

## **Systems**

# **IBM 3735 Programmable Buffered Terminal Form Description Macro Instructions and Form Description Utility Program Logic Manual (OS and DOS Systems)**

**Program Numbers OS 360S-CQ596  
DOS 360N-CQ490**

# **IBM**

## **Preface**

The How to Use This Book section of this publication defines the audience for which this program logic manual is intended, explains how the book is organized, and suggests how the reader may best familiarize himself with its contents.

### *First Edition (March 1972)*

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM equipment, refer to the latest SRL Newsletter for the editions that are applicable and current.

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

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be sent to IBM Systems Development Division, Publications Center, Department E01, P.O. Box 12275, Research Triangle Park, North Carolina 27709. Comments become the property of IBM.

## Contents

<b>How to Use This Book</b> . . . . .	iv
<b>Part 1. Introduction to the IBM 3735 Programmable Buffered Terminal</b> . . . . .	1-1
<b>Part 2. Logic of the Form Description Macro Instructions</b> . . . . .	2-1
<b>Part 3. Logic of the Form Description Utility</b> . . . . .	3-1
<b>Part 4. Appendixes and Glossary</b> . . . . .	4-1
<b>Index</b> . . . . .	X-1

## How To Use This Book

This Program Logic Manual describes in detail the internal specifications and logic of the IBM System/370 and System/360 Operating System (OS) and Disk Operating System (DOS) programming support for the IBM 3735 Programmable Buffered Terminal (hereafter referred to as the 3735). The 3735 programming support includes both Form Description (FD) macro instructions and Form Description utility programs to provide the operating environment for applications using preprinted (fixed-format) forms and batch processing.

This publication is divided into four major parts: Part 1 gives an overall introduction to the 3735; Part 2, the Form Description macros information; Part 3, the Form Description utility information; and Part 4 contains the rear matter of the appendixes and glossary. The 3735 FD macros and FD utility information contained in Parts 2 and 3 is directed to the IBM program systems representatives and system engineers who provide program maintenance and who need information about the internal organization and logic of the FD macros and FD utility.

Because the OS and DOS programming support for the 3735 are so similar, both are discussed as one. Where significant differences exist, they are explained as required.

The second part of this book, dealing with the macros, is subdivided into three sections containing the following information.

Section 1 is the Introduction to the FD macros. The general information presented in the introduction is basic to an understanding of the macros and includes descriptions of the following: (1) the macros themselves, (2) the general purpose of the macros, (3) the main functions of the macros, and (4) the system environment required to assemble the macros for both OS and DOS.

Section 2, Method of Operation, describes the overall FD macro structure and the macro functions. It discusses in detail each function and the method used to accomplish that function. When appropriate, diagrams depict visually the concept being described.

Section 3, Form Description Program Organization, describes the detailed logical organization of the FD programs that result from the assembly of FD macro instructions. This section notes specific OS and DOS dependencies as they are required.

The third part of this book dealing with the FD utility, is subdivided into six sections containing the following information.

Section 1, Introduction to the FD utility, contains general information that is basic to an understanding of the utility and includes descriptions of the following: (1) the utility, (2) the general purpose of the utility, (3) the main functions of the utility, (4) the system environment required to execute the utility for both OS and DOS.

Section 2, Method of Operation, describes the overall logical flow and functions for the utility. It discusses in detail each function and the method used to accomplish that function. When appropriate, diagrams depict visually the concept being described.

Section 3, Program Organization, describes the detailed logical organization of the FD utility and includes flowcharts to illustrate the logical flow of the utility. This section denotes specific OS and DOS dependencies as they are required.

Section 4, Directory, defines the CSECT names and other references used for the FD utility. It also provides cross-references applicable for the utility program listings. All this information is presented in charts and tables.

Section 5, Data Area Layouts, refers back to Section 2 and Section 3, which describe graphically and verbally the format and contents of the various data areas used by the FD utility.

Section 6, Diagnostic Aids, is not provided because of the simplicity of the programs.

The fourth major part of this publication contains four appendixes and a glossary of definitions for the technical terms used in this book.

To understand the logic of the 3735 programming support, the reader must have a general understanding of the System/370 or System/360 Operating System or Disk Operating System and of the macro language facility of the assembler. In addition, he should also be familiar with the following prerequisite publications:

*IBM 3735 Programmable Buffered Terminal*

*Concept and Application*, GA27-3043

*Programmer's Guide*, GC30-3001

*IBM System/360 Operating System Assembler Language*, GC28-6514

*IBM System/360 DOS, TOS Assembler Language*, GC24-3414



## **Part 1. Introduction To The IBM 3735 Programmable Buffered Terminal**

# Contents

<b>IBM 3735 Programmable Buffered Terminal</b> . . . . .	1-3
Configuration . . . . .	1-3
Program Support . . . . .	1-3
System Considerations . . . . .	1-4
Forms Design . . . . .	1-4
Form Descriptions . . . . .	1-4
Form . . . . .	1-4
Page . . . . .	1-4
Line . . . . .	1-5
Field . . . . .	1-5
Control . . . . .	1-5
Programming Considerations . . . . .	1-5
Form Description . . . . .	1-5
Assembly of Macro Instructions . . . . .	1-5
Utility . . . . .	1-6
Transmission . . . . .	1-6

## The IBM 3735 Programmable Buffered Terminal

The IBM 3735 Programmable Buffered Terminal provides an effective means of manipulating the data required for day-to-day business operations. The 3735 brings many of the capabilities of the central computer to the operator, thereby allowing the terminal to automatically provide operator guidance, error detection, formatting, editing, and other services.

The IBM 3735 Programmable Buffered Terminal is a programmable terminal consisting of the IBM Selectric® I/O II Keyboard Printer and a highly sophisticated desk-side control unit. It is designed primarily for applications that use preprinted (fixed-format) forms and batch processing.

The 3735 uses two levels of program control. First, the user generates Form Description (FD) programs, via the FD macros and the FD utility, to specify the functions desired for processing a specific type of form. Second, a microcoded terminal control program, recorded in the terminal control unit during manufacture, interprets the FD programs and provides detailed terminal control.

The Form Description (FD) macros assembled by the host operating system describe the forms to be processed. The Form Description (FD) utility arranges the forms for later transmission to the 3735 via the user's application program and teleprocessing access method. The FD programs reside in the 3735 terminal until replaced by the user.

### CONFIGURATION

The 3735 is a supported device for the IBM System/370 Model 135 and up and for the IBM System/360 Model 22 and up. The basic 3735 terminal configuration provides for communication over point-to-point switched (dial-up) or nonswitched (with multipoint control) common-carrier facilities at speeds of 1200 or 2000 bits per second. (World Trade provides for a 600 bits-per-second transmission speed.) Communication over multipoint nonswitched (leased) lines at 1200, 2000, or 2400 bits per second can be provided by the addition of a special feature to the basic 3735 terminal.

The basic configuration of the 3735 consists of an IBM Selectric® I/O II Keyboard Printer and desk-side control unit. The control unit consists of (1) a small processor, (2) a binary synchronous communication (BSC) line adapter, and (3) a non-removable magnetic-disk storage device. The magnetic-disk storage device within the control unit contains (1) the terminal control program, (2) the FD programs, (3) an inquiry message or response, (4) user-generated

data records, and (5) (optionally) a CPU ID list. The number of FD programs, each representing a different form type, that can be stored depends on the length of the programs. Basic storage capacity for user FD programs and data storage is about 62K bytes. This basic capacity is expandable in increments of 42K bytes to a total of about 145K bytes.

This storage capacity is provided in 22K-byte tracks, each of which is divided into forty seven 480-byte sectors. The basic 3735 terminal has three tracks for storing the FD programs and user data. Three sectors of each of these tracks are reserved for control purposes, and four bytes of each of the remaining sectors contain chaining information. In the sectors that store FD programs, 234 bytes contain the programs and six bytes contain chaining information. The remaining 240 bytes of FD program sectors on disk are not used. When in the buffer, the terminal control program uses the second half of these bytes as work space.

Three optional features provided for the 3735 are the IBM 5496 Data Recorder, the IBM 3286 Printer Model 3, and the IBM Operator Identification Card Reader attachments. The 5496 is a buffered operator-oriented, key-entry unit used to punch, interpret, and read the 96-column punched card. The 3286-3 is a printer adapter used as a secondary printing device. The identification card reader reads magnetically recorded data from the IBM 16-character ID card (standard credit card size 2-1/8" x 3-3/8") that has a magnetic stripe and from the other IBM credit card with data up to 39 characters.

### PROGRAM SUPPORT

The IBM System/370 and System/360 support the 3735 under the Operating System (OS) and the Disk Operating System (DOS). This support provides (1) for generating FD programs, (2) for preparing these programs for transmission to the terminal, and (3) for transmitting data between the computer and the terminal.

The 3735 terminal uses the binary synchronous method of communications line control (BSC), making the terminal compatible with most BSC systems and programs. Since the 3735 is a BSC device, the binary synchronous support that exists in OS TCAM and BTAM and in DOS BTAM is sufficient to handle the transmission of data between the CPU and the terminal.

The FD programs are generated using System/370 and System/360 OS and DOS Assembler Language macro instructions. These FD macro instructions provide error checking for the user.

A Form Description utility is provided to prepare the FD programs for transmission to the 3735. The FD programs are placed in an output data set for selection and transmission by the user's application program. When the terminal receives these programs, they reside on the 3735 disk and the operator uses them to control terminal functions. The FD utility operates under OS and DOS. It operates independently of the user's teleprocessing program but is dependent upon the assembly of the FD macro instructions. The utility is scheduled through the input job stream.

The minimum system that can use the FD macros and the FD utility is:

OS/DOS – System/370 Model 135

OS – System/360 Model 40, 128K

DOS – System/360 Model 22, 32K

*Note:* DOS requires 32K bytes of storage to support teleprocessing.

## SYSTEM CONSIDERATIONS

The following paragraphs outline functions that can be specified for processing a given form and discuss the formatting of data transmitted to and received from the CPU.

### Forms Design

The 3735 places few restrictions on forms design. The standard platen accommodates forms up to 15 inches wide with up to 130 characters per line. Printing is at ten characters per inch; vertical spacing is at six lines per inch. An original and up to four carbon copies can be printed.

### Form Descriptions

Form descriptions should be developed sequentially from left to right, top to bottom, and page by page, since forms are usually processed in this manner during preparation.

The 3735 form descriptions are presented in the following sequence:

*Form:* Descriptions that apply to the overall form.

*Page:* Descriptions that apply to a single page of the form.

*Line:* Descriptions that apply to a single line of the form.

*Field:* Descriptions that apply to a single field of the form.

*Control:* Descriptions that define the status checking and processing to be done on the form.

### Form

The form description (specified by using the FDFORM macro) must name the form and specify a three-digit identifier. The terminal operator uses this identifier to call out the form description program for processing a document.

The user can specify an operator message for the form. This message is included within the body of the FD program and can be requested by the terminal operator before beginning processing under FD program control. The message may include instructions for document preparation, machine setup, application name, and so forth.

The form description can indicate the tab settings to be used during processing of the form. Each tabular stop is indicated by listing the character position (relative to the form) at which the stop is to be set. The terminal operator can perform a tab set routine, under FD program control, before starting to process forms under this FD program.

The form description can specify the left margin stop that is to be used when processing the form. The margin stop is indicated by specifying, relative to the form, the character position in which the terminal's left mechanical margin stop is to be set. The position (n) may range from 0 to 129; the default is 0. Actual output operation can begin in position n+1.

The form description can indicate the format of data records created by an FD program before being sent to the central computer. This format can be described via one of two packing (condensing) functions: one that deletes consecutive trailing blanks and one that both deletes consecutive trailing blanks and inserts a separator character between data fields. If the data record format is to be sent to the CPU in its entirety, the form description can indicate that no packing function is to be performed. The packing option selected remains in effect throughout the processing of the entire form.

The form description can indicate that a 96-character read buffer and a 96-character punch buffer are to be treated as a single 192-character read/punch buffer for temporary data storage for the optional 5496 Data Recorder. If the read/punch buffer is used for either a read or a punch operation for the 5496, the form description can indicate explicitly the location in the buffer to be read or punched. The form description can also vary the checking of the 3286-3 line printer buffer. The line printer buffer can also be used to print on the 3286-3 provided the location to be written from is specified explicitly in the form description.

The form description also can indicate that the user may specify Katakana instead of ASCII code in processing forms.

### Page

The page description (specified by using the FDPAGE macro) provides identification and size information about a single page of the form. The size information is used during generation of the FD program to ensure that the fields and lines of the form will fit the page size intended. If the form consists of more than one page, each individual page must be described. If a single page is used, the page description can be included with the form description. Line and field descriptions for the page follow each page description.

Each page description identifies the page being described by number and optionally by name. Page numbers are normally in ascending sequence but need not be consecutive.

The overall height of the page is described by indicating the total number of usable line positions on the page; for example, a page 11 inches long has a height of 66 (at six lines per inch).

Vertical margins indicate the first line and the last line which may be defined. Vertical margins are indicated by line number, counting from the top of the page; for example, if the first line on a form may be the fourth line from the top, and the last line may be the 64th line, the vertical margins are 4 and 64.

### *Line*

Line descriptions (specified by using the FDLINE macro) describe each line to be processed within a page. A line description, identified by a line number (represented by a number from 1 to n for each page on a Selectric or by an accumulative number for a page on a matrix printer), should be provided for each line to be processed by the 3735.

The line description defines the line's horizontal space requirements. Each line description contains a line identification, width, and horizontal margins. If the width and horizontal margins of all lines on a page are the same, these descriptions can be included with the page description.

If FD program control for a series of lines is identical, the series can be designated for cycling. Repeating line and field descriptions for each line is then unnecessary. Describe the first line or group of lines and indicate the number of repetitions. In addition, indicate the part of the form description program to be processed following the cycle. This cycle control is desirable for two reasons: it shortens the FD program, thus saving storage space on the 3735 disk; and it allows the operator to terminate cyclic operation if all lines are not required on a specific transaction.

### *Field*

Field descriptions (specified by using the FDFIELD macro) follow each line description and specify the control desired for each field of the line. Many commonly used functions (such as, centering data within a field, comma and decimal point insertion, and self-check number verification) have been implemented within the terminal control program.

The field's horizontal position on the line identifies each field description. This field is expressed as the first (left) and last (right) character positions of the field or, optionally, as the first character and length. Maximum field size is 127 characters.

Field descriptions tell where the data comes from (source), what is done with it (operation), and where it is to go (sink). The operator can specify a variety of checking functions for the input data. The application program processes the source data as required by using arithmetic and

compare functions. Several devices can be specified for the destination (sink) of data from the specified source:

- the Selectric® I/O II printer
- an inquiry buffer (INQ)
- the IBM 5496 Data Recorder
- the IBM 3286 Printer Model 3

The Selectric® I/O II keyboard and the IBM Operator Identification Card Reader are also supported devices for the origination (source) of data.

### *Control*

This description (specified by using the FDCTRL macro) provides for (1) testing of logical conditions (indicators), (2) modifying the content of indicators and counters, and (3) executing commands and nonsequential operations. Use of these logical and arithmetic functions can implement such functions as field skipping. The forms designer should define the function desired and the data available; this allows the macro encoder to implement an FD program that provides exactly the functions desired. The control description also includes card reading and punching instructions. A read card instruction signals the 5496 card reader to read a card and the data enters a card-image buffer where it is available for use by the FD program. Similarly, a punch card instruction signals the 5496 punch to accept data from a card-image buffer, and punch the data into a card.

## **PROGRAMMING CONSIDERATIONS**

### **Form Description**

#### *Assembly of Macro Instructions*

The System/370 and System/360 OS and DOS Assembler Language macro instructions provide the user with a means of describing his forms. The output of the system assembly is an object module suitable for input to the FD utility. Also produced is an assembly listing that includes diagnostic and error information concerning the macro instructions. The assembly provides the following functions:

- Check input statements for completeness and accuracy.
- Provide meaningful comments to aid in debugging the program.
- Calculate sequences of motion control characters for proper print-element positioning.
- Sequence-number output records (object module).
- Set flags in the output if the input statements contain errors that invalidate the output.

The output of the assembly is object modules whose data portions are unpacked FD programs. These object modules

can be punched in cards or written in a data set as card images, then used as the input to the FD utility.

