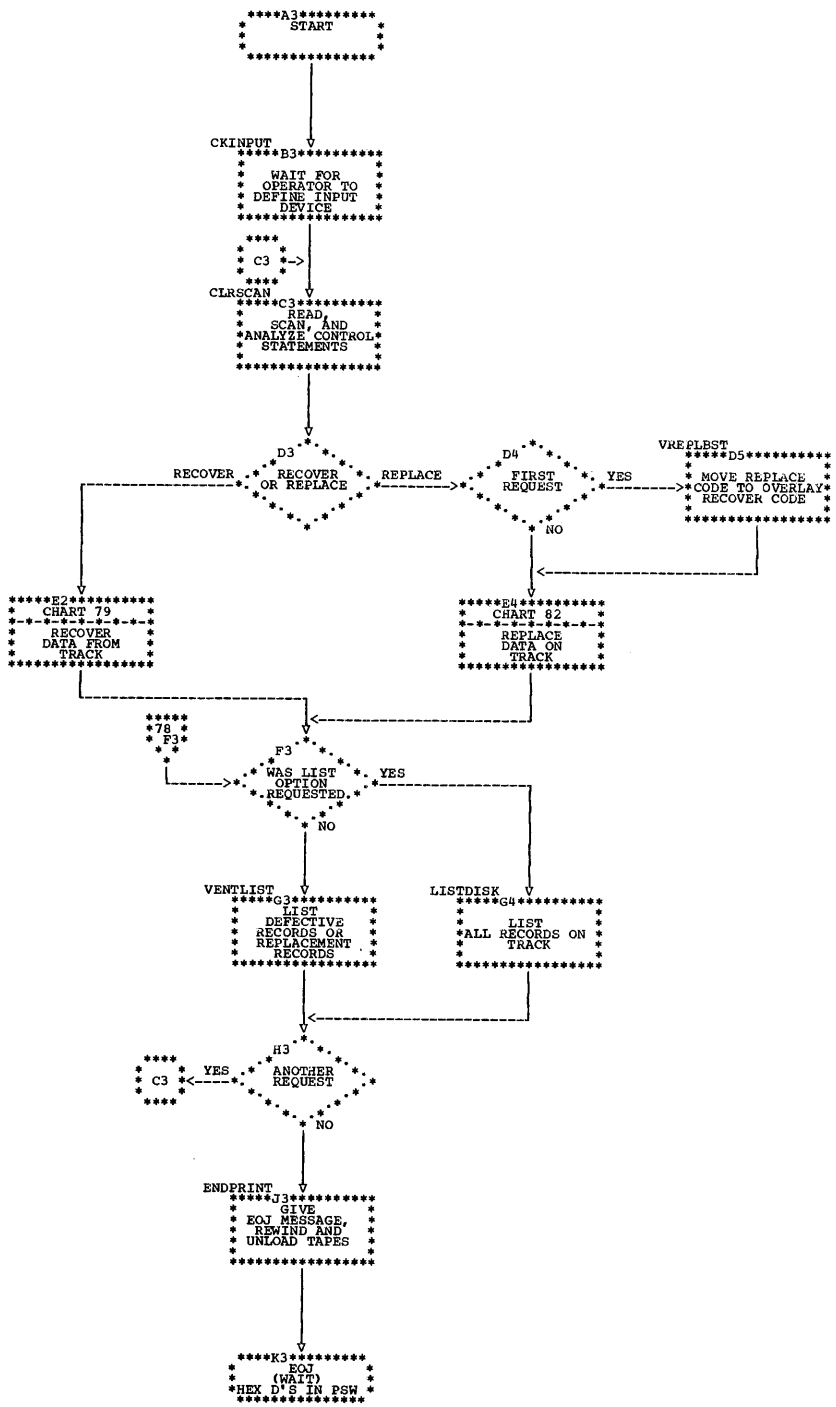


Chart 79. IBCRCVRP Recover Logic

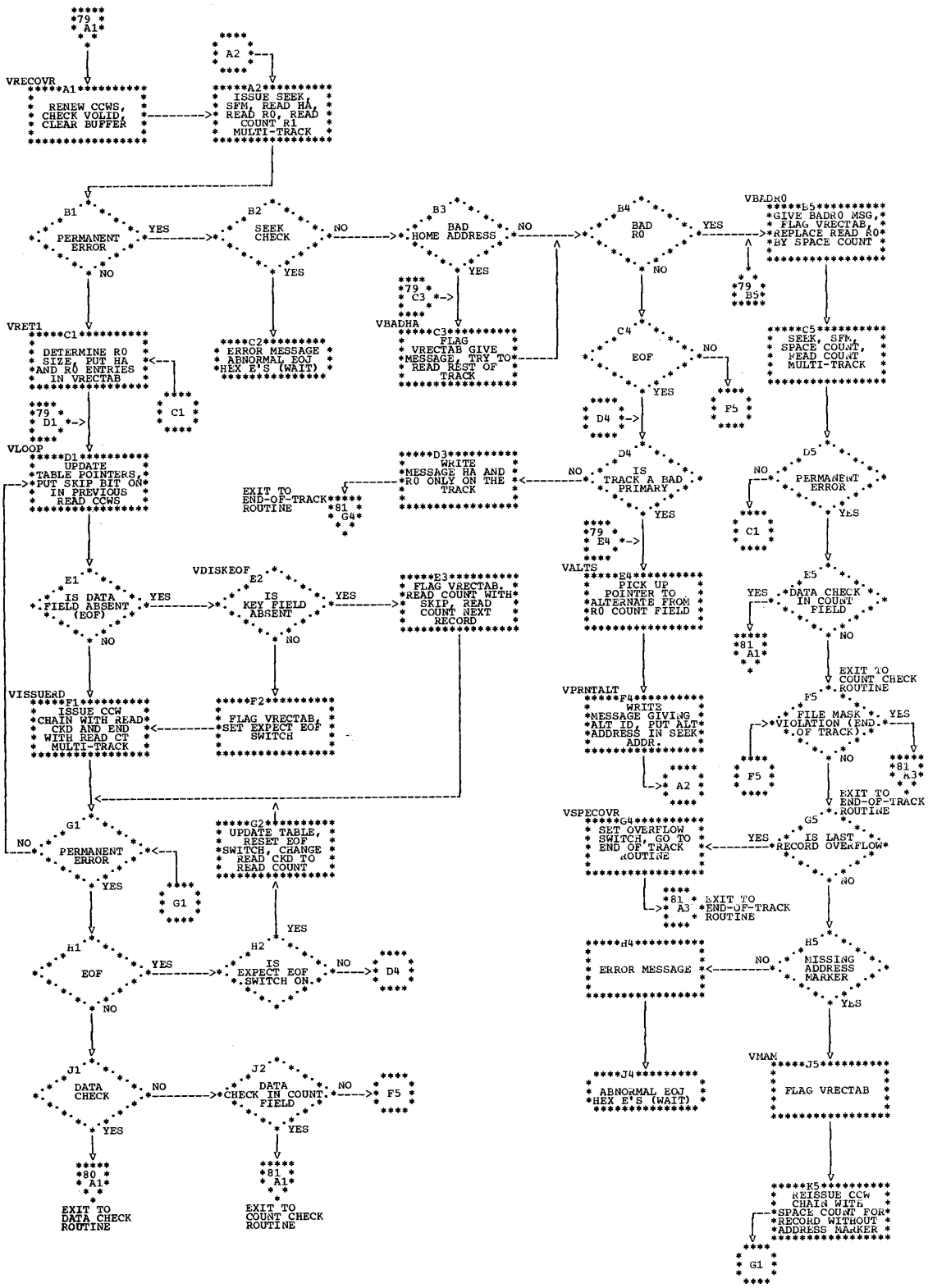


Chart 80. IBCRCVRP Recover Data Check Routine

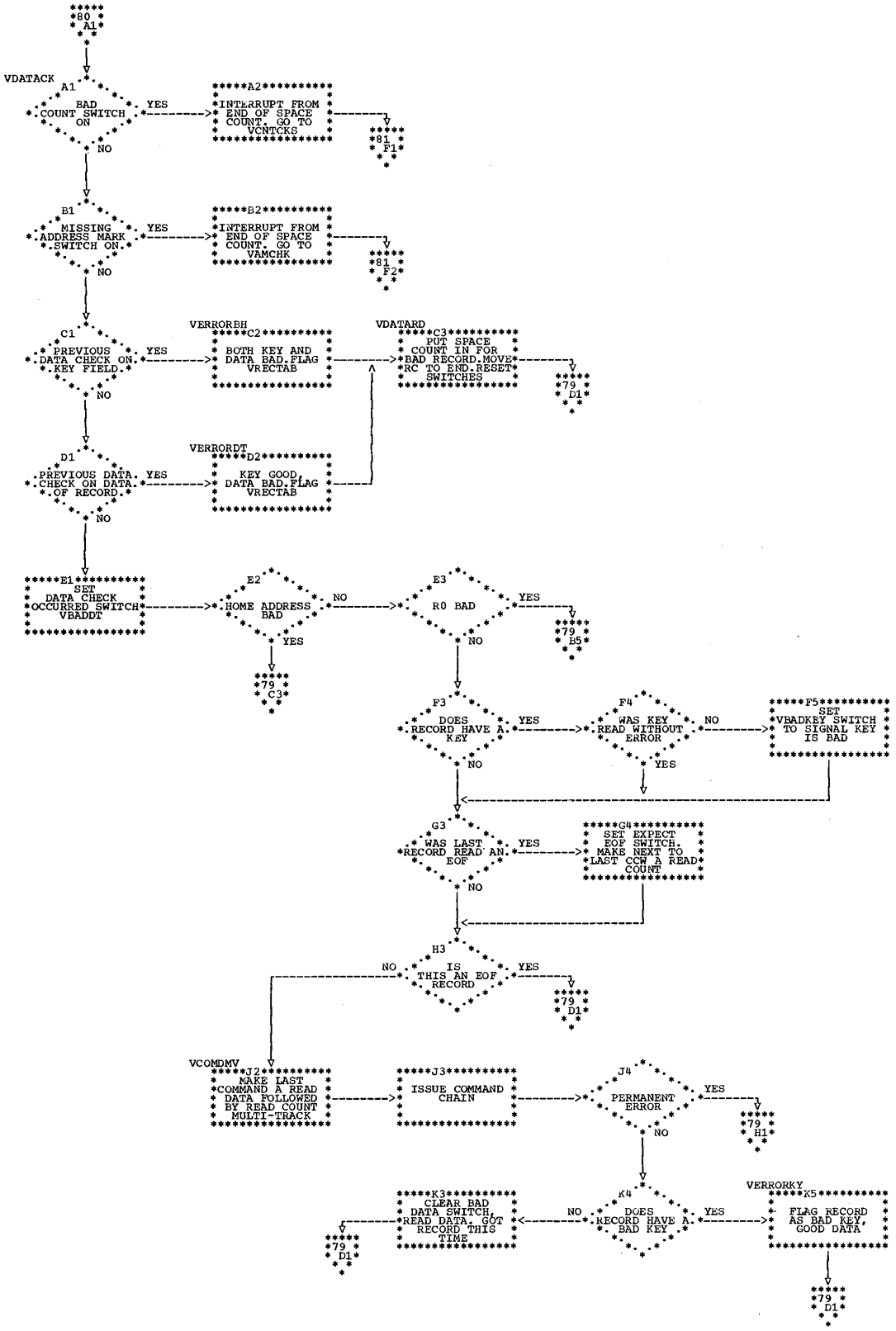


Chart 81. IBCRCVRP Recover Count Check and End-of-Track Routines

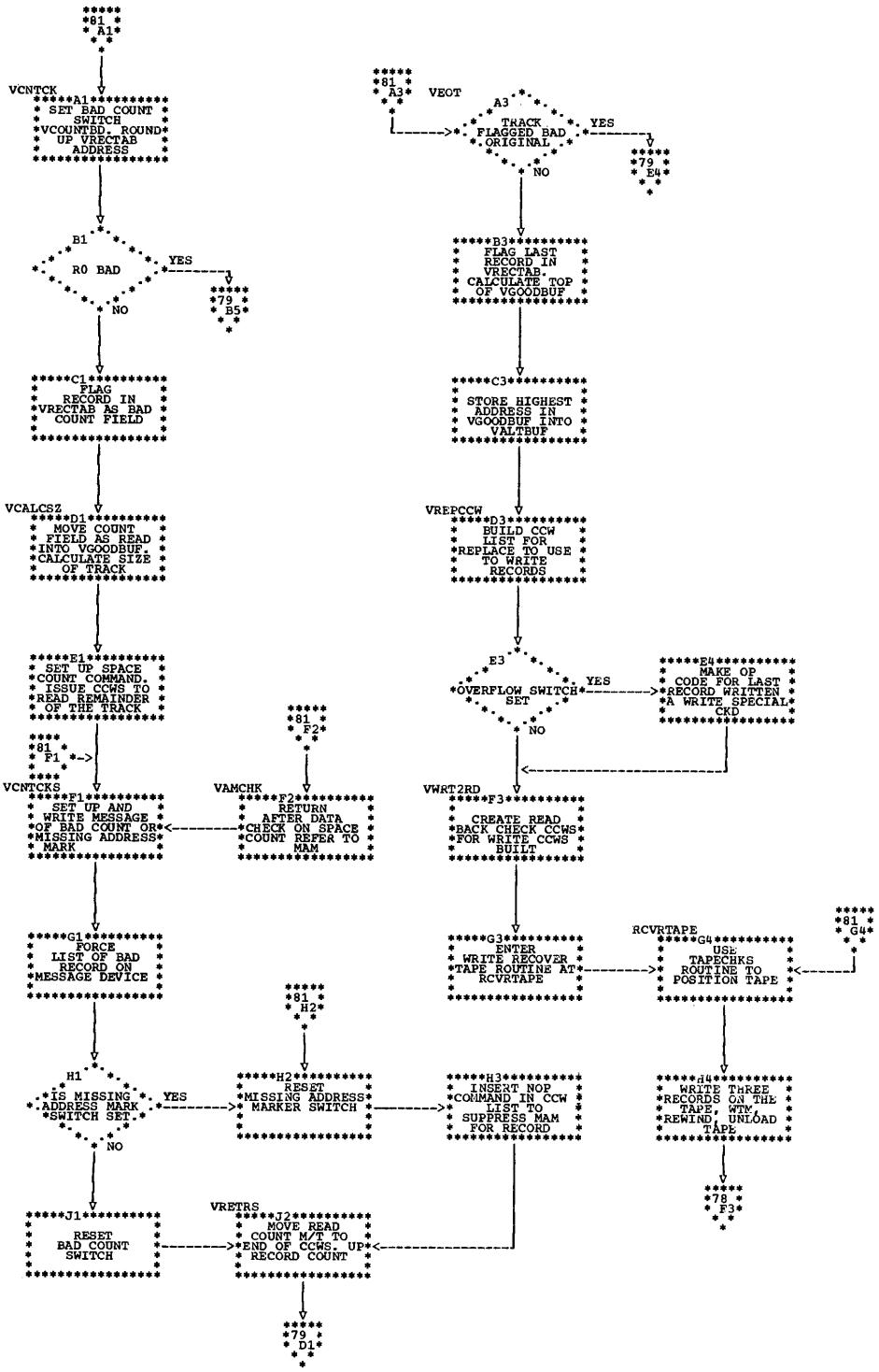
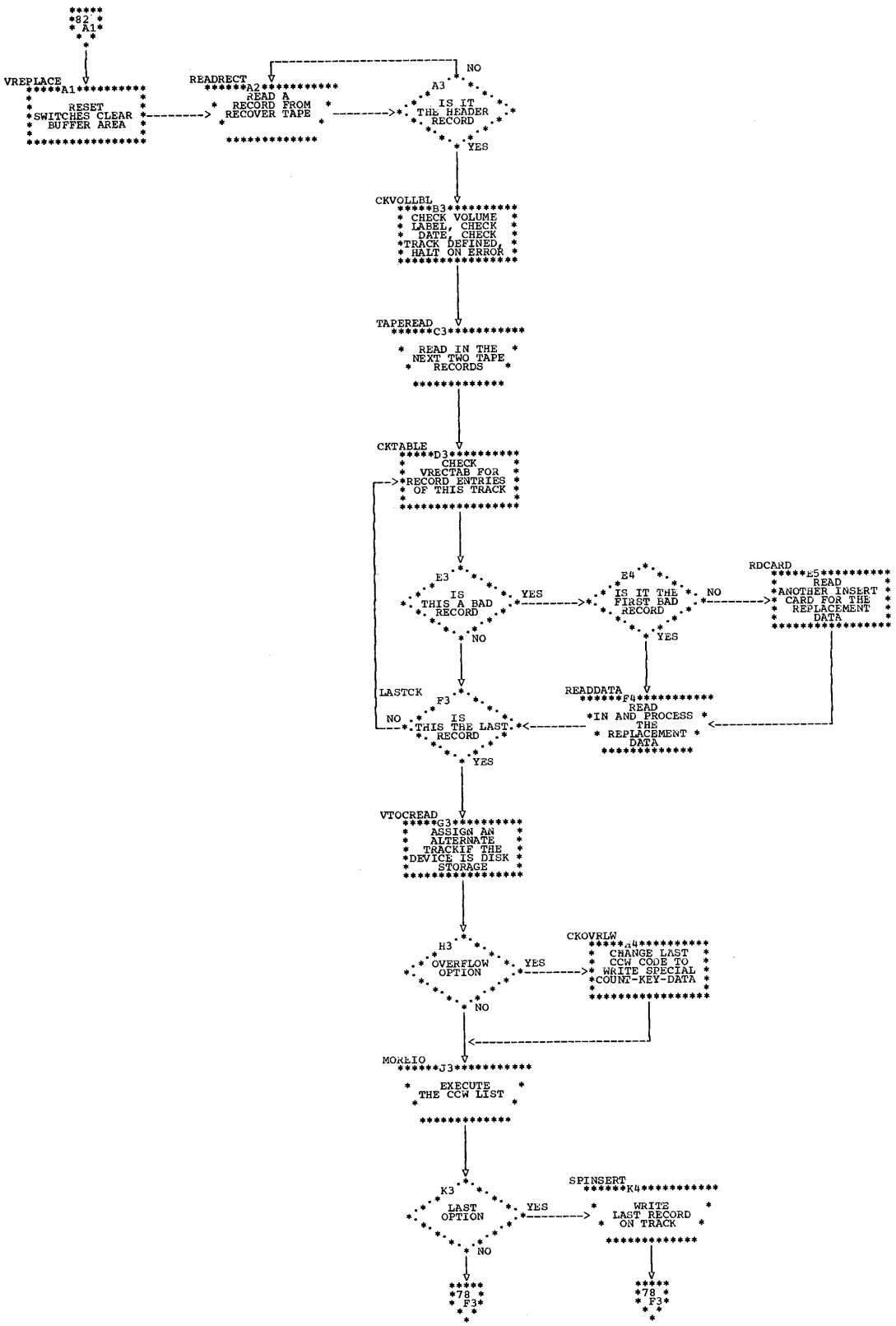


Chart 82. IBCRCVRP Replace Logic



Appendix A: Modules of Utility Programs

This appendix describes the modules of each utility program. The module names given are the SYS1.UT506 names. When these names differ from equivalent SYS1.LINKLIB names, the latter are given in parentheses. In the case of the independent utilities IBCDMPRS, IBCRCVRP, and IBCDASDI, the previous statement is not applicable since these programs are part of the SYS1.SAMPLIB data set.