### *Utility*

The FD utility prepares the unpacked FD programs for transmission to the 3735 terminal. The utility operates basically the same under OS and DOS. It reads the output of the assembly from a card reader or equivalent sequential input device, checks for program integrity, and arranges the unpacked FD programs into blocks of 476 bytes. Then the utility writes these unpacked FD program blocks into a user-specified data set that is available to the user's application program.

The main storage required by the OS FD utility is no more than that required for the minimum OS Linkage Editor available to the system. The main storage required by the DOS FD utility exclusive of the Linkage Editor step is no more than 12K bytes. The following three I/O devices are required: a card reader or equivalent device; a printer; and a direct-access storage device (DASD). The utility uses no more than 10 tracks of 2311 storage, or the equivalent, for program residence. The user's secondary storage requirements depend on the number, size, and complexity of the forms being described.

The FD utility is scheduled through the input job stream, encompassing three sequential job steps: Control, Linkage Editor, and Storage. There is one optional feature: in OS, the JCL PARM feature; in DOS, the RPLACE control card.

In OS, if a form is a duplicate (one with the same form name) of a member in the user's output data set and the REPLACE option has not been specified, the form is stored as a new member under a temporary name: for example, IDFTMP0, IDFTMP1, IDFTMP2, . . . , IDFTMP9. If these temporary names have already been exhausted, the new member is not stored. The form is replaced if the REPLACE option has specified that all duplicates are to be replaced, or has specified, by name, the forms to be replaced. The action taken is noted in the listing.

In DOS, if a form is a duplicate of a member in the user's output data set and the RPLACE card has not been used, the form is stored as a new member under a temporary name: for example, IJLFTM00, IJLFTM01, IJLFTM02, . . . , IJLFTM09. If these temporary names have already been exhausted, the new member is not stored. The form is replaced if the RPLACE card has specified that all duplicates are to be replaced, or has specified, by name, that the form be replaced. The action taken is noted in the listing.

To include the FD utility in his operating system, the user must copy it from the component library on which it is distributed. This may require allocating additional space. Prior to the execution of the FD utility, the system programmer must ensure that suitable input and output data sets are designated through correct coding of the job control statements.

If an uncorrectable error is encountered, the FD utility takes the appropriate action, depending on which step of the three-step sequence is being executed. The control step terminates the job step; the storage step deletes a partially created sequence of blocks from the user's data set. Each of the three steps produces a diagnostic listing. The listing produced by the last step names each unpacked FD program that was added to, deleted from, or not added to the user's data set.

The response to environmental errors arising from improper input, such as a card missing or out of sequence, is to write a message in the diagnostic listing, produced in the control step, and to exit. Input following the erroneous card is not processed. The response to errors such as insufficient allocation of space in a data set is to terminate processing and to note the condition in the listing. The response to implementation errors, such as errors in system control blocks, is those error recovery and termination options provided by OS and DOS.

The diagnostic listing produced from the execution of each of the three utility job steps contains information of interest to the programmer adding, creating, modifying, or extending a set of unpacked FD programs and includes a list of messages.

Following are some examples of the types of diagnostics noted in the listing:

- Successful completion
- Last valid record (card) number
- Erroneous record (card) number
- Invalid sequence number
- Invalid deck ID
- Premature end-of-file
- Invalid card type
- Name of FD program stored
- Duplicate name, if it was stored and if so, under what name
- Insufficient space

The messages that are listed vary slightly between OS and DOS. The programmer response to any errors indicated is to correct the error and re-execute the job. Another alternative is for the utility to note the error and continue processing the next valid data group.

### *Transmission*

The output of the FD utility consists of 476-byte blocks containing macro-generated bit strings forming the data portion of FD messages. These FD programs containing the unpacked code that is interpreted at the 3735 reside in a user's data set. The user must create a teleprocessing application program to select and transmit the FD programs, just as he must create application programs to process the data captured and transmitted by the 3735 terminals.

Not all FD programs in the user's library must be transmitted to all 3735 terminals during FD program transmission to individual terminals. The user can be selective, sending FD programs from the library to specific terminals. However, during transmission of the FD programs from the user's data set to the terminal, all the FD programs that are to reside at a terminal must be transmitted at the same time. FD programs cannot be added to those already residing at the terminal. To get the total number of FD programs desired at the terminal, the FD programs already there must be retransmitted along with any additional ones.

The FD programs must be transmitted according to the following scheme:

1. The 3735 terminal transmits its unpacked data to the CPU first.
2. The CPU must transmit a user-prepared block which informs the 3735 that the following transmission blocks are FD programs.
3. The CPU transmits the FD programs in 476-byte unpacked blocks.
4. The CPU must transmit a user-prepared block that informs the 3735 that transmission of FD programs is complete.





## **Part 2. Logic Of The Form Description Macro Instructions**

## Contents

<b>Part 2. Objectives</b> . . . . .	2-3	FDPAGE Macro Instruction . . . . .	2-19
<b>Section 1. Introduction</b> . . . . .	2-4	FDLINE Macro Instruction . . . . .	2-19
Form Description Macro Instructions . . . . .	2-4	FDFIELD Macro Instruction . . . . .	2-19
The Structural Macros . . . . .	2-4	FDCTRL Macro Instruction . . . . .	2-22
The Procedural Macro . . . . .	2-4	FDEND Macro Instruction . . . . .	2-23
The Delimiting Macro . . . . .	2-5	The Assembly of Form Description Programs . . . . .	2-24
The Trace Macro . . . . .	2-5	Form Description Program Header Assembly . . . . .	2-24
The Display Macro . . . . .	2-6	Address Resolution and Motion Control Assembly . . . . .	2-26
Form Description Macro Organization . . . . .	2-6	Immediate Byte Assembly . . . . .	2-27
System Requirements . . . . .	2-6	Field Control Descriptor Assembly . . . . .	2-28
Operational Considerations . . . . .	2-7	<b>Section 3. Form Description Program Organization</b> . . . . .	2-31
Input and Output of the Form Description Macro Assembly . . . . .	2-7	Form Description Programs . . . . .	2-31
Input . . . . .	2-7	FD Program Header . . . . .	2-31
Output . . . . .	2-7	Immediate Bytes and Field Control Descriptors . . . . .	2-32
<b>Section 2. Method of Operation</b> . . . . .	2-12	Immediate Bytes . . . . .	2-32
Form Description Macro Structure . . . . .	2-12	Field Control Descriptors . . . . .	2-39
FD Macro Functions . . . . .	2-19	FD Program Trailer . . . . .	2-50
FDFORM Macro Instruction . . . . .	2-19	FD Program Maintenance . . . . .	2-50
		Trouble Shooting . . . . .	2-50

## Illustrations

<i>Figure</i>	<i>Title</i>	<i>Page</i>	<i>Figure</i>	<i>Title</i>	<i>Page</i>
2-1	3735 Program Logic Flow . . . . .	2-5	2-19	The FD Program Header . . . . .	2-31
2-2	OS FD Macro Assembly Output . . . . .	2-8	2-20	NOP, GOTO, and Conditional GOTO Immediate Bytes . . . . .	2-33
2-3	DOS FD Macro Assembly Output . . . . .	2-9	2-21	The Begin, Repeat, and End Cycle Immediate Bytes . . . . .	2-35
2-4	OS Control Section Format . . . . .	2-10	2-22	The Index Space and Total Batch Immediate Bytes . . . . .	2-36
2-5	DOS Control Section Format . . . . .	2-11	2-23	The Cancel Form, End Form, Clear Counter, and Set Indicator Immediate Bytes . . . . .	2-36
2-6	Keyed Form Description Unpacked Program Block (KUPB) . . . . .	2-11	2-24	The Selectric Command Immediate Bytes . . . . .	2-36
2-7	Hierarchy of FDFORM Macro Calls . . . . .	2-13	2-25	The Clear STG and Inquiry Command Immediate Bytes . . . . .	2-38
2-8	Hierarchy of FDPAGE and FDLINE Macro Calls . . . . .	2-14	2-26	The 5496 Command Immediate Bytes . . . . .	2-38
2-9	Hierarchy of FDFIELD Macro Calls (2 Parts) . . . . .	2-15	2-27	The 3286 Line Printer Command Immediate Bytes . . . . .	2-39
2-10	Hierarchy of FDEND Macro Calls . . . . .	2-17	2-28	The IDR and CCR Command Immediate Bytes . . . . .	2-39
2-11	Hierarchy of FDCTRL Macro Calls . . . . .	2-18	2-29	The FCD Structure . . . . .	2-40
2-12	Overview of the IN Inner Macros (2 Parts) . . . . .	2-20	2-30	The Data Source Group . . . . .	2-41
2-13	Summary of Assembled FDFORM Macro Functions . . . . .	2-22	2-31	The Data-Type Byte . . . . .	2-43
2-14	Summary of Assembled FDPAGE Macro Functions . . . . .	2-23	2-32	The Validity and Function Bytes . . . . .	2-45
2-15	Summary of Assembled FDLINE Macro Functions . . . . .	2-24	2-33	The Validity Group . . . . .	2-45
2-16	Summary of Assembled FDFIELD Macro Functions . . . . .	2-25	2-34	The Function Group (3 Parts) . . . . .	2-46
2-17	Summary of Assembled FDCTRL and FDEND Macro Functions . . . . .	2-26	2-35	The End Control Byte . . . . .	2-50
2-18	The FD Program Structure . . . . .	2-31	2-36	The FD Program Trailer . . . . .	2-51

Part 2 is designed to be used to trouble shoot, debug, and repair problems in the 3735 FD macro programming support. The program listings of the macro expansions are much too lengthy for the user to find an error in a single statement within the internal code. For this reason, the user should perform the following analytical steps to find and correct errors.

1. Verify that the user-specified FD macros are correctly coded.
2. List the records created by the macro assembly using one of the sample programs described in Appendix G or Appendix H in the *IBM 3735 Programmer's Guide*, GC30-3001.
3. Verify that the records listed are correct. If they are not valid, alter the FD utility output with a temporary fix. Refer to Part 2, Section 3 for information about the FD program maintenance procedures to follow. Also, consult Part 2, Section 3 for the correct formats of the FCD bytes and immediate bytes that are needed for correct FD program execution.

The information in this part of the book is subdivided into three main sections as follows.

Section 1 is the introduction to the FD macros and describes in general terms the macros and their relationship to one another. The section discusses the system requirements needed to define and use the macros and the operational considerations of the macros (input and output of the FD macro assembly) including a brief explanation of the MNOTE messages generated by a macro assembly.

Section 2 describes the method of operation that the FD macros use and includes a detailed explanation of the macro organization of inner and outer macros. Also discussed is the step by step procedure that takes place in the assembly of FD macros to generate an FD program.

Section 3, Form Description Program Organization, explains in great detail each portion of an FD program and includes diagrams to aid in visualizing each byte generated by an assembly. The subjects discussed are the FD program header, the FCD bytes, the immediate bytes, and the FD program trailer. Following this information is a description showing how to diagnose and repair errors that appear in FD programs after the assembly process is complete. This discussion is very important to finding and correcting errors in the FD macro programming support.

## Section 1. Introduction

The Form Description (FD) macros are a unique family of IBM System/370 and System/360 OS and DOS Assembly Language macro instructions. They are designed to facilitate the description of terminal-oriented data processing forms in terms of, (1) form structure, (2) data field attributes, and (3) the activities required of a sophisticated terminal and its operator in the processing of such forms. The FD macros are an interrelated group of macro statements that provide a symbolic means of describing to the 3735 (1) the structure of a data processing form, (2) the characteristics of each data field of the form, and (3) the processing to be done on each data field. Assembly of user-specified sets of FD macros results in an object module made up of field control descriptors (FCDs) assembled in an unpacked (expanded) state. The text part of the object module is a collection of bytes that comprise one or more FD programs. The FD program object modules must be restructured into program blocks by the FD utility, which stores them on a direct-access storage device (DASD). Later these program blocks are selectively retrieved and transmitted to specified 3735 terminals. There they are stored on a magnetic-disk storage unit in packed (normal) form, to be recalled as needed by the terminal operator or by the terminal control program. The terminal control program interprets each set of program blocks as a set of directions for the processing of one type of form. Refer to Figure 2-1 for the logic flow of the 3735.

### FORM DESCRIPTION MACRO INSTRUCTIONS

The FD macros are grouped into three classes: structural, procedural, and delimiting. The four structural macros are FDFORM, FDPAGE, FDLINE, and FDFIELD. The procedural macro is FDCTRL and the delimiting macro is FDEND.

Two optional macros available for diagnostic aids are FDTRACE and FDDSPPLY. These two macros can be included with the specifications on the FD macros or added to the internal code.

#### The Structural Macros

The structural macros are hierarchical in character, which restricts the order in which they may be coded. The forms encoder must therefore think of his forms in terms of an information structure in which forms are made of pages, pages are made of lines, lines are made of fields, and fields

are made of characters. The structural macros describe the structural organization of the form and the processing required by each field. They are normally coded so as to maintain forward progression through the entire form; for example, from top to bottom on a page, and from left to right on a line.

The structural macros are designed to take advantage of the promotability of many of their keyword operands. *Promotability* is the ability of a keyword operand to be coded in some particular macro instruction (for example, FDFIELD, FDLINE, FDPAGE) and also to be coded in one or more macros of higher authority. The *authority* of a macro statement extends through an FD program until the same type of macro statement (or one of higher authority) appears later in the FD program. This concept allows a number of the attributes of pages, lines, and fields to be coded at a structural level higher than their minimum level of applicability, and, optionally, to be temporarily overridden at a lower level. Thus, the ability to promote an operand helps to minimize the total amount of encoding required to describe a form.

#### The Procedural Macro

The procedural macro instruction FDCTRL enables the forms encoder to request that the IBM 3735 test the status of one or more terminal control program indicators, also called logic switches. In the FDCTRL macro the encoder may request unconditionally (if no test was coded) or conditionally (based on the result of the test) that any combination of the following actions be taken:

- Modify the status of one or more program logic indicators.
- Perform clearing and/or arithmetic operations on terminal counters.
- Perform clearing on terminal storage.
- Read into, punch from, and/or clear unit-record data buffers.
- Connect and disconnect the communication line for inquiry/response purposes.
- Alter the sequence in which elements of the form are processed.
- Position the 3286-3 Printer to a new line for printing.

FDCTRL macros may appear in any location within the FD macro statements that describe a specific form.

### The Delimiting Macro

The delimiting macro FDEND closes the description of each form and prepares for a form description that may follow by (1) confirming that no source statements have been lost, (2) completing code generation, and (3) reinitializing the global variables used in the internal processing.

### The Trace Macro

There is in addition to the required FD macros an optional diagnostic macro named FDTRACE. This macro provides for the user the ability to trace the activity that occurs during the processing of each FD macro. For further information refer to Part 4, Appendix B of this book.