IEBCOMPR

IEBCROOT
is the root segment; it opens and closes SYSPRINT, writes messages, and calls the proper modules.

IEBCOMPM
is the message module.

IEBCANAL
interprets returns from IEBCCS02.

IEBCMAIN
when the data sets are partitioned, compares directories to determine whether one is a subset of the other; when the data sets are sequential, it compares the data sets.

IEBPTPCH

IEBPPUN1
is the root segment; it opens and closes SYSPRINT data set, calls proper modules, and prints all messages and control cards.

IEBPPMSG
is the message module.

IEBPPAL1
obtains storage for and then constructs tables and work areas, calls and then interprets returns from IEBCCS02, checks for valid parameters.

IEBCCS02
opens and closes SYSIN data set, reads and scans cards, returns data to IEBPPAL1.

IEBPPCH1
is the processor module; it handles sequential and partitioned data sets, opens and closes SYSUT1 and SYSUT2 data sets, checks for valid control cards, and examines tables built by IEBPPAL1.

IEBCOPY

IEBCOPYA
is the root segment; it gives control to the proper modules, and prints all error messages and control cards.

IEBCOPYB
is the message module.

IEBCOPYC
opens and closes SYSPRINT data set, obtains storage for and constructs work areas and tables, calls and then interprets returns from IEBCCS02, and when control cards are present, checks them for validity.

IEBCOPYD
is the processor module; it opens and closes SYSUT1 and SYSUT2 data sets, analyzes tables from COPYC and: if total copy, reads in directory and sorts by TTRs; if exclusive copy, sorts exclude table by MEMBER NAME sequence, reads the data set directory, compares for excludes of directory names, and sorts directory names by TTRs; if inclusive copy, copies included names and moves data from input buffer to output buffer.

IEBEDIT

IEBEDIT
extracts records from a master file of JCL statements to create an edited input stream data set.

IEBGENER

IEBGENRT
is the root segment; it opens and closes SYSPRINT, writes all messages and control cards, and gives control to proper modules.

IEBGMESG
is the message module.

IEBGSCAN
obtains storage for and then constructs tables, calls and then interprets returns from IEBCCS02, analyzes control cards.

IEBGENR3
is the processor segment root module; it opens and closes input and output

data sets and performs label processing.

IEBGENS3 (IEBGENR3)
performs I/O operations for variable spanned records.

IEBGEN03 (IEBGENR3)
performs I/O operations for non-variable spanned records.

IEBMOVE2
moves logical records from input to output buffer.

IEBEDIT2
moves, with editing, logical records from input to output buffer.

IEBCONH2
converts data from H set BCD to EBCDIC.

IEBCONP2
converts data from packed to zoned decimal.

IEBCONZ2
converts data from zoned to packed decimal.

IEBLENP2
computes total output record whenever an input record is encountered.

IEHUCSLD

IEHUCSLD
checks for type of operation, for universal character printer, and for buffer load characters; issues WTOR to mount proper chain; loads the buffer and verifies it, if specified.

IEHIOSUP

IEHIOSUP
finds first load module of SVC routine then loads succeeding modules, reads in the member, and updates member's XCTL table, if present.

IEHINITT

IEHINITT
is the root segment; it opens and closes SYSIN and SYSOUT, builds tape label image in main storage, extracts information from the JFCB, and links to SVC 39 to write the tape label.

IGC0003I (SVC 39)
writes a tape volume label followed by a dummy header label and a tapemark.

IEHSCAN
reads control statements and scans them for INITT command and for keywords.

IEHPRNT
is the message module.

IEHDASDR

IEHDAOUT formats and writes dumped information to the SYSOUT data set.

IEHDASDR
is the entry point for the program. It performs initialization and passes control to the Control routine.

IEHDASDS
is the Control routine. It processes control statements and passes control to the functional routines.

IEHDCELL
is the Data Cell Analysis routine. It performs surface analysis of data cell volumes.

IEHDDATE
is the Date routine. It obtains the day's date and passes it to the Print routine, IEHDPRNT.

IEHDEXCP
is the I/O subroutine of the Dump routine. It performs all I/O operations during a dump except for those performed by IEHDAOUT.

IEHDGETA
is the control routine for performing alternate track assignment.

IEHDLABL
writes new volume serials and owner names on direct access volumes.

IEHDMSGB
is the Message Builder routine. It selects, constructs, and stores messages.

IEHDMSGS
is the message CSECT. It contains the messages used by the IEHDASDR program.

IEHDPASS
is the Password Protection routine. It checks the passwords required for security protected data sets, and checks data set expiration dates.

IEHDPRNT
writes messages to the SYSOUT data set.

IEHDREST
is the Restore routine. It reads dumped information from a restore tape and writes the information on direct access volumes.

IEHDSCAN
is the Scan routine. It reads control statements and scans them for syntax errors, one field at a time.

IEHDVTOC
is used by the Analysis routine to write system data on direct access volumes.

IGC0008B
is the first load of the SVC 82 routine. It builds DEBs for new direct access volumes and passes control, when necessary, to one of the other loads.

IGC0108B
is a load of the SVC 82 routine. It assigns an alternate track on a direct access volume.

IGC0208B
is a load of the SVC 82 routine. It updates UCBs to reflect new volume serials or VTOC location changes.

IGG019P8
is the End-of-Extent appendage routine. It modifies extent limits and file masks in DEBs.

IGG019P9
is the Abnormal End appendage routine. It is used to bypass I/O Supervisor error processing.

IEHMCVE

IEHMOVE
is the root segment; it obtains a save area.

IEHMVSRS
loads modules if required.

IEHMVXSE
gets three work files and a work area.

IEHMVXSF
is the first-time control module for IEHMVSSF.

IEHMVSSF (IEHMVSF)
mounts volumes.

IEFWMSKA (IEHMVSF)
is the systems device mask table.