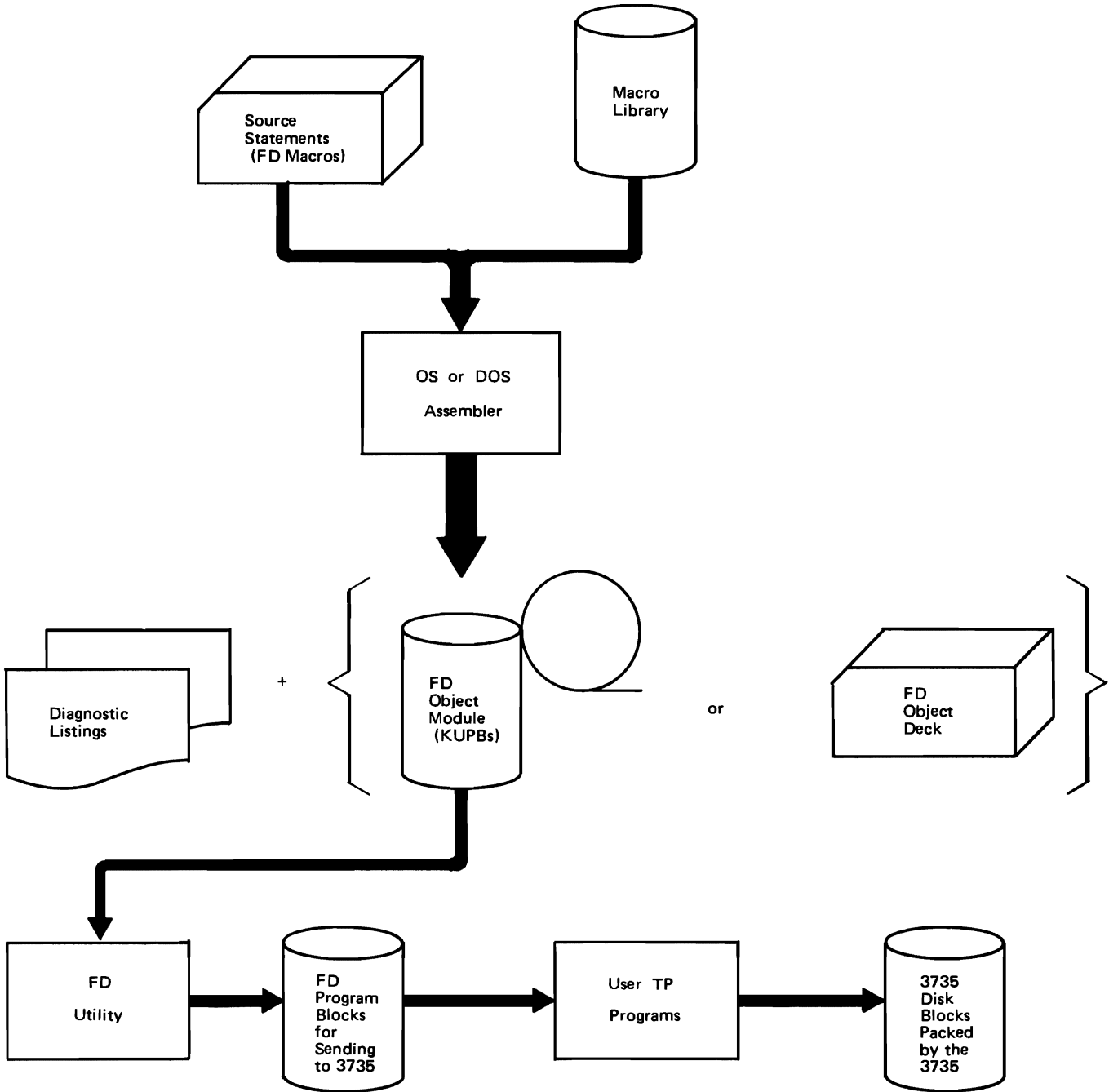


Figure 2-1. 3735 Program Logic Flow

## The Display Macro

The optional display outer macro `FDDSPPLY` activates the `IDFDSP` inner macro, which displays the information requested in the call of `IDFDSP`. For further information refer to Part 4, Appendix B of this book.

## Form Description Macro Organization

An ordered set of FD macros collectively define a form structure with the following characteristics:

- The type (`FDFORM`, `FDPAGE`, and so forth) of FD macro determines the structural level of the macro.
- The structural level of the macro determines, in turn, the scope of the macro and of its keyword operands. The *scope* of a structural macro is that portion of the form description beginning with the macro statement itself and continuing to, but not including, the first equal or higher structural level.
- The form structure allows the FD macros to generate explicit sequences of output-element position-control information needed to locate successive fields in the forms being processed.

Each FD macro consists of three logical divisions, which may be physically distributed: operand validation, parameter establishment, and field control descriptor generation.

The operand validation section of an FD macro tests the value of each operand coded (or the default value) in the present occurrence of the macro. If this section identifies an error, it issues an appropriate `MNOTE` message. The `MNOTE` messages used by the FD macros provide diagnostic information regarding coding errors in the user FD macro statements and provide descriptive information for verifying the correctness of each macro specification.

The parameter establishment section of an FD macro statement establishes the form description program parameters from either coded or default values and saves them in global variables for subsequent reference and assembly by other FD macros.

The field control descriptor generation section of an FD macro composes the field control descriptor (FCD) that is the FD program. It performs (1) the generation of control information for the FD utility, (2) the resolution of symbolic references, and (3) the generation of the FCD data. The 3735 terminal interprets this data as directions for the processing of one data field or the performance of one control operation.

In some cases, the macros reserve space in the assembled data for unresolved references and save the location at which the reference occurred. When the FD macro that

resolves the reference is encountered, the macro inserts the resolved reference in the required position by means of the backward origin facility of the assembler (`ORG` statement) and deletes the saved location. The maximum number of concurrent unresolved references that the FD macros are designed to accommodate within any one form description is 64.

The FD macros manipulate global variables to record the assembly status (whether in form mode or not), structural level, and the occurrence of a `CYCLE` operand. Global variables are values assigned to `SET` symbols in one macro definition used to vary the statements that appear in other macro definitions. These variables enable decisions to be made whether each FD macro statement is coded in a permissible relation to the others, and whether it is suitable for use as a `CYCLE` limit or target, or as a `GOTO` target. Additional global binary switches record the occurrence of a serious error, so that further generation of useless code may be suppressed.

## SYSTEM REQUIREMENTS

In order to use the FD macros, a user must have a host operating system (IBM System/370 or System/360 OS or DOS) with an assembler and a macro library to contain the FD macros. The operating system must contain at least 128K bytes of storage space for OS or 32K bytes for DOS in order to use the FD macros and the FD utility. DOS requires 32K bytes of storage to support teleprocessing.

Any set of FD macro statements can be assembled by any DOS or OS assembler. The assembler must have access to a macro library containing the FD macro definitions. These definitions must be appropriate for the operating system on which the FD program object modules are to be processed and used. If the FD macros are not in the macro library, the assembler generates error messages indicating undefined operation codes. Changing from DOS to OS or *vice versa* requires no recoding, merely reassembly, provided that the code does not exceed the capacity of the assembler.

The assembly of FD macros generates object modules that support the functions of the 3735 control program performed under FCD control. An FCD is that part of a 3735 FD program that the 3735 interprets as the directions for the processing of one data field or the performance of one control operation. An FD program is a set of control data bytes that collectively describe to the 3735 terminal the activity required to process one specific type of form.

## OPERATIONAL CONSIDERATIONS

This section describes, in general terms, the input to, and output from, the FD macro assembly.

### Input and Output of the Form Description Macro Assembly

#### Input

The input for the FD macro assembly consists of one or more ordered sets of FD macro instructions, each set describing one form required for a data processing application, beginning with an FDFORM statement and ending with an FDEND. The macros do not generate executable code. They are first assembled into object modules of FD programs; these programs are input to the FD utility. The utility prepares the resulting FD programs for being sent to the 3735. The 3735 executes the FD programs via the terminal control program interpretive routines.

The FD macro instructions have no direct connection with normal system operation characteristics such as, linkages, return codes, completion codes, abnormal termination, or wait states. They use MNOTE severity codes similar to those of other macro instructions of the host operating system. The MNOTE messages provide diagnostic information regarding coding errors in the user's FD macro statements and provide descriptive information for verifying that each macro specification is correct.

#### Output

*MNOTE Messages:* The MNOTE severity codes generated by the FD macros as output on the assembly listings are indicated below with their meanings and typical uses.

\* Information only; not treated as an error by the assembler. Typical use: to print the attributes of the form, page, line, or field.

0 Mild warning; the condition described is unexpected, but cannot be confirmed as an error. Typical use: to indicate that an operand contains excess suboperands or characters. The FD macros make no assumptions of operand values and ignore the excess suboperands or characters.

8 Severe warning: the condition described constitutes a confirmed error that invalidates the current form (but not the other forms in the same assembly). Operand checking may continue for the invalid form. Typical use: to indicate that the FDEND macro has been processed, leaving one or more forward references unresolved.

The return code from the assembly step equals the highest MNOTE severity code produced in the current assembly. In an IBM System/370 or System/360 OS system, the job control COND operand of the EXEC statement may be used to halt the job on severity code eight. The output of an assembly that incurred a severity code of eight is not usable

and cannot be processed correctly by the FD utility or the 3735. In an IBM System/370 or System/360 DOS system the results of the assembly must be inspected for freedom from error before the results can be further utilized.

*Object Module:* The output from an assembly of a set of FD macro statements is a listing of the input (the FD macros) and the results, together with an object module comprised of ESD, TXT, and END records. The text part of the object module contains FD programs (keyed unpacked FD program blocks and control information) made up solely of nonrelocatable absolute-valued constants. Refer to Figure 2-2 for the OS diagram of FD macro assembly output or to Figure 2-3 for the DOS diagram of FD macro assembly output.

*Control Sections:* The assembly of a set of FD macro source statements generates one or more control sections. Each of the control sections (with the possible exception of the last, which may be shorter by a multiple of 486 bytes) is 2920 bytes long (for OS) or 1464 bytes long (for DOS). The control section is subdivided into 486-byte items called keyed unpacked form description program blocks (KUPBs), plus a four-byte (for OS) or six-byte (for DOS) end-of-assembly indicator. The end-of-assembly indicator tells of the presence or absence of additional control sections. This indicator is set to all zeros for all but the last control section, for which it is set to all ones. Refer to Figure 2-4 for the format of the OS control section or to Figure 2-5 for the format of the DOS control section.

Each KUPB consists of a 10-byte key field and a 476-byte unpacked program block (UPB). Within the key field are two subfields: (1) an eight-byte name subfield and (2) a two-byte count subfield. Within the UPB field are two more subfields: (3) a 470-byte data subfield and (4) a six-byte end-of-form subfield. The subfield contents follow:

1. The name subfield contains the form name, left justified and padded to the right with blanks, as needed. This is the name by which the FD program will be known in the user's FD program library.
2. The count subfield commences with a binary zero and is incremented by one in each KUPB within a single FD program. However, if there was an error in the assembly of the program, this subfield contains a binary -1 (X'FFFF'). If the assembled program is incomplete, this subfield contains a binary -2 (X'FFFE').
3. The data subfield contains the actual instructions that are sent to the 3735 to control the terminal's actions during forms creation and data capture. The first 2 bytes of this subfield contain X'4070' used for head-sector backward chaining on the disk sector.
4. The data in the six-byte end-of-form subfield is also sent to the 3735. This subfield is set to all zeros in every KUPB except the last one in an FD program. In the last KUPB, it is set to all ones to indicate the end of the FD program.

Figure 2-6 illustrates the format of the KUPB and the UPB.

The first control section resulting from an FD macro assembly in OS is named IDF1000. Subsequent control sections are named IDF1001, IDF1002, and so forth, by incrementing the control section counters in the FDFORM macro. The first DOS control section is named IJLF1000, with subsequent control sections named IJLF1001, IJLF1002, and so forth, by incrementing the control section counters in the FDFORM macro.

As the FD macros generate an FCD, they update a global count of bytes. When a multiple of 470 is reached, the macros clear the counter and generate 3X'0000'. Next, the macros generate from globals a form name subfield and a count subfield. At form's end, the macros pad out the current KUPB if incomplete, and generate 3X'FFFF'.

To avoid a nonproductive read operation, the user's application program can examine the end-of-form subfield when retrieving the successive unpacked program blocks for

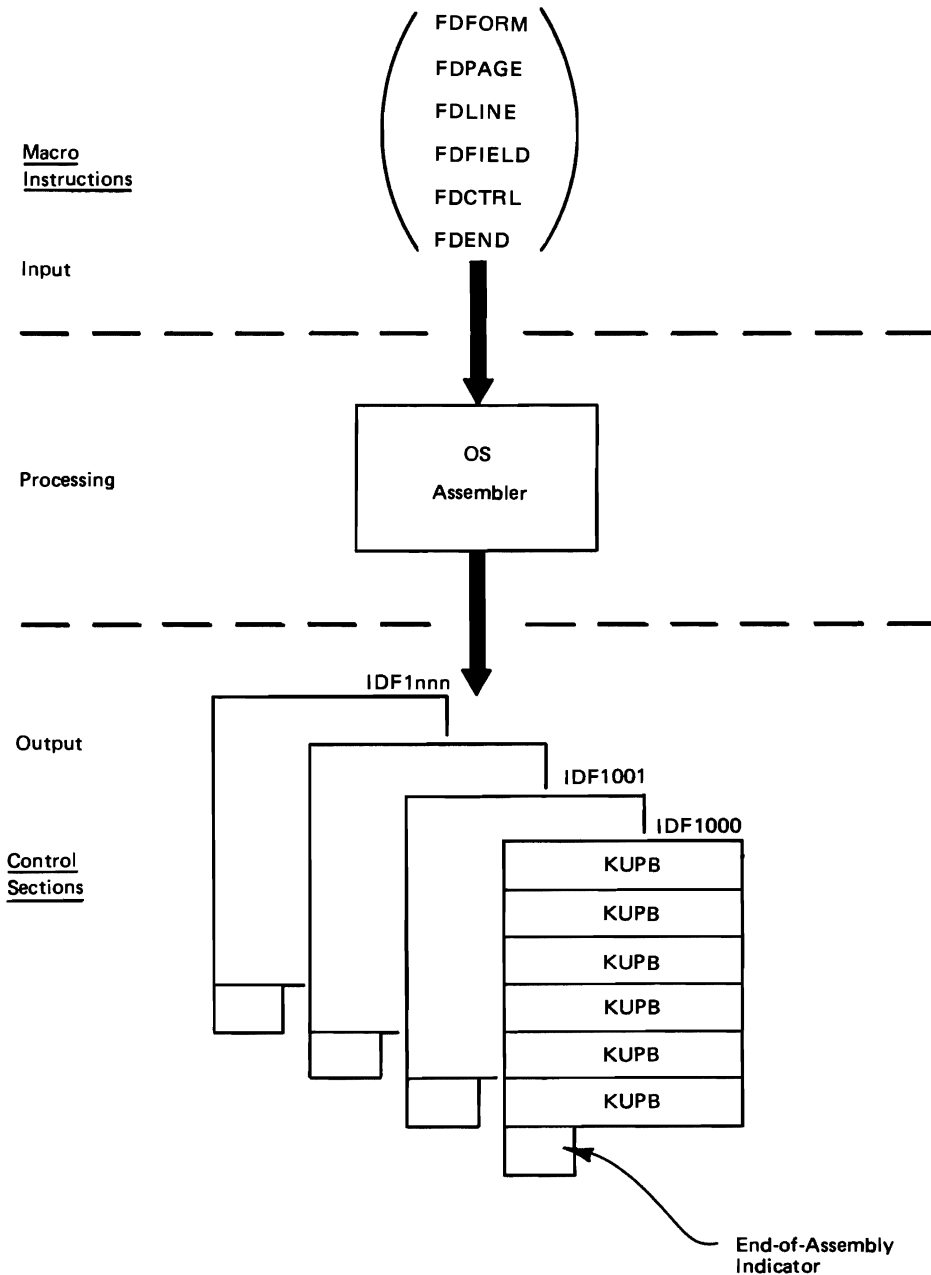


Figure 2-2. OS FD Macro Assembly Output



transmission to a 3735 terminal. In DOS, this examination *must* be done because the FD utility will have stored the UPBs in an indexed sequential data set, and the end of a form will generally *not* coincide with an end-of-file indication.

Although only the first 470 bytes (the data subfield) of each UPB are significant to the 3735, it is more convenient to transmit the entire 476 bytes, since this size of I/O area is required for each data block received from the 3735.

*Segments and Paths*: The listing from the assembly of an FD program divides the program into paths and segments.

Information about the paths and segments helps verify the correctness of the resulting FD program and trace the sequence of actions if the FD program produces unexpected results at the 3735 terminal.