IEHMVEST
clears work areas and initializes for a request.

IEHMVESJ
reads cards.

IEHMVSSS (IEHMVESS)
builds tables and sets switches.

IEHMVESI
opens the catalog for a data set group operation.

IEHMVESC
reads the catalog and writes it onto SYSUT1 for a data set group operation, or writes the catalog onto SYSUT2 for a move or copy catalog.

IEHMVESH
closes the catalog and sets up for following request.

IEHMVSSZ (IEHMVESZ)
checks for volume or data set.

IEHMVSSV (IEHMVESZ)
obtains 'FROM' DSCB, links to module for mounting of 'FROM' volume.

IEHMVMRZ (IEHMVESZ)
writes messages.

IEHMVSRZ (IEHMVESX)
handles routing and errors.

IEHMVSRV (IEHMVESX)
allocates the catalog on two volumes if necessary.

IEHMVSRK (IEHMVESX)
reads unloaded records.

IEHMVSRY (IEHMVEXV)
handles routing and errors.

IEHMVSSX (IEHMVEXV)
allocates two data sets.

IEHMVSTC (IEHMVEXV)
reads 'FROM' partitioned data set directory.

IEHMVMRY (IEHMVEXV)
writes messages.

IEHMVSSY (IEHMVESY)
handles routing and errors.

IEHMVSRM (IEHMVESY)
writes first unloaded record when applicable.

IEHMVSRX (IEHMVESY)
builds 'TO' and 'FROM' DCBs, handles 'TO' DD and 'FROM' DD.

IEHMVMSY (IEHMVESY)
writes messages.

IEHMVMRZ (IEHMVESY)
writes messages.

IEHMVETJ
reads 'FROM' and writes 'TO' sequential or partitioned data set without performing reblocking.

IEHMVESL
reads 'FROM' and writes 'TO' sequential or partitioned data set; reblocks type F records.

IEHMVESM
reads 'FROM' and writes 'TO' sequential or partitioned data set; reblocks type V records.

IEHMVSRD (IEHMVERD)
builds unloaded records.

IEHMVSRM (IEHMVERD)
writes unloaded records.

IEHMVSRA (IEHMVERA)
recreates unloaded record in original state.

IEHMVSRK (IEHMVERA)
reads unloaded records.

IEHMVSTA (IEHMVETA)
builds unloaded record and creates original record.

IEHMVSRM (IEHMVETA)
writes unloaded records.

IEHMVSRK (IEHMVETA)
reads unloaded records.

IEHMVMTA (IEHMVETA)
writes messages.

IEHMVESR
gets directory entries from SYSUT3 work file.

IEHMVETG
gets directory entries from SYSUT1 of includes or selects.

IEHMVESU
writes messages.

IEHMVESN
closes 'TO' and 'FROM' data sets; determines next module.

IEHMVMSN (IEHMVESN)
writes messages.

IEHMVESQ
catalogs and uncatalogs moved data sets.

IEHMVMSQ (IEHMVESQ)
writes messages.

IEHMVESP
catalogs and uncatalogs copied data sets.

IEHMVESO
checks errors - job abort or request.

IEHMVESK
closes SYSIN; scratches and closes SYSUT1, SYSUT2, and SYSUT3.

IEBISAM

IEBISAM
is the root segment; it sets up a common work area, obtains input parameters, sets switches, and passes control to the required module.

IEBISC
copies records of an indexed sequential data set.

IEBISU
retrieves logical records sequentially from an indexed-sequential data set.

IEBISSO (IEBISU)
creates 80-byte logical records with fields as defined for 'unloaded' data sets.

IEBISL
reconstructs indexed-sequential data set from 'unloaded' data.

IEBISSI (IEBISL)
retrieves logical records from an 'unloaded' data set.

IEBISPL
prints logical records of an indexed sequential data set.

IEBISF
writes messages, prints error messages if applicable, and returns completion code to root segment.

IEHPROGM

IEHPROG1 (IEHPROGM)
gets work area, reads SYSIN, mounts volumes if applicable.

IEHPROG2 (IEHPROGM)
issues SVCs for cataloging, uncataloging, deleting, connecting, releasing, BLDA, DELET.

IEHPROG3 (IEHPROGM)
contains and writes messages.

IEHPROG4 (IEHPROGM)
opens input and output DCBs.

IEHPROG5 (IEHPROGM)
prepares for the volume mounting module IEHMVSSF.

IEHLIST

IEHQSCAN (IEHLIST)
reads control cards.

IEHPRMSG (IEHLIST)
message module.

IEHPRINT (IEHLIST)
scans and prints requested data from VTOCs, catalogs, and directories.

IEBUPDAT

IEBUPDAT
updates 80-character logical record libraries.

IEBUPDTE

IEBUPDT2 (IEBUPDTE)
creates partitioned or sequential data sets, sequences new data sets, resequences old data sets, replaces or reproduces data set members, or adds members to a partitioned data set.

IEBUPLOG (IEBUPDTE)
opens SYSPRINT and writes messages.

IEBUPDTE
reads control cards, and opens SYSUT1 and SYSUT2.

IEBASCAN (IEBUPDTE)
scans and analyzes control statements and sets appropriate flags.

IEBUPNIT (IEBUPDTE)
initializes the region IEBUPCON and opens SYSIN data set.

IEBUPXIT (IEBUPDTE)
contains exit routines for the program.

IBCDMPRS

IBCDMPRS
creates backup copies of direct access volumes.

IBCRCVRP

IBCRCVRP
recovers usable data from a defective track, assigns an alternate track, and merges replacement data with the recovered data onto the alternate track.

IBCDASDI

IBCDASDI
initializes and assigns alternate tracks to a direct access volume.

IFCEREP0
modules for this utility program are summarized in Figure 25.

IEBDG

IEBDG
is the control module that is the interface with a calling program. It opens the input, output, and message data sets, and it reads the program's control cards.

IEBFDANL
analyzes the keywords and parameters on an FD card and begins construction of an entry in the FD table.

IEBFDTBL
completes the construction of the FD entry that was begun by the FD analysis module. It assigns FD card default values if necessary.

IEBCRANL
analyzes the keywords and parameters on a CREATE card and builds a create table entry, a picture table, an FD address table, and an exit name table.

IEBCREAT
generates output records by using information from (1) input data sets, and (2) tables built by previous modules, as required. It permits user modifications before final record output. It releases storage obtained for information tables.

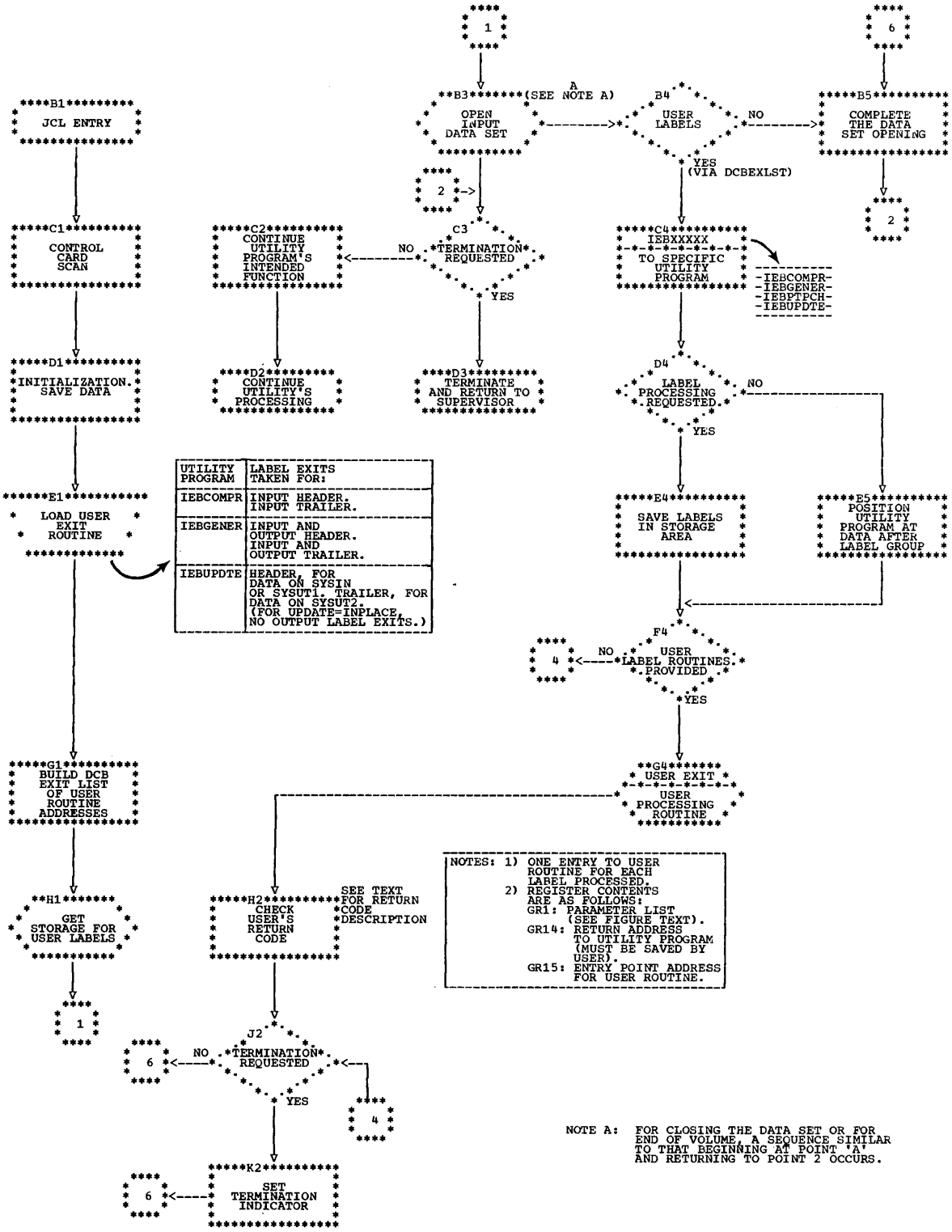
IEBDGMSG
is the message module, and it controls the paging on a message printer.

IEBDGCUP
is the clean-up module that closes DCBs and frees storage for DCEs and buffer pools.

Appendix B: User-Label Processing

With respect to the processing of user labels by user routines, Figure 68 shows the general logic of the following utility programs: IEBCOMPR, IEBGENER, IEBPTPCH, and IEBUPDTE.

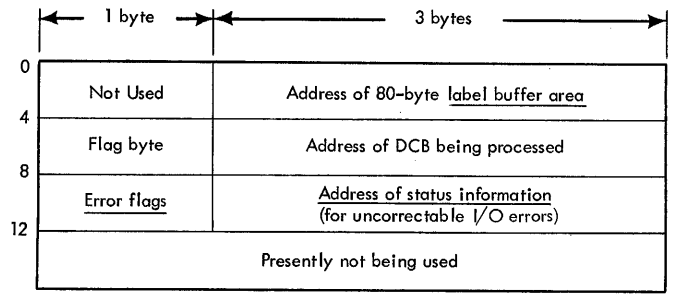
The following text discusses parameter information passed from a utility program to a user routine, and return code information passed from a user routine to a utility program.



• **Figure 68. General Logic of Utility Program With User Label-Processing Routine Exits**

Parameter List

When the utility gives control to a user label-routine, general register 1 contains the address of a parameter list whose format is given in Figure 69.



• Figure 69. Parameter List Passed to User-Label Exit Routine

A description of the underlined fields indicated by the parameter list in Figure 69 is given below.

- label buffer area: prior to entering a label routine, user header or trailer labels are read into this area by the operating system. When a user's label routine constructs labels, the labels are placed (one at a time) in this area.

- status information address: if an uncorrectable I/O error occurs during the reading or writing of a user label, bit 0 of the high-order (error flags) byte of this field is set to 1. The three low-order bytes of this field contain the address of the standard status information for SYNAD routines. (See the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646.)

Note: At volume switch time, the utility routines use the information contained in the flag byte of the second word to indicate end of volume or end of data.

PARAMETER LIST MODIFICATION

For IEBUPDTE, the following modifications are made to the parameter list:

- When there are user label-processing routines, the first meaningful field of the parameter list passed to the user output-label routine points to the label buffer. This buffer, which contains a label data record from the SYSIN data set, is for the user to inspect before the record is written as a label.

- If the error status information in the parameter list is established as a result of a reading error, the user routine must return one of the return codes (described in the next section) or the program will be terminated.

- If the error status information is established as a result of a recording error, bit 1 (of the error-flags byte) is set to 1 to indicate that the error occurred during an output operation. In this case, the user routine must return a code of either 0 or 4, or the program will be terminated.