Several actions create a path:

- The start of the FD program
- The encoding of the SAVELOC operand outside of a cycle
- The joining of two paths

The segments of each path are numbered in ascending order beginning with one and increasing to 255. Each segment

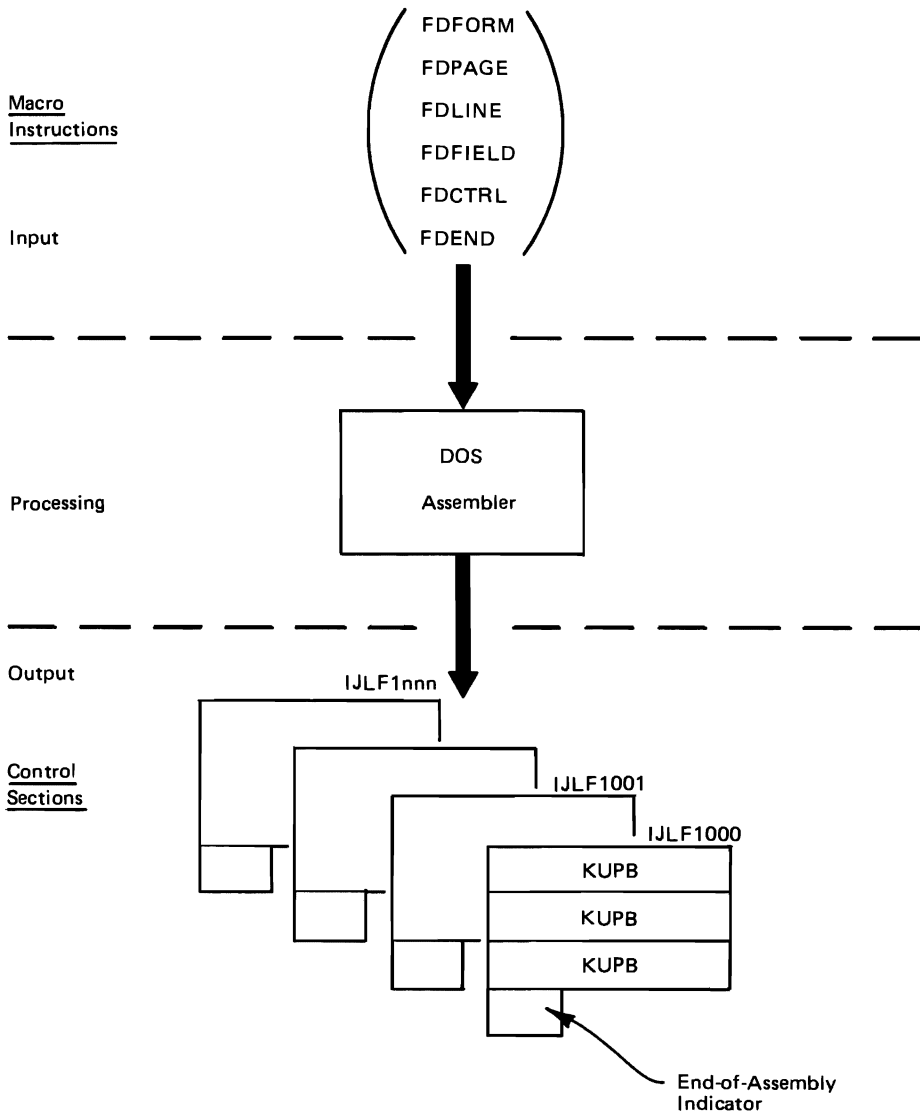


Figure 2-3. DOS FD Macro Assembly Output

transfers control either to another path or to a higher-numbered segment in the same path. If a path is ever entered, the 3735 always executes unconditionally the first segment of that path. The remaining segments of that path are executed conditionally or unconditionally with respect to the first segment. If the first segment of a path is not executed, no part of that path is executed. If the first segment is executed, all provisionally unconditional segments that follow are executed. Then, if their specified conditions are met, the following conditional segments are executed. A path ends when one of the following situations occurs:

- The start of another path
- The end of the FD program
- The occurrence of an unconditional STOP or CANCEL command

Several conditions create the segments comprising a path:

- The start of a path
- The issuance of a branch (for example, GOTO)
- The creation of a cycle
- The encoding of SAVELOC within a cycle

- The macro following the limit of the cycle (the post-limit macro)
- The joining of several segments in the path

Segments and the actions occurring within them are conditional or unconditional as explained in the previous paragraph.

At the end of each path, MNOTE messages describe the indicators, buffers or devices, and the counters used in the path. The MNOTE messages indicate warnings describing possible erroneous use of these storage areas at the end of a path or when the error condition occurs. Two such errors are as follows:

- Requesting output from a buffer that has no previous reference to it in the path
- Loading a buffer and not getting the output from it before the path ends

MNOTE messages indicate the transfer of control between paths and segments when the branches are resolved at the end of each path.

Proceed to the next section of Part 2 for an explanation of the FD macro structure and more detailed information about the FD macro assembly process.

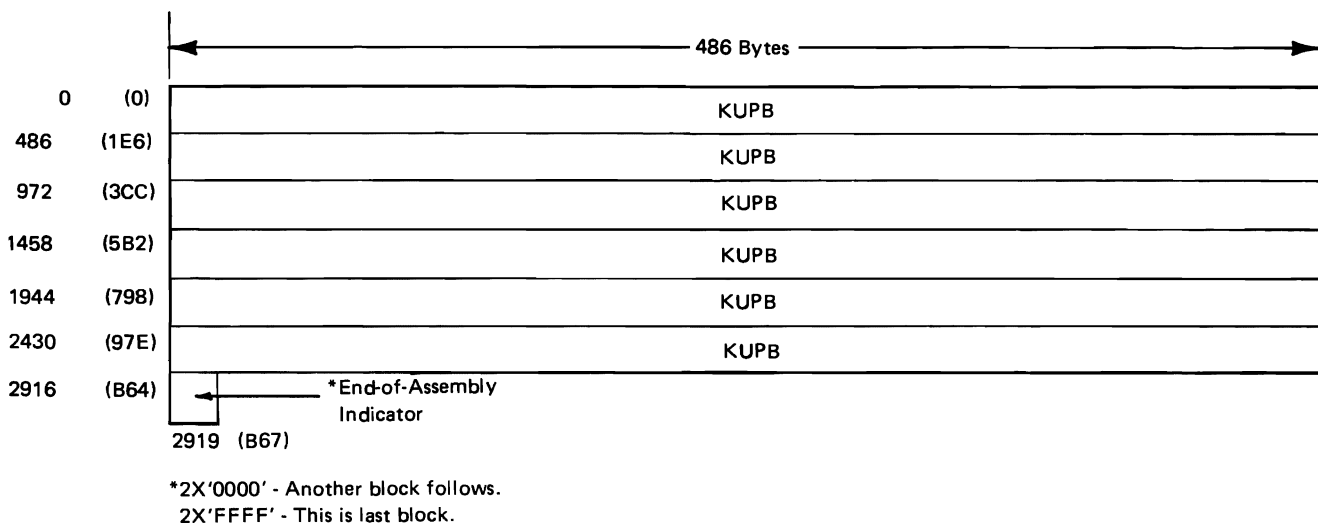
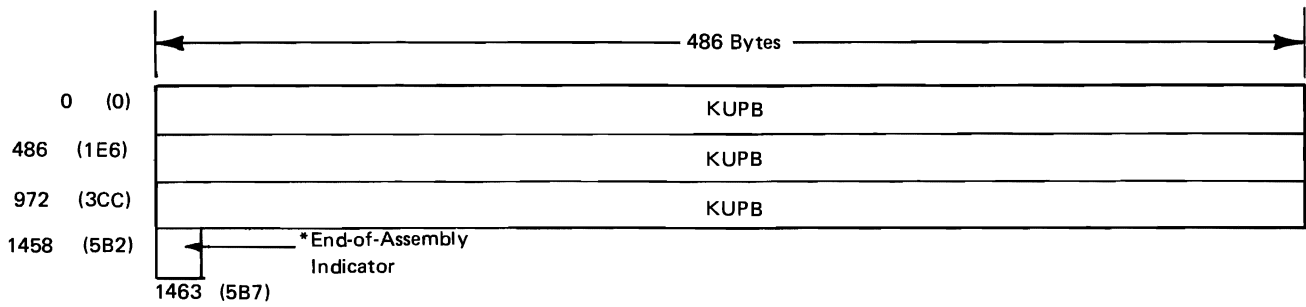
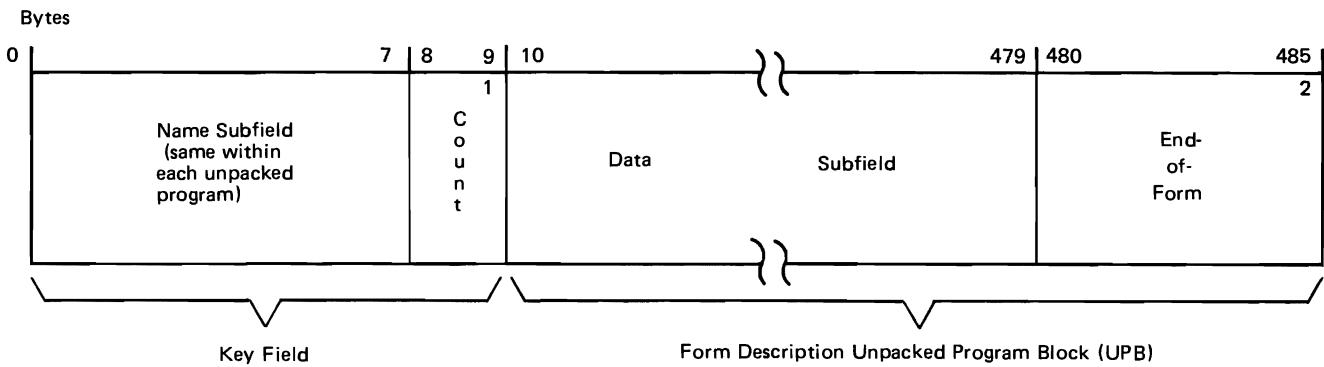


Figure 2-4. OS Control Section Format



\*3X'0000' - Another block follows.  
 3X'FFFF' - This is last block.

Figure 2-5. DOS Control Section Format



- 1 H'00' - First KUPB  
 Incremented by 1 to end of unpacked program; set to -1 and -2 for errors
- 2 3H'00' - All KUPBs except the last  
 3H'-1' (3X'FFFF') - Last KUPB of Form

Figure 2-6. Keyed Form Description Unpacked Program Block (KUPB)

## Section 2. Method of Operation

### FORM DESCRIPTION MACRO STRUCTURE

The FD macro instructions are grouped into three classes; structural, procedural, and delimiting. The four structural macros (FDFORM, FDPAGE, FDLINE, and FDFIELD) describe the structure of a data processing form and the characteristics of each data field of the form. The procedural macro FDCTRL regulates terminal operations that are not uniquely associated with the processing of data within a specific document field. The delimiting macro FDEND closes the description of each form and prepares for a form description that may follow.

In addition to these standard FD macros, the optional diagnostic macros, FDTRACE and FDDSPY, provide valuable debugging tools. For further information refer to Appendix B in Part 4 of this book.

Eleven operands in the four structural macros are promotable:

PAGE	HEIGHT=,VMRG=;
LINE	WIDTH=,HMRG=;
FIELD	SOURCE=,KIND=,SELFCHK=,SINK=;
	JUSTIFY=,FILL=,UL=.

The SOURCE='string' operand specification is the exception in the promotable SOURCE options and is not promotable. These operands, of course, can be specified in the FDFORM macro, thereby minimizing the amount of coding required by the encoder.

Eleven operands are not promotable:

PAGE	page number,SAVELOC=;
LINE	line number,CYCLE=,SAVELOC=;
FIELD	CYCLE=,PICTURE=,IND=,CTR=;
	COUNT=,COMPARE=,SAVELOC=;
	BATCH=,field boundaries.

The SAVELOC= operand can be coded for all of the structural macros except FDFORM, although it is not promotable. The CYCLE= operand, also not promotable, can be coded for the FDLINE, FDFIELD, and FDCTRL macros.

The programming support for the 3735 Form Description macro instructions is composed of an organization of nested macro statements. The six FD macro statements—FDFORM, FDPAGE, FDLINE, FDFIELD, FDCTRL, and FDEND—comprise a group called *outer macros*, macro statements specified by the 3735 forms encoder. The functions of the outer macros are augmented by seventeen *inner macros*, macro statements called by the outer macros or by the inner macros. The inner macros consist of the IDFIN01, IDFIN02, IDFIN03, IDFIN04, IDFIN05, IDFIN06, IDFIN07, IDFIN08, IDFIN09, IDFIN10, IDFIN11, IDFTR, IDFASM, IDFMSG, IDFMSG1, IDFMSG2, and IDFMSG3

inner macros for OS and of the IJLFIN01, IJLFIN02, IJLFIN03, IJLFIN04, IJLFIN05, IJLFIN06, IJLFIN07, IJLFIN08, IJLFIN09, IJLFIN10, IJLFIN11, IJLFTR, IJLFASM, IJLFMSG, IJLFMSG1, IJLFMSG2, and IJLFMSG3 inner macros for DOS. These macros are hereafter referred to as the IN01, IN02, IN03, IN04, IN05, IN06, IN07, IN08, IN09, IN10, IN11, TR, ASM, MSG, MSG1, MSG2, or MSG3 macros. The global variables required by both the outer macros and the inner macros for their processing are contained in the IDFGBL copy book for OS or in the IJLFGBL copy book for DOS (hereafter called the GBL copy book). The first statement of every macro is COPY IDFGBL for OS or COPY IJLFGBL for DOS, which action ensures that each macro has access to all global variables.

The nesting concept is divided into levels as follows:

1. The outer FD macro instructions call the IN01, IN02, IN03, IN04, IN05, IN06, IN07, IN08, IN09, IN10, and IN11 (the IN macros) inner macros.
2. The IN macros call the MSG, the MSG1, the MSG2, the MSG3, the TR, and the ASM inner macros as they are needed.
3. The MSG, MSG1, MSG2, and MSG3 macros issue MNOTE diagnostic messages and exit.
4. The outer FD macros may also call the ASM, the MSG, MSG1, MSG2, or MSG3 inner macros. This hierarchy is further illustrated in Figures 2-7 through 2-11, inclusive.

The outer FD macros contain promotable operands plus the operands that are unique to each one of the macros. For example, the FDFORM macro has the FID=, MRGSTOP=, MESSAGE=, HTAB=, PACKING=, DEVICES=, and BUFFERS= operands. The operands in the IN inner macros are positional rather than keyword as in the outer macros. The outer macro statements perform their own sequence checking to determine whether the outer macros are in the correct sequence. Each keyword operand in the outer macros is mapped onto a positional operand in one of the IN inner macros. This mapping begins with the operands that appear most frequently in the outer macros and progresses to the operand that is specified the least number of times.

The outer FD macros call one or more of the IN inner macros to set up the mapping of all the operands that the forms encoder specified in the outer macros. The IN inner macros perform the mapping process until all the outer FD macro operands are placed in global variables (arrays). If there is an error, the IN macros call the MSG, MSG1, MSG2, or MSG3 inner macro to generate MNOTE messages.