- For header labels only, a fifth entry in the parameter list occurs under the conditions given below. The first byte of this entry is meaningless, and the last three bytes contain the address of the label that has been replaced from the old master data set (SYSUT1). The conditions (all of which must be present) for the occurrence of the entry are:

  1. An update of the old master is specified via the keyword UPDATE=INPLACE.

  2. A LABEL statement must be specified for header labels in the input data set.

  3. A user label-routine corresponding to the LABEL statement is specified and user labels are encountered on SYSUT1.

Return Codes

One of the following return codes must be placed in general register 15 when a user (label-processing) exit routine gives control back to the utility program. An incorrect (or no) code results in termination of the program.

Type of Routine	Code	System (Utility) Response
Input header or input trailer label	0	Resume normal processing. Ignore additional labels in the label group.
	4	Read next user label into buffer area. Return control to user-exit routine. Resume normal processing if no more labels.
	16	Request termination of label processing. Utility program performs clean-up functions and terminates.
Output header or output trailer label	0	Resume normal processing. No label is written from buffer area.
	4	Write label from buffer area. Resume normal processing.
	8	Write label from buffer area. If less than eight labels created, return to exit routine. Otherwise, resume normal processing.
	16	Request termination of label processing. Utility program performs clean-up functions and terminates.

RETURN CODE MODIFICATIONS

1. For IEBUPDTE, the following modifications are made to the return codes when the keyword UPDATE=INPLACE is specified.

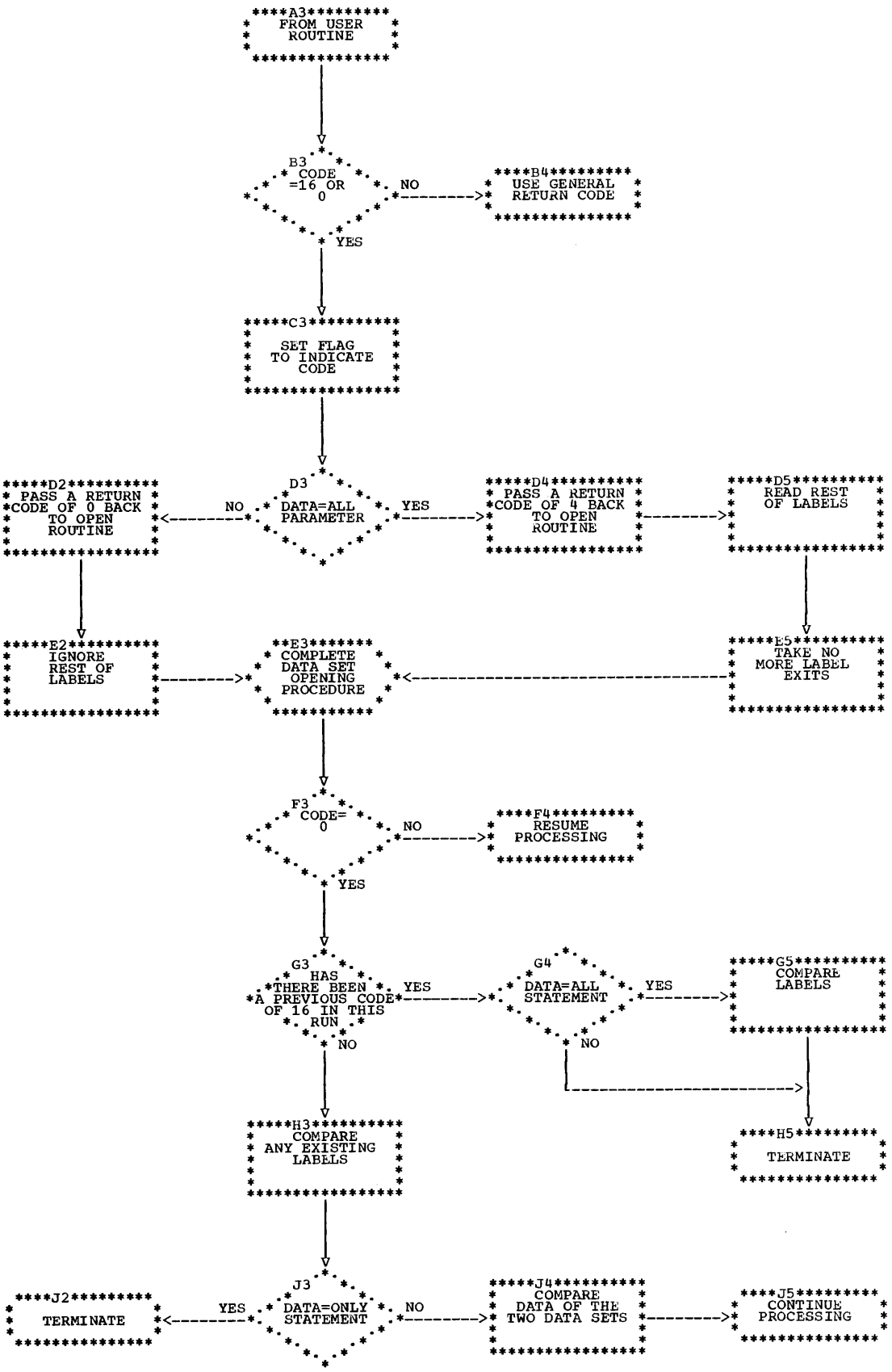
Type of Routine	Code	System (Utility) Response
Input header	0	Same as above.
	4	Same as above.
	8	Write label from buffer area. Resume normal processing.
	12	Write label from buffer area. Read next label into buffer area. Return control to user exit routine. Resume normal processing if no more labels.
	16	Request termination of label processing. Utility program performs clean-up functions and terminates.

2. For IEBCOMPR, the following modifications are made to the return codes, depending on the operand in the LABELS statement: (See Figure 70)

Type of Routine	User Return Code	LABELS Statement	System (Utility) Response
Input header or trailer Labels	16	DATA = ALL	Return a code of 4 to Open routine. Take no additional label exits.
	16	DATA ≠ ALL	Return a code of 0 to Open routine. Ignore rest of labels.
	0*	DATA = ALL	Same as for code 16.
	0*	DATA ≠ ALL	Same as for code 16.

*After SYSUT1 and SYSUT2 have been opened, the following conditions are tested and the response indicated is taken.