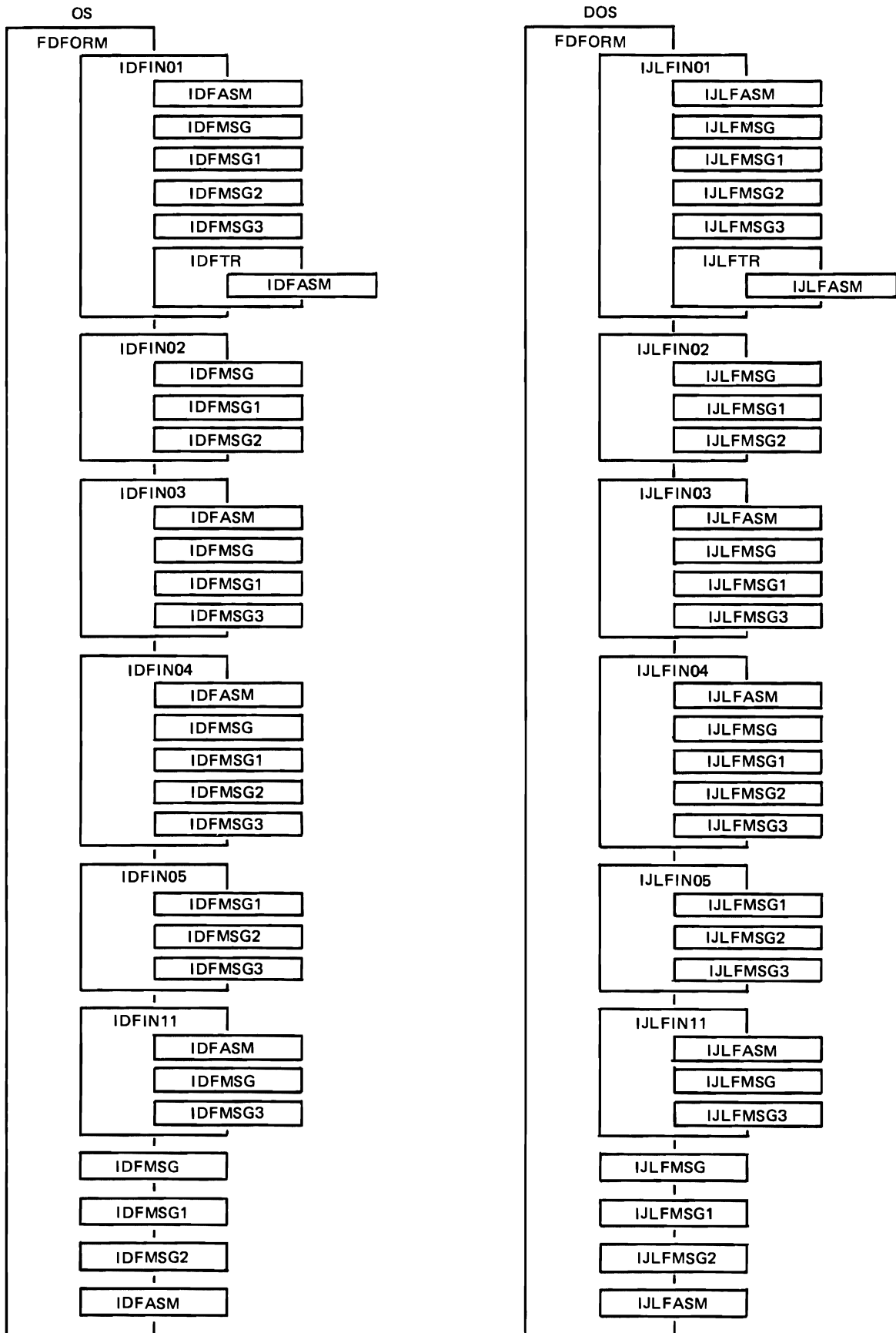


Figure 2-7. Hierarchy of FDFORM Macro Calls

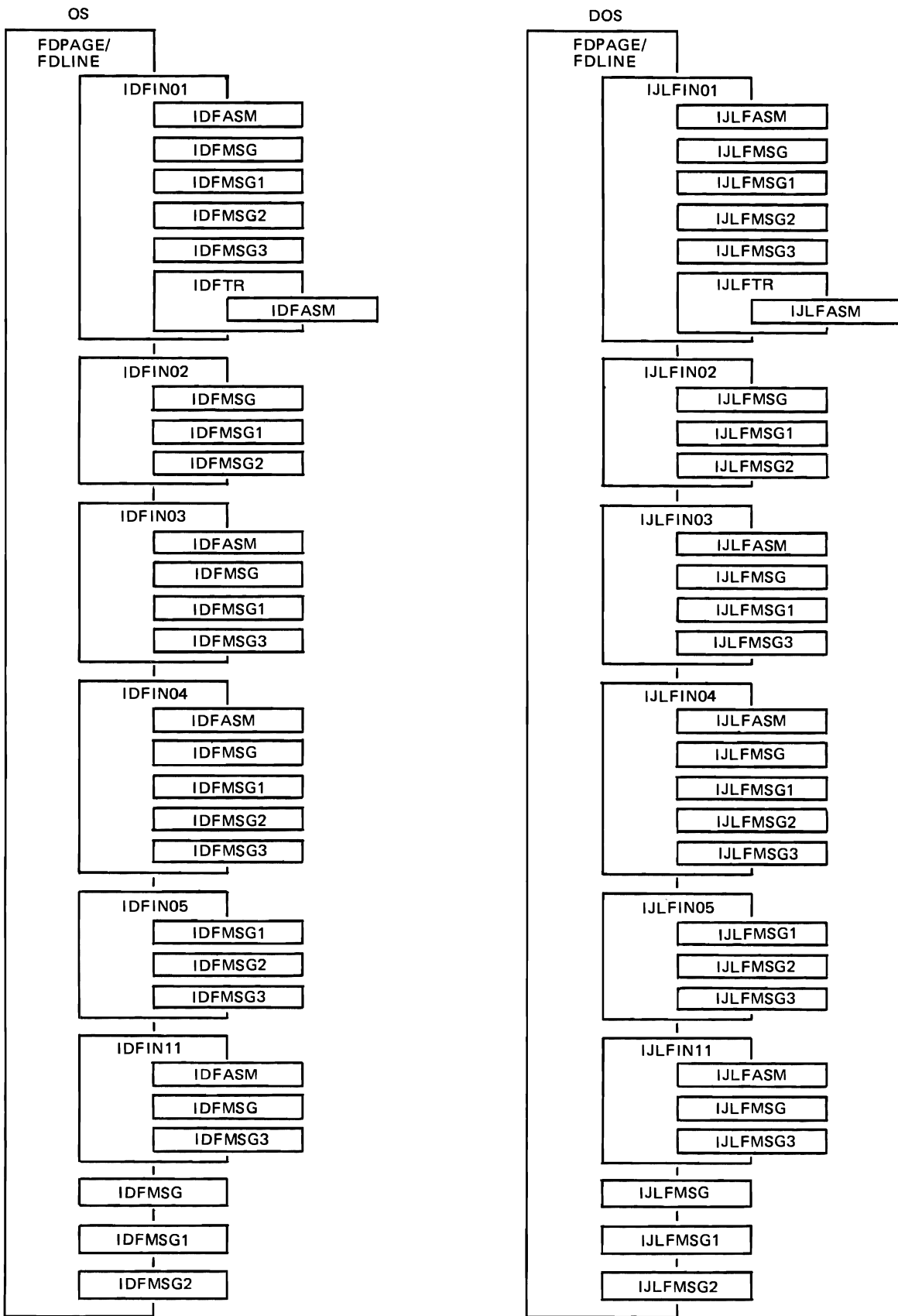


Figure 2-8. Hierarchy of FDPAGE and FDLINE Macro Calls

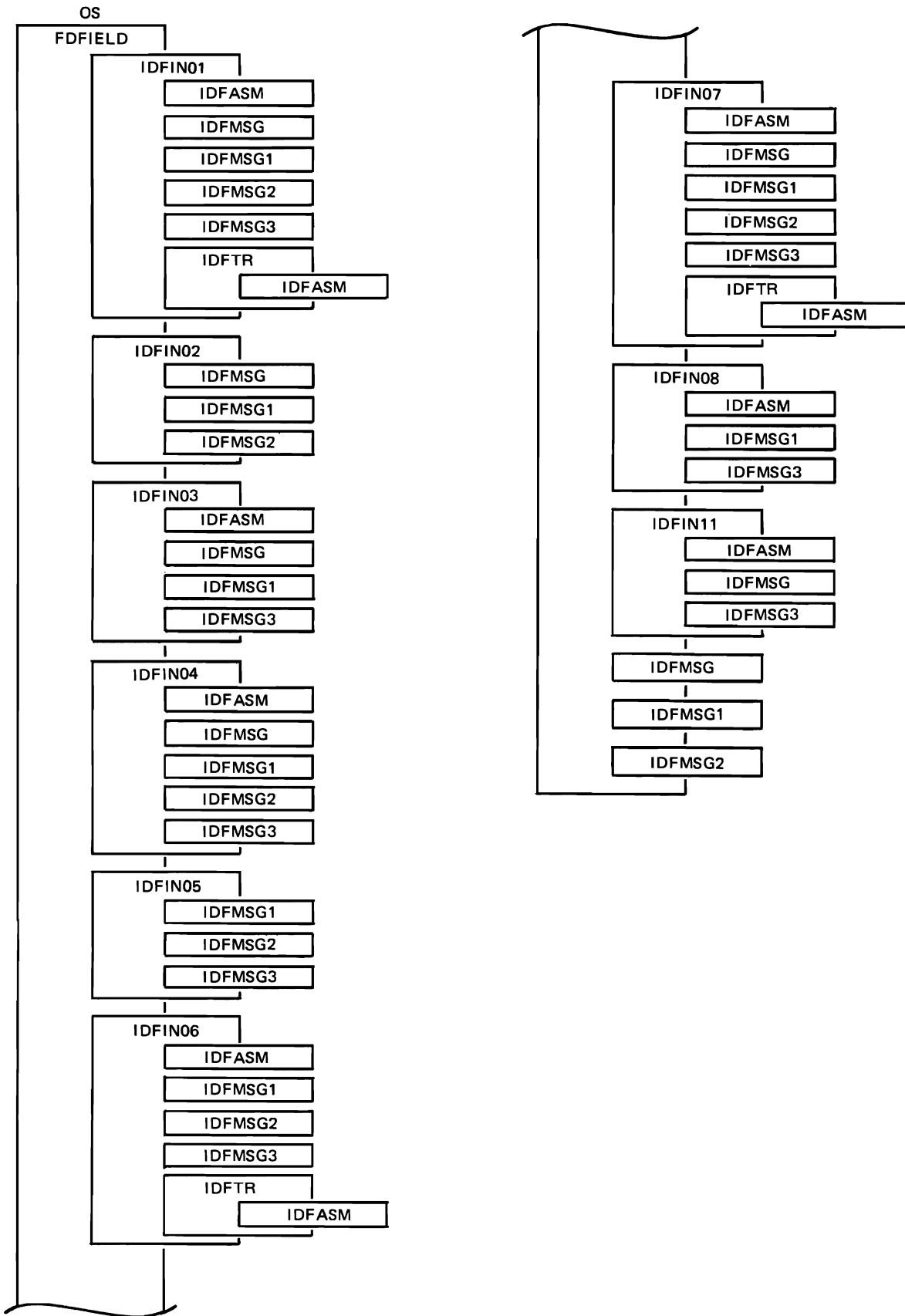


Figure 2-9. Hierarchy of FDFIELD Macro Calls (Part 1 of 2)

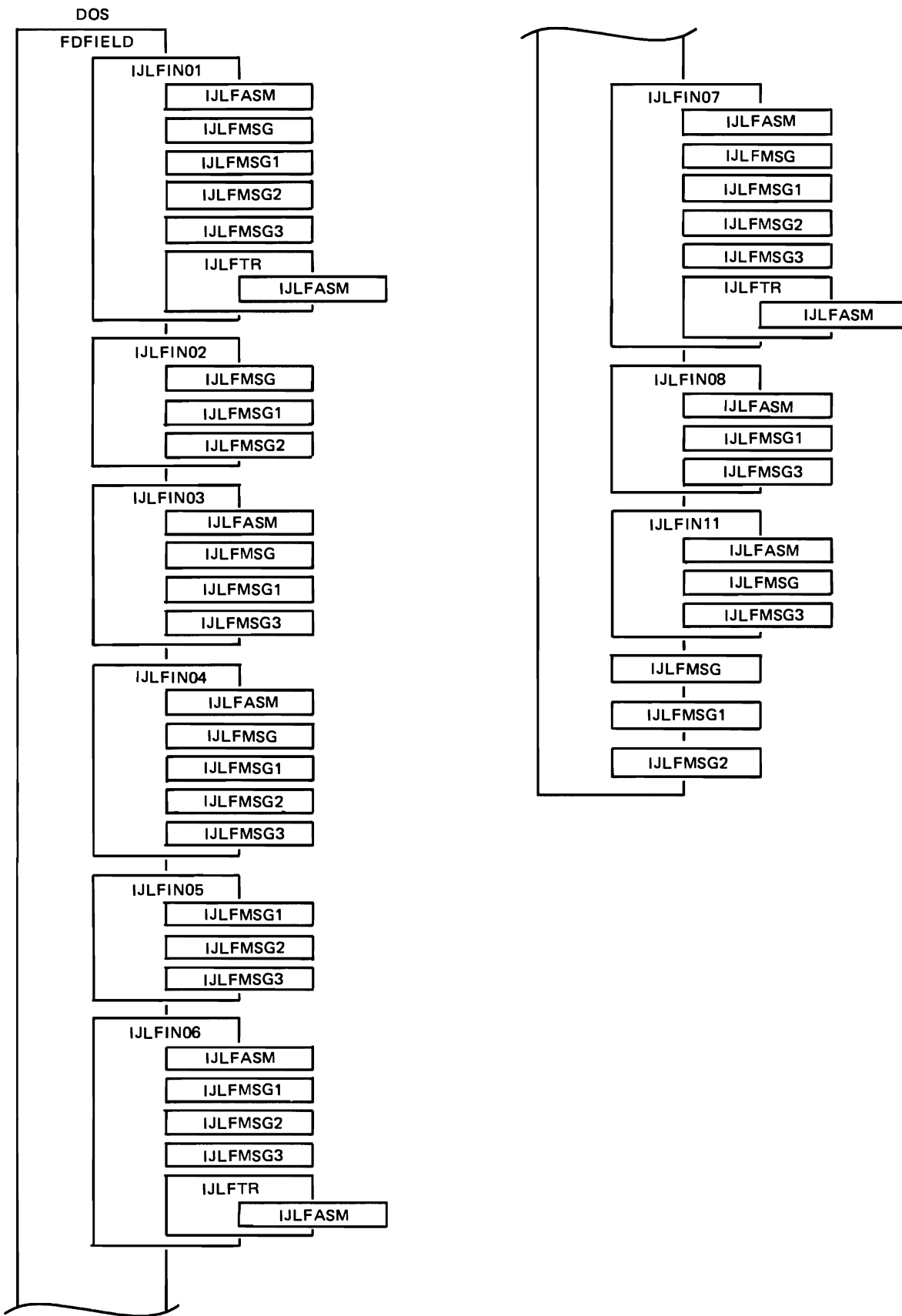


Figure 2-9. Hierarchy of FDFIELD Macro Calls (Part 2 of 2)



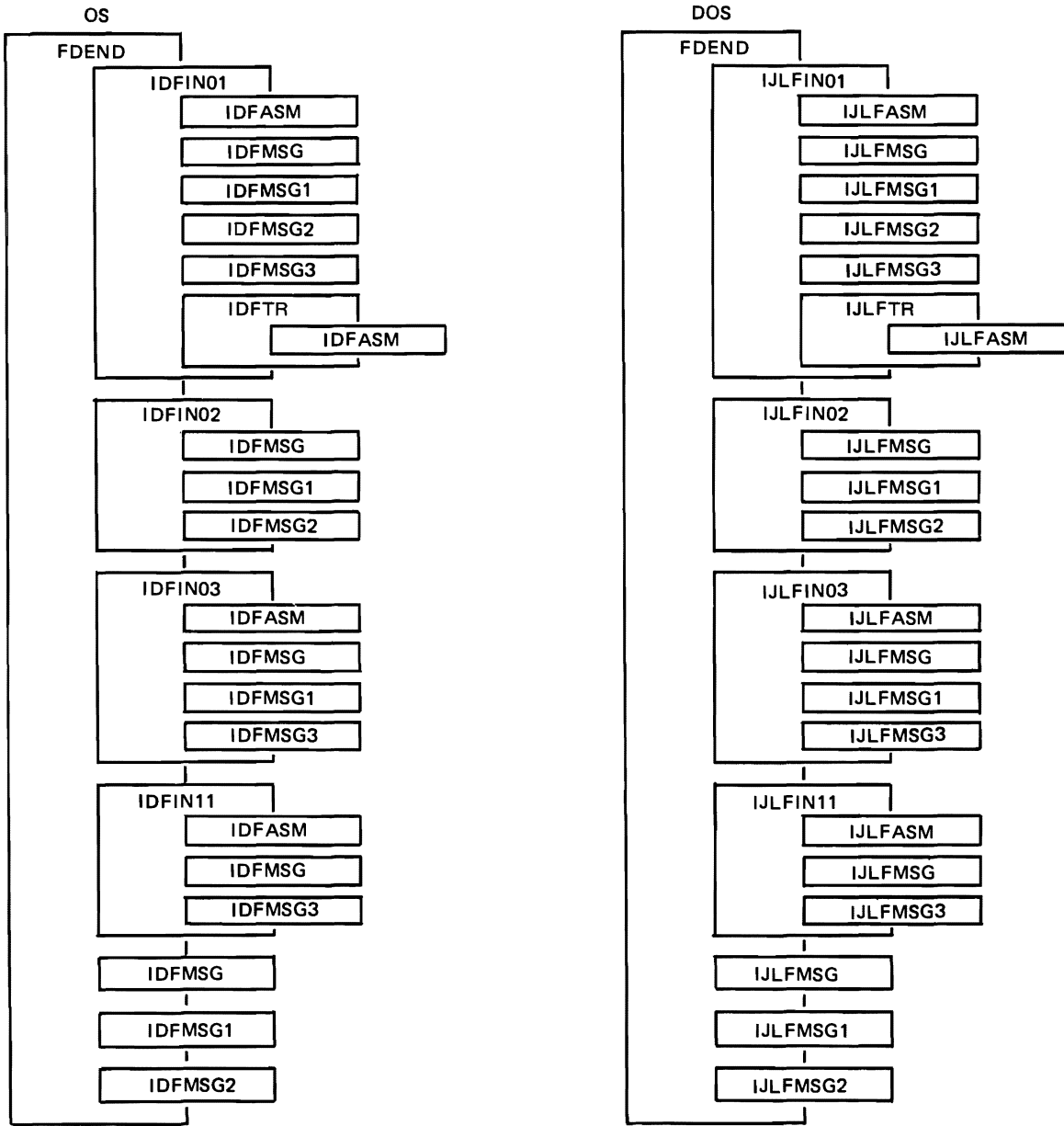


Figure 2-10. Hierarchy of FDEND Macro Calls

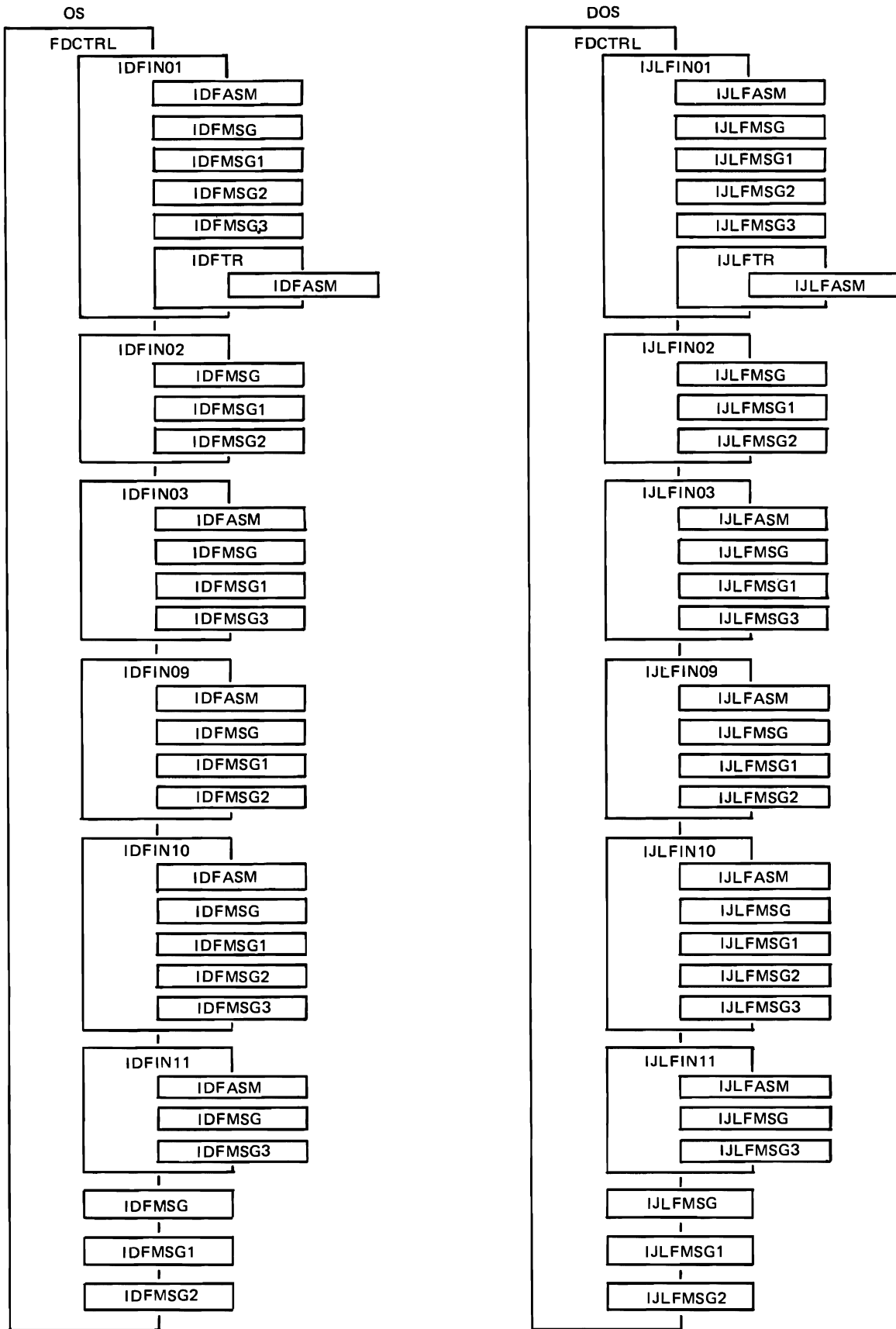
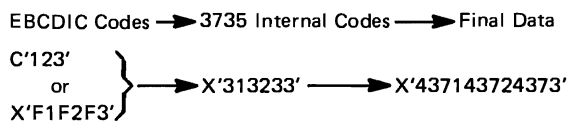


Figure 2-11. Hierarchy of FDCTRL Macro Calls

The TR inner macro is invoked to translate to internal 3735 code the character strings that the forms encoder specified in his outer FD macro statements. If there are errors during the assembly processing, the MSG, MSG1, MSG2, or MSG3 inner macro issues MNOTE messages.

Figures 2-7 through 2-11 illustrate the hierarchy of the outer and inner macros according to their macro call structure. Figure 2-12 shows the logical flow of the inner macro processing according to the order in which the processing occurs.

The line transmission code of the 3735 terminal is either an EBCDIC subset or an ASCII subset. The FD macro assembly arranges character data that is part of a form description into unpacked and padded internal 3735 codes. The resulting even-numbered bytes have a X'40' high-order pad, and the odd-numbered bytes have a X'70' high-order pad. An example of this code transformation is as follows.



## FD MACRO FUNCTIONS

### FDFORM Macro Instruction

The FDFORM macro checks that the previous macro did not start chaining and that the previous macro, if any, was an FDEND statement. Refer to Figure 2-13 for a summary of the FDFORM macro functions.

The FDFORM macro must have a symbolic name, which is comprised of from 1 to 8 valid alphanumeric characters, with the first alphabetic. This name is used as the catalog name by which the operating system and the system programmer refer to the form being described. The name should begin with an alphabetic character and may not contain any blanks or special characters. The assembler considers \$, #, and @ to be alphabetic characters.

Each FDFORM macro initializes global arrays (variables). These arrays provide for the temporary storage of data that pertains to unresolved forward references, for the recording of structural level and assembly mode (form or nonform), and for the downward transmission of the values of promotable operands specified in the FDFORM and the other structural macros.

The FDFORM macro arranges the three-digit FID operand as a group of six bytes with high-order padding added. This format makes possible nontransparent transmission to the 3735 terminal without regard to the original content of the data. The macro generates the bytes indicating the PACKING specification for the form and processes the MRGSTOP, DEVICES, and BUFFERS operands. The assembly also arranges the characters in the MESSAGE operand into a group if the user so specifies in the FDFORM macro.

If the user specifies horizontal tabular stops, the assembly generates a X'7F' delimiter and the tabular stop intervals. Finally, the assembly of the FD macros generates a X'00' delimiter as an end-of-heading indicator. For additional information about the format of the assembled code, refer to the discussion of the Form Description Programs that follows in Part 2, Section 3.

The inner macros called by the FDFORM macro process the HEIGHT, VMRG, WIDTH, HMRG, SOURCE, SELFCHK, KIND, SINK, FILL, JUSTIFY, and UL operands.

### FDPAGE Macro Instruction

The structural FDPAGE macro marks the beginning of each page description within the current form description. Refer to Figure 2-14 for a summary of the FDPAGE macro functions.

The FDPAGE macro checks that the previous macro did not start chaining and whether FDPAGE is coded within a form. The inner macros called by the FDPAGE macro process the SAVELOC, HEIGHT, VMRG, page number, WIDTH, HMRG, SOURCE, SELFCHK, KIND, SINK, FILL, JUSTIFY, and UL operands.

### FDLINE Macro Instruction

The structural FDLINE macro indicates the beginning of each line description within the current page description. Refer to Figure 2-15 for a summary of the FDLINE macro functions.

The FDLINE macro checks that the previous macro did not start chaining and whether the FDLINE macro is coded within a page. FDLINE calls the inner macros to process the SAVELOC, WIDTH, HMRG, and line number operands.

If the CYCLE operand is specified, the FDLINE macro assembles the bytes to transfer control to the start of the cycle, a begin-cycle delimiter, and a cycle count. The macro reserves space for the index count (index to the target line) and for the address of the target field control descriptor that gives directions for the processing of a data field or the performance of a control operation. The CYCLE operand creates two unresolved forward references.

The FDLINE macro also processes the SOURCE, SELFCHK, KIND, SINK, FILL, JUSTIFY, and UL operands by calling inner macros.

### FDFIELD Macro Instruction

A structural FDFIELD macro is required for each field within the current line description. Refer to Figure 2-16 for a summary of the FDFIELD macro functions.

The FDFIELD macro checks that the previous macro did not start chaining (unless it was an FDFIELD macro) and whether the FDFIELD is coded within a line. The macro calls inner macros to process the SAVELOC, CYCLE, and field boundary operands.

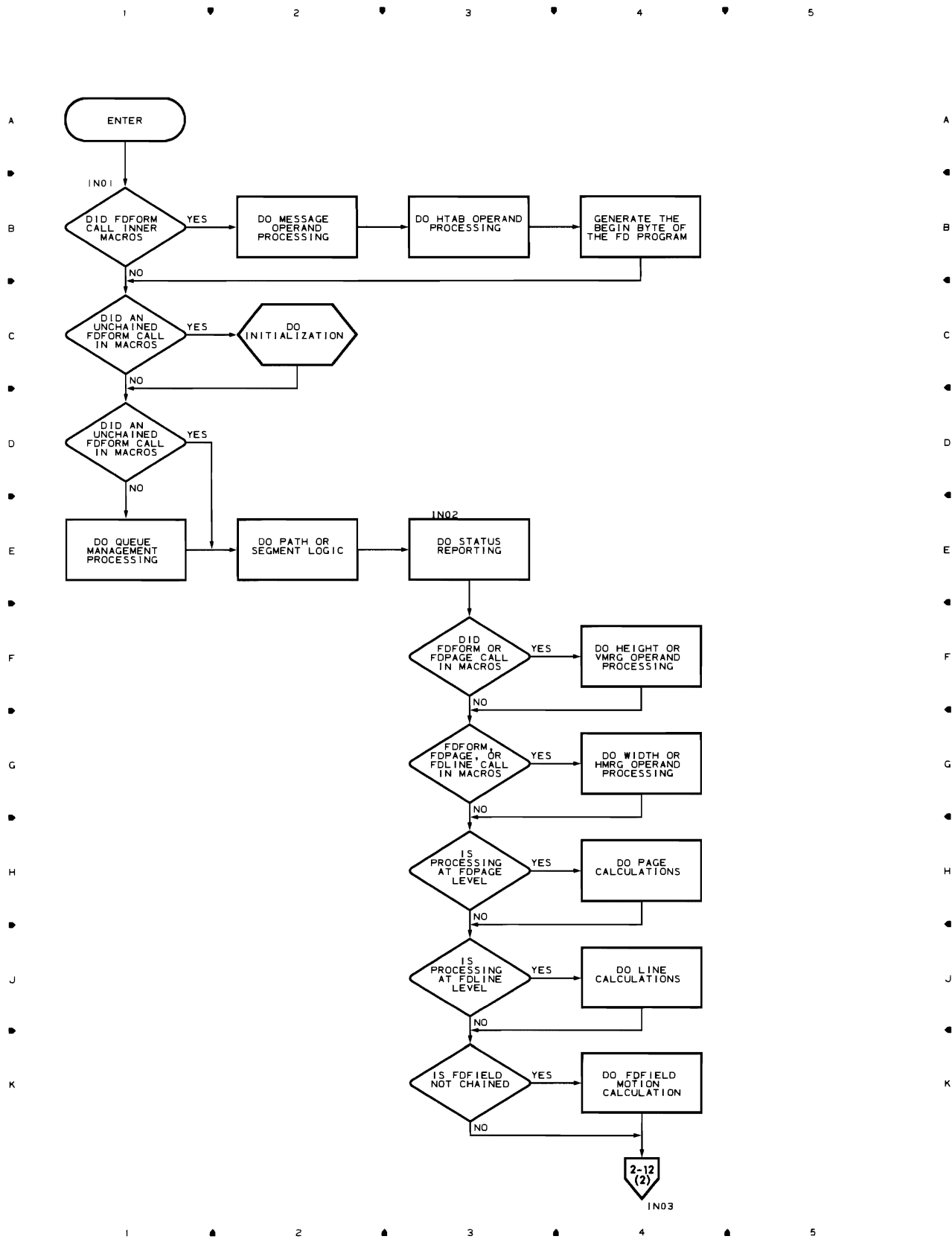


Figure 2-12. Overview of the IN Inner Macros (Part 1 of 2)

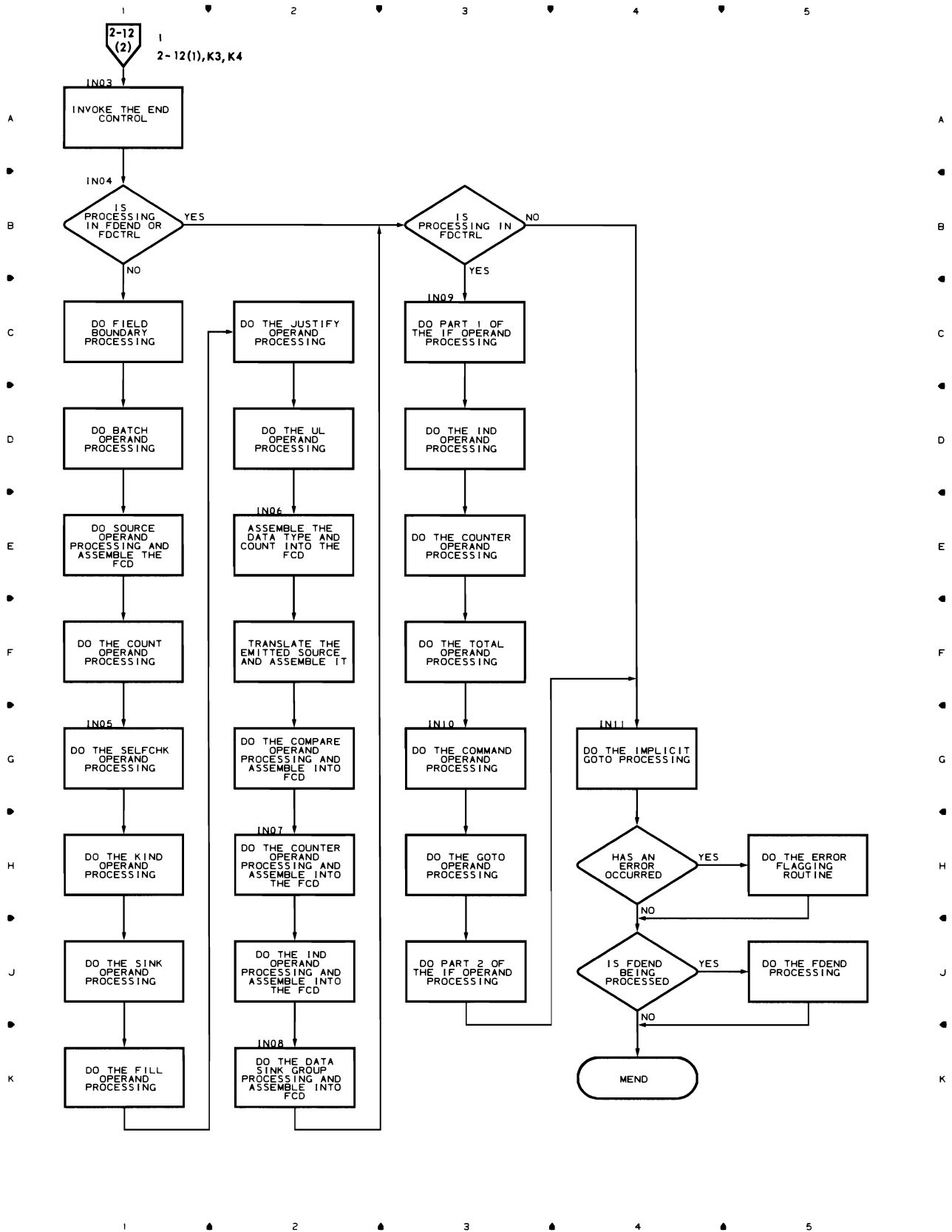


Figure 2-12. Overview of the IN Inner Macros (Part 2 of 2)