0, with a previous code of 16	DATA = ALL	Compare the labels, then terminate the processing.
	DATA ≠ ALL	Terminate the processing.
0, with no previous code of 16		Compare available labels. Then check LABELS statement operand as follows:
	DATA = ONLY	Terminate the processing.
	DATA ≠ ONLY	Compare the data of the appropriate data sets.



•Figure 70. Return Code Modification for IEBCOMPR Program

Index

IBM Technical Newsletter

File Number	S360-30
Re: Form No.	Y28-6616-4
This Newsletter No.	Y28-2363
Date	February 1, 1969
Previous Newsletter Nos.	None

IBM SYSTEM/360 OPERATING SYSTEM
INPUT/OUTPUT SUPERVISOR
PROGRAM LOGIC MANUAL

This Technical Newsletter, a part of release 17 of IBM System/360 Operating System, provides replacement pages for Input/Output Supervisor Program Logic Manual, Form Y28-6616-4. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be inserted and/or removed are listed below.

65,66

A change to the text or a small change to an illustration is indicated by a vertical line to the left of the change; a changed or added illustration is denoted by the symbol • to the left of the caption.

Summary of Amendments

Changes have been made to correct printing errors.

Note: File this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Programming Systems Publications, P.O. Box 390, Poughkeepsie, N.Y. 12602

PRINTED IN U.S.A.

When processing is complete, the DEVTYPE routine places a return code in general register 15. An error return code of 04 indicates one of the following conditions:

- No output area specified: the area parameter is missing from the DEVTYPE macro instruction.
- DD name not found: there is no TIOT entry that corresponds to the DD name supplied.
- Invalid UCB unit type field: the UCB unit type field (byte 4 of the device code field) does not specify a direct access, tape, or unit record device.

If the request is completed satisfactorily, the DEVTYPE routine places a return code of 00 in general register 15, and exits via SVC 3.

IOHALT ROUTINE

When the IOHALT routine (SVC 33) receives control, it is given the address of the UCB associated with the device to be stopped. It checks that address for validity, then inspects the UCB device code field to make sure that the device is not a direct access device.

IOHALT branches to a resident I/O Supervisor subroutine to issue the HIO instruction and examine the resulting condition code. If no error occurs, control returns to IOHALT.

The appropriate condition code is placed in general register 15. If the operation was successful, the IOHALT routine places a post code (X'48') in the ECB code field of the IOB and issues SVC 3 to exit.

SECTION V: CONTROL BLOCK AND TABLE FORMATS

The I/O supervisor uses control blocks and tables to communicate with itself, with the rest of the control program, with processing programs, and with I/O devices.

This section of the publication describes in detail the characteristics of the control blocks and of the tables used by the I/O supervisor.

Figures 8 through 21 show control block and table formats. Where pertinent, a discussion of the fields follows the figures.

For convenience, the following control blocks and tables are presented in alphabetical order:

- Attention Table
- Channel Search Table
- Data Control Block
- Data Extent Block
- Device Table
- Error Recovery Procedure Interface Bytes (ERPIB)
- Event Control Block
- Input/Output Block
- Logical Channel Word Table
- Request Element Table
- Statistics Table
- UCB Lookup Table
- Unit Control Block

ATTENTION TABLE

The attention table is used by the I/O supervisor to obtain the addresses of the attention routines required to service the I/O devices attached to the system.

The attention table has the following characteristics:

- Creation. The table is created at system generation time.
- Storage Area. The table resides, as a permanent part of the resident supervisor, in protected resident storage (when protection is available).
- Size. The table contains one 4-byte entry per attention routine, up to a maximum of 64 entries.
- Means of Access. The ATNTAB byte, supplied by the user in the UCB, is added to the starting address of the attention table to obtain the proper attention routine entry in the table for the device.
- Format. The format of an attention table entry is shown in Figure 8.

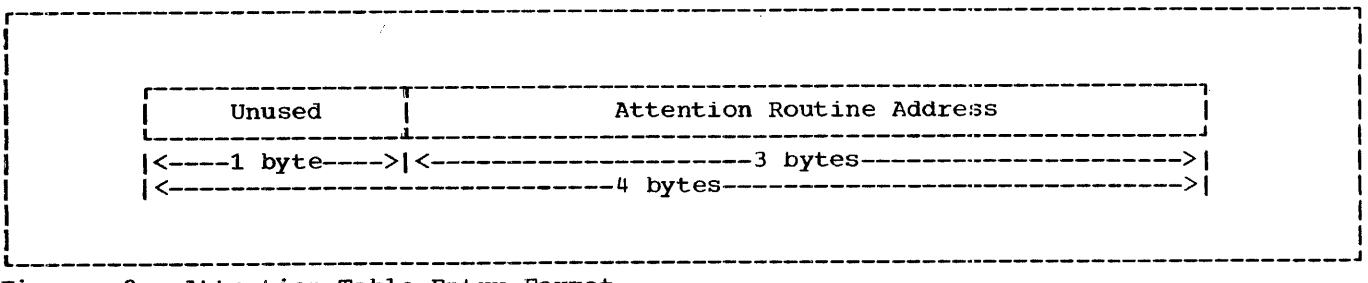


Figure 8. Attention Table Entry Format

READER'S COMMENT FORM

IBM System/360 Operating System
Utilities
Program Logic Manual

Form Y28-6614-4

- Is the material:

	Yes	No
Easy to read?	☐	☐
Well organized?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Accurate?	☐	☐
Suitable for its intended audience?	☐	☐

- How did you use this publication?
  - ☐ As an introduction to the subject
  - ☐ For additional knowledge

  Other .....

- Please check the items that describe your position:
  - ☐ Customer personnel
  - ☐ IBM personnel
  - ☐ Manager
  - ☐ Systems Analyst
  - ☐ Operator
  - ☐ Programmer
  - ☐ Customer Engineer
  - ☐ Instructor
  - ☐ Sales Representative
  - ☐ Systems Engineer
  - ☐ Trainee

  Other .....

- Please check specific criticism(s), give page number(s), and explain below:
  - ☐ Clarification on page(s) .....
  - ☐ Addition on page(s) .....
  - ☐ Deletion on page(s) .....
  - ☐ Error on page(s) .....

Explanation:

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

YOUR COMMENTS PLEASE . . .

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

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

Fold

Fold

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM Corporation
P.O. Box 390
Poughkeepsie, N.Y. 12602

Attention: Programming Systems Publications
Department D58

Fold

Fold

IBM S/360 Printed in U.S.A. Y28-6614-4

IBM®

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

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

IBM S/360 Printed in U.S.A. Y28-6614-4

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

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