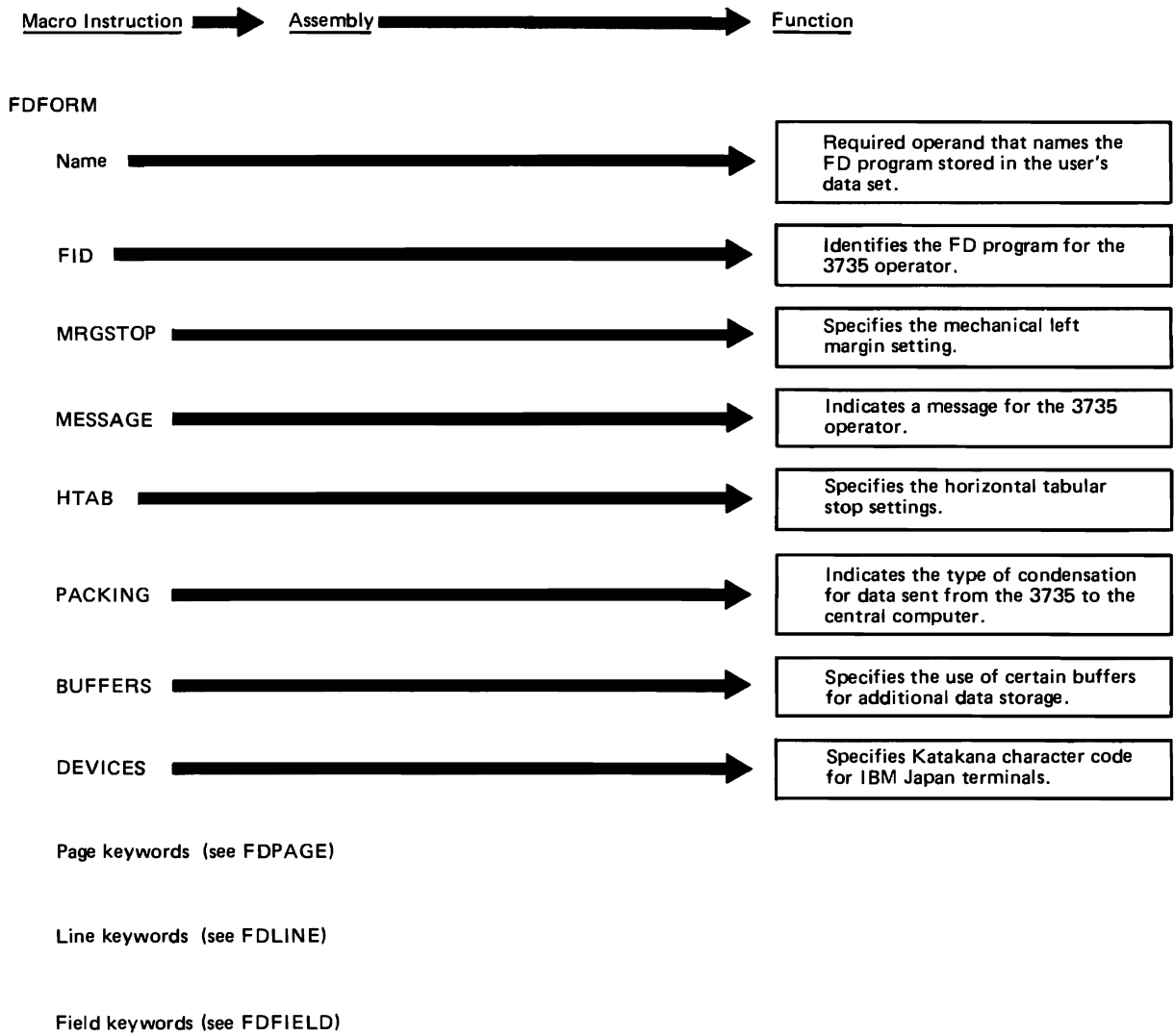


Figure 2-13. Summary of Assembled FDFORM Macro Functions

The assembly of control sequences produces mechanical motion from the previous output-element position to the position of the first output character. If the current macro resolves one or more forward references, the macro builds an inward-branch table to funnel the branches into the start of the field control descriptor (FCD).

Each branch table entry contains the disk address of the target field, in sectors and bytes, relative to the end of the entry. The resolution of each forward reference, at assembly time, alters the location counter (and the control section, if necessary) to point to the null code at which the reference was made. This null code is then modified to an address pointing to the associated table entry, which in turn points to the current FCD.

The FDFIELD macro calls inner macros to process and generate FCDs from the BATCH, SOURCE, COUNT, SELFCHK, KIND, SINK, FILL, JUSTIFY, UL, COMPARE,

CTR, and IND operands. The inner macros set tables (global variables) to allow assembly of control sequences to the next FCD.

**FDCTRL Macro Instruction**

The procedural macro FDCTRL regulates terminal operations that are not uniquely associated with the processing of data within a specific document field. Refer to Figure 2-17 for a summary of the FDCTRL macro functions.

The macro checks whether any chaining originated from a previous FDCTRL macro and if FDCTRL is coded within a form.

If the SAVELOC operand is specified, FDCTRL saves the location of the current macro within the form description for backward reference to it (reference is to the name coded on the FDCTRL macro).

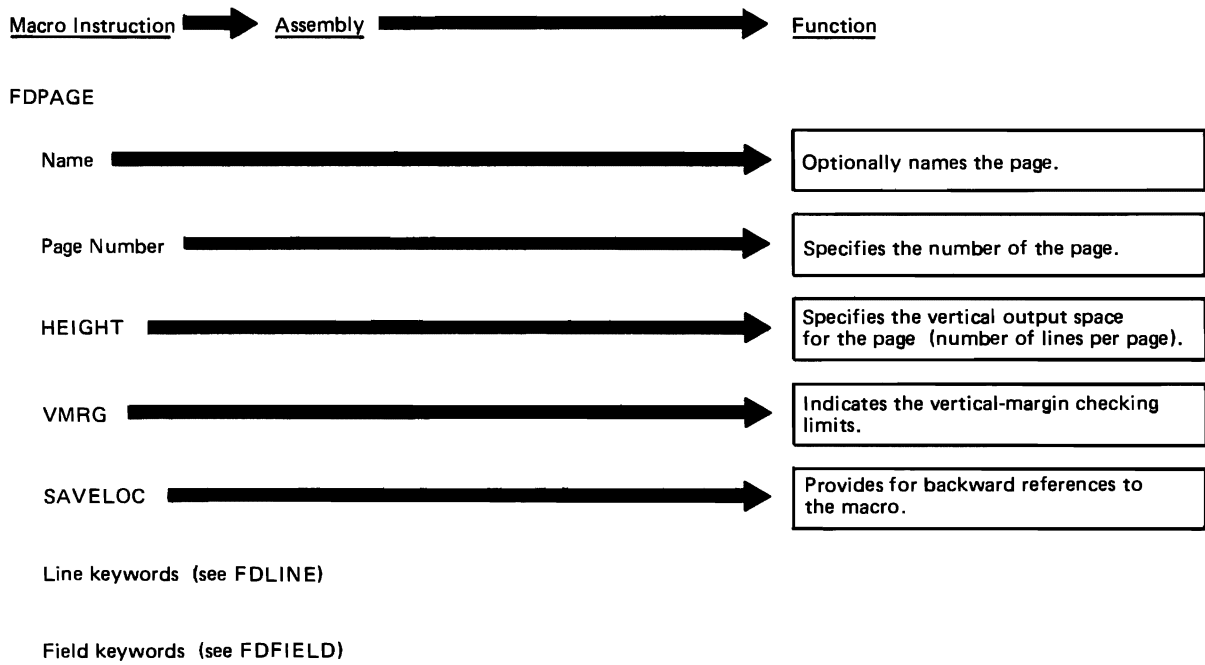


Figure 2-14. Summary of Assembled FDPAGE Macro Functions

If the CYCLE operand is specified, FDCTRL assembles the byte to transfer control to the start of the cycle, a begin-cycle delimiter, and a cycle count.

The IF operand causes assembly of a conditional execution sequence consisting of one or more indicator tests and an unresolved forward reference. The unresolved forward reference is resolved after FDCTRL assembles the last byte resulting from the other operands coded in that FDCTRL statement. The reference serves to bypass the actions of the other operands if the indicator test fails.

The IND operand assembles code required to set, reset, or invert one or more program logic indicators.

The CTR operand assembles one FCD for each arithmetic operation to be performed on one or more counters. Each of these FCDs has the following properties: CTR operand of FDFIELD, digit character-string source, and null sink.

If the TOTAL operand is specified, the FDCTRL macro sets bits in an FD program to cause the 3735 to add to the specified counters the values of numeric data fields. Specified FD programs process these fields (form IDs are coded in the TOTAL operand) and flag them as belonging to particular data batches within the FD programs. If both TOTAL and COMMAND operands are specified on the same FDCTRL macro, the macro does the processing for the TOTAL operand before that for the COMMAND operand.

The COMMAND operand assembles (in the sequence in which the suboperands are coded) the terminal control commands that affect the reader and the reader buffer, the punch, the punch buffer, the line adapter, the line printer

buffer, the line printer, the inquiry (INQ) buffer, the storage (STG) buffer, and the operator identification card reader. In addition, this operand assembles the command to stop processing the current form and to advance to the next copy of the form, if any.

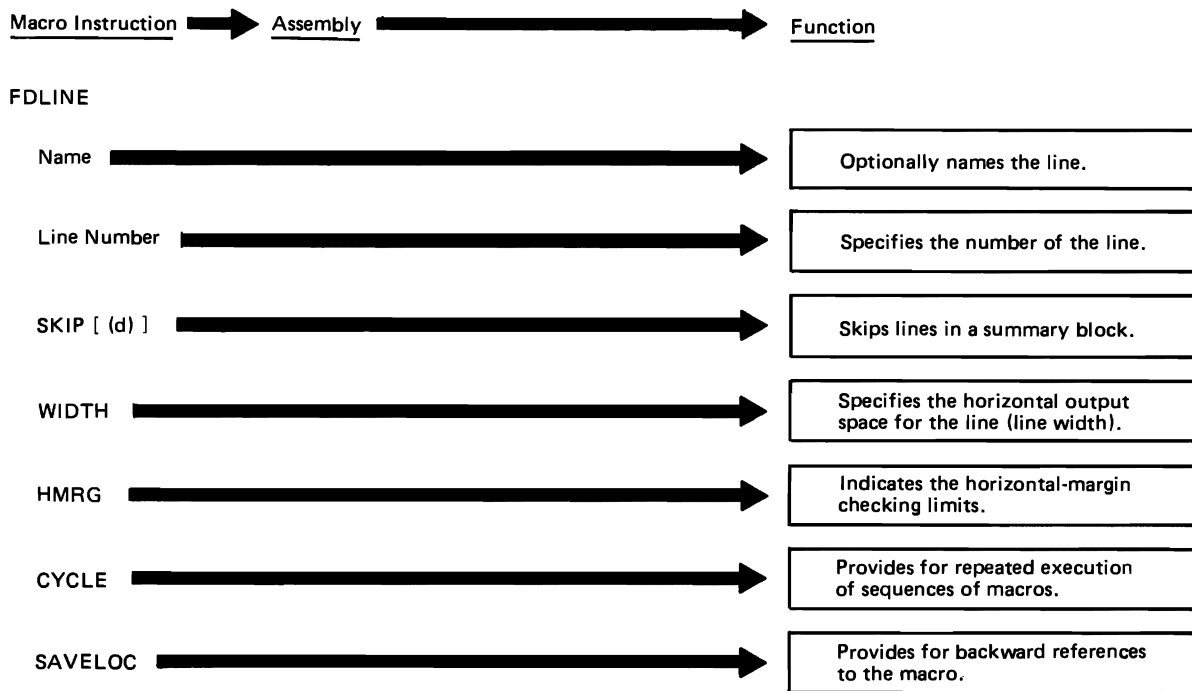
If the GOTO operand is specified in the FDCTRL macro, the macro enters the specified destination in the list of unresolved forward references. The GOTO operand assembles null code that is to be filled in later when the destination of the GOTO specification is resolved when SAVELOC is specified for the destination, in which case the motion is immediately generated.

Finally, the FDCTRL macro sets up tables (global variables) to allow the assembly of control sequences to the next FCD or FDCTRL macro.

### FDEND Macro Instruction

The delimiting macro FDEND indicates the end of each form description. The macro must be specified once as the last macro of each group of FD macros that collectively describe a single form. The FDEND macro confirms that no source statements have been lost, completes code generation, and reinitializes the global variables of the FD macros in anticipation of a possible form description to follow. Refer to Figure 2-17 for the FDEND macro function.

If any forward references are outstanding, FDEND either resolves them or issues an MNOTE message with a severity code of eight if they cannot be resolved.



Field keywords (see FDFIELD)

Figure 2-15. Summary of Assembled FDLINE Macro Functions

At the end of each path, the termination process for the FDEND macro prints out a list of messages describing the resources used, the resources that are incorrectly used, and the warning conditions that have occurred in the assembly. Each of these MNOTE messages has a severity code of zero.

The FDEND statement assembles sufficient control sequences to advance the forms to the page 1, line 1, column 1 position of the next form. If continuous forms are not used, FDEND ejects the current form from the printer.

The FDEND macro supplies the end-processing indicator X'7E' by generating the unpacked sequence X'477E'. If this sequence does not fill out the current KUPB, the macro generates an additional pad of X'476E' to complete the block. When the form program records are compressed on the disk, the 3735 terminal discards the pad characters.

### THE ASSEMBLY OF FORM DESCRIPTION PROGRAMS

The assembly of Form Description programs is logically divided into four main portions: assembly of the FD program header, address resolution and motion control assembly, assembly of immediate bytes, and assembly of the FCD. FD program header assembly takes place during the processing of the FDFORM macro statement (or statements if the MESSAGE and/or HTAB operands are chained). Address resolution and motion control assembly occurs just before the immediate bytes are assembled. The immediate

byte assembly is done before assembling the FCDs. The assembly of an FCD begins after the testing of the promotable operands in the FDFIELD macro statements and continues with the testing of the operands that are not promotable. The exception, when the BATCH operand is specified, is that BATCH is always assembled first generating an immediate command. An FCD is also assembled during the processing of an FDCTRL macro statement when the operands require the internal generation of a data field. Other assemblies performed when needed are those for cycle controls and for FD program branch codes.

The FD program assembly proceeds by byte pairs, with each IDFASM (OS) or IJLFASM (DOS) inner macro creating a two-byte DC statement of the form X'4x7y', or the binary or halfword equivalent. The 3735 terminal repacks this DC statement to the form X'Pxy', where P is the internally generated parity, x is three bits, and y is four bits. The IDFASM (OS) or IJLFASM (DOS) inner macro instruction (called the ASM inner macro) performs the FD program assembly process except for character strings that are normally translated and assembled in the TR inner macro.

### Form Description Program Header Assembly

The assembly of each FD program begins with the three decimal characters of the FID operand on the FDFORM macro statement. The assembler translates each character



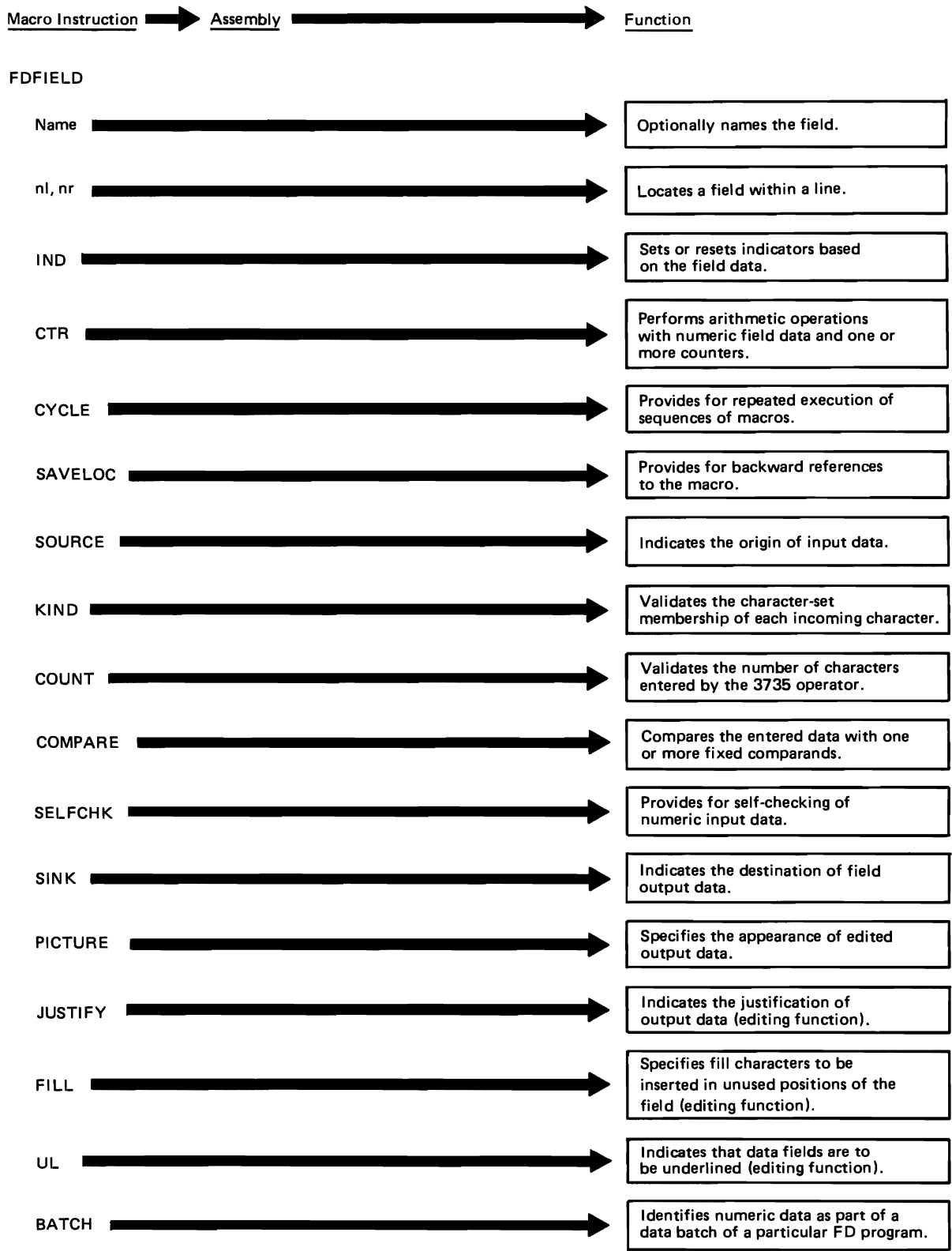


Figure 2-16. Summary of Assembled FDFIELD Macro Functions

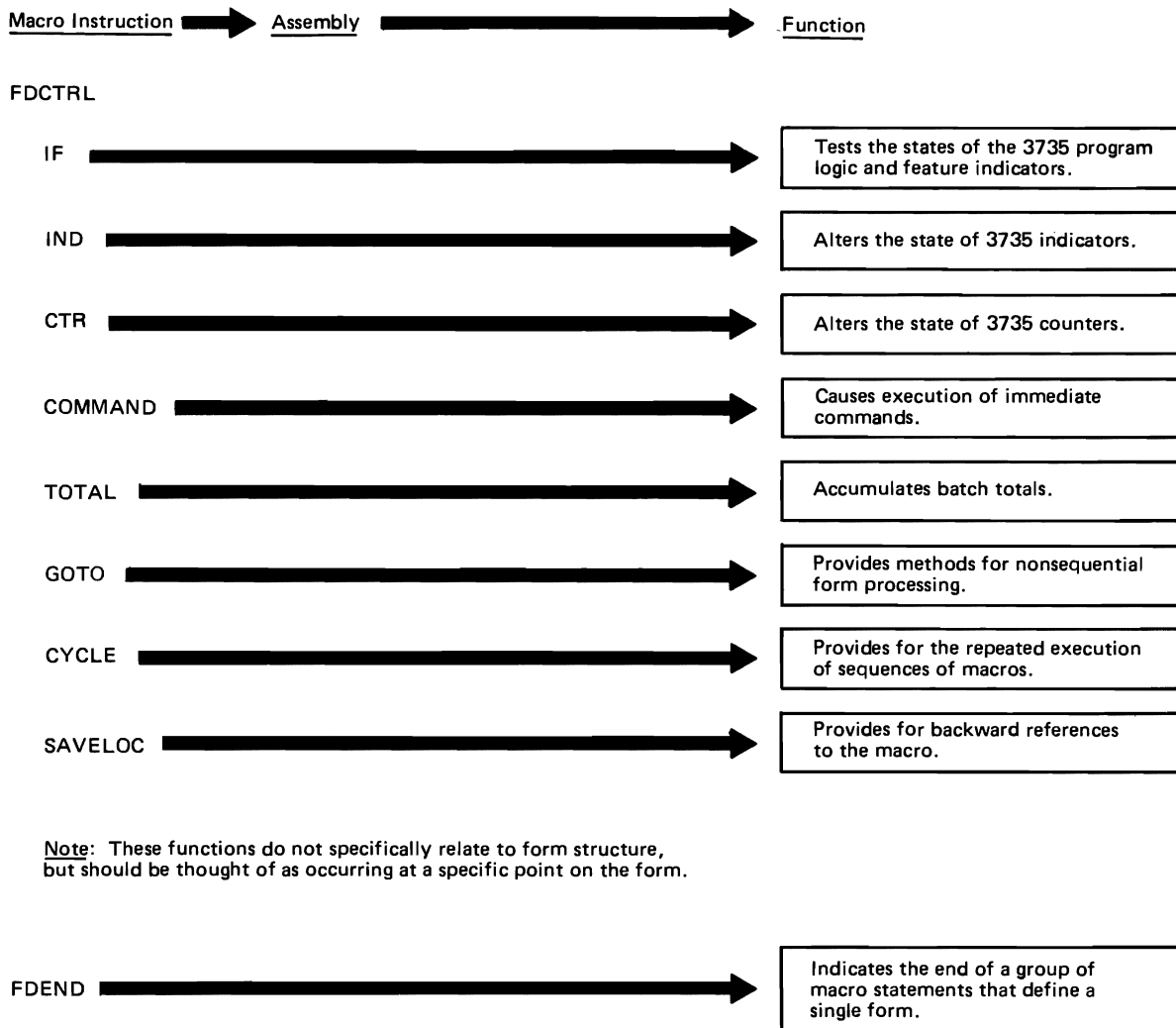


Figure 2-17. Summary of Assembled FDCTRL and FDEND Macro Functions

to character pairs in internal 3735 code, and then assembles each character pair into the FD program header as the first three byte pairs in the header. The fourth byte pair is assembled from the two bits that indicate the PACKING=DELIMIT and PACKING=YES specifications. The assembler now places unpacked and filled binary zeros in the next four byte pairs of the header. Later the FDEND macro will overlay these zeros with the number of lines in the forms.

If the MESSAGE operand is specified on the FDFORM macro statement, the TR inner macro translates the character string coded in the operand to 3735 code and assembles that code into message byte pairs in the FD program header.

If the HTAB operand is specified on the FDFORM macro statement, the ASM inner macro assembles into the header unpacked and filled X'7F'. This byte is followed

by pairs of bytes containing the number of positions that separate adjacent horizontal tabular stops. These tabular stops are from the horizontal tabulation vector in the IDFGBL (OS) or IJLFGBL (DOS) copy book.

The ASM inner macro, finally, assembles a one-byte pair of unpacked and filled binary zero to complete the FD program header assembly. Figure 2-19 in Part 2, Section 3 illustrates the format of the assembled FD program header.

#### Address Resolution and Motion Control Assembly

The FD macros assemble sequences of the following bytes to transfer control sequentially or nonsequentially between FCDs and immediate bytes and to control the position of the Selectric print element.

1. The ASM inner macro assembles the end control bytes of the FCD.

2. ASM assembles the Selectric motion immediate byte: X'70' plus an end control byte, where the end control byte indicates the type of Selectric motion to be performed: space, backspace, tab, line feed, or new line. Refer to the Selectric immediate byte discussion in Part 2, Section 3 for additional information.
3. ASM assembles the GOTO branching address in the format 61 A1A2, where
  - 61 indicates a GOTO immediate byte;
  - A1A2 indicate a two-byte address representing the relative sector and byte displacements needed to reach the destination address.

Refer to the GOTO immediate byte discussion in Part 2, Section 3 for more detailed information.

### Immediate Byte Assembly

After the address and motion control assembly, the assembly of the immediate command bytes proceeds according to the following steps.

1. The ASM inner macro assembles the NOP immediate byte to fill unused space.
2. ASM assembles the three GOTO immediate bytes, the first byte of which causes an unconditional branch to the FD program byte indicated by the A1A2 address in the following bytes.
3. If the IF operand is specified in the FDCTRL macro, ASM assembles the conditional GOTO immediate bytes. These bytes specify a branching address provided an IF indicator specifies such a branch.
4. If a CYCLE operand is specified, ASM assembles:
  - a. First, the seven begin cycle immediate bytes in the form 63LLRRA1A2, where
    - LL represents the total number of Selectric form lines in the cycle;
    - RR represents the total number of cycle repetitions;
    - A1A2 represent the branch to the target macro.
  - b. Second, the repeat cycle immediate byte to signal the point at which the cycle repeats.
  - c. Finally, the end cycle immediate byte to cause the 3735 to index the number of lines remaining in the LL subfield and to do a carrier return.
5. The ASM inner macro assembles the two index space immediate bytes to indicate the number of spaces needed for spacing from the margin stop to the first column of a new line on a Selectric form entered using line-feed motion.
6. If the TOTAL operand is specified, ASM assembles the seven total batch immediate bytes to scan records created by execution of FD programs.
7. If the CTR operand in the FDCTRL macro is specified, ASM assembles the three clear counter immediate bytes to clear a specified counter to +0.

8. If the IND operand in the FDCTRL macro is specified, ASM assembles the three set indicator immediate bytes to cause the 3735 to set (=1), reset (=0), or invert the specified indicator.
9. The ASM inner macro assembles next the two Selectric command immediate bytes.
10. If the COMMAND operand is so specified, ASM assembles:
  - a. The two clear STG immediate bytes to clear the buffer associated with the specified device attached to the 3735.
  - b. The two inquiry command immediate bytes to clear the INQ buffer, to request input to the INQ buffer, or to send output to the INQ buffer.
  - c. The two 5496 command immediate bytes to perform operations on the 5496 buffer.
  - d. The two or four 3286-3 line printer command immediate bytes to perform operations on the line printer buffer and on the 3286-3, itself.
  - e. The two operator ID reader command immediate bytes to perform operations on the IDR and CCR buffer.
11. If the COMMAND operand is specified, ASM assembles:
  - a. The cancel form immediate byte to cause the 3735 to cancel forms.
  - b. The end form immediate byte to cause the 3735 to end the form, creating a record of its execution in the CPU data directory.

### Field Control Descriptor Assembly

After the address resolution and motion control assembly, the assembly of the FCD proceeds according to the following steps.

1. For FDFIELD macro statements with the BATCH operand specified, the ASM inner macro assembles into the FCD the batch group, which consists of operation code bytes X'6700' that are unpacked and filled. A byte pair derived from the batch number completes the batch group bytes.
2. The ASM inner macro assembles the data source group according to the following conditions:
  - a. If the data source is not the keyboard, the ASM macro assembles a source-code byte pair from the value obtained by concatenating the value B'101' and four bits of the data source global variable.
  - b. If FID, RSN, or 'string' is specified, the ASM macro does not assemble a second part.
  - c. If the source is CTR, the ASM inner macro assembles the counter number as a single arithmetic-value byte pair.
  - d. If the source is a buffer, ASM assembles the starting-character location as two arithmetic-value byte pairs.

*Note:* The outer FD macro saves the location of the first source type code byte to be used for future reference.

3. The ASM inner macro assembles the data-type byte pair, which contains:
  - a. Katakana, alphabetic, or numeric data-type bits from the data source global variable;
  - b. Transmit-as-entered and print-as-entered flag bits;
  - c. Validity flag bit;
  - d. Function flag bit.

Except for the following conditions, the byte pair uses the data source global bits as they are:

  - a. If SOURCE='string' is specified, the assembly forces KIND=U (no testing) to be indicated.
  - b. If SOURCE=CTR (d), SINK=CTR (d), PICTURE, the numeric COMPARE, or IND operands are specified, the assembly forces KIND=N to be indicated. The late discovery of this condition forces the reassembly of this byte pair.

The flag bit settings indicate certain conditions as follows:

  - a. If SINK=TMT is specified with left justify, blank fill, and no PICTURE operand, the transmit-as-entered flag is on.
  - b. If, in addition to the previous conditions, SINK=PRT is specified, the print-as-entered flag is on (unless SINK=(PRT,AFTER) is specified).
  - c. If SOURCE='string' is specified, the validity flag is off. Otherwise, if NUMPAD, minimum or exact count, and COMPARE are specified; OPTIONAL is not specified; and SELFCHK is specified not NO, the validity flag is on.
  - d. The function flag is on to indicate one or more of the following:
    - (1) CTR or IND is specified;
    - (2) COMPARE operand is chained through more than one statement;
    - (3) a sink other than PRT or TMT is specified;
    - (4) editing is specified for PRT or TMT, or SINK=(PRT,AFTER).
4. The ASM inner macro assembles the character-count byte pair, an arithmetic value. The character count is determined according to the SOURCE operand as follows: FID and RSN indicate three characters and CTR (d) indicates ten characters. The string length (K'-2) is used if the SOURCE operand is not chained; otherwise, the character count is from the COUNT operand that specifies an exact value.

For SOURCE=KBD, the macro chooses the character count by the following priority: first, if the exact or maximum COUNT operand is specified, the ASM inner macro gets the character count from this operand. Second if field lower and upper bounds are specified, the inner macro calculates the character count indicated in the bounds. Finally, if neither of

the above are available, the inner macro selects the character count from the field lower bound and explicit LNG (d) specification.

The ASM inner macro also assembles the character count for buffers from several alternatives according to a priority. The following list indicates the choices from first to last:

- a. Buffer lower and upper bounds.
- b. Buffer explicit LNG (d) operand specification.
- c. Exact COUNT operand specification.
- d. Field lower and upper bounds.
- e. Field explicit LNG (d) operand specification.

If SOURCE='string' is coded, the TR and ASM inner macros translate, unpack, and fill the entire string during assembly processing. The string assembly generates a character count. If the actual count does not agree with the preassembled count, the ASM macro reassembles the count unless the string is to be chained.

5. The ASM inner macro assembles the validity byte pair only when the validity flag is on in the data-type byte pair (see step 3). The first three bits (0-2) represent attributes of the SOURCE=KBD operand; the next two bits (3 and 4) control the testing of the numbers of characters entered and are mutually exclusive; the last 2 bits (5 and 6) control the testing of the COMPARE and SELFCHK functions. Bits 0, 1, 2, and 6 are derived from the data source global variable in the IDFGBL (OS) or IJLFGBL (DOS) copy book. The bits and their indications when the validity flag is on are as follows:

Bit	Indication
0	NUMPAD operand is coded
1	AUTOEOF is not coded
2	OPTIONAL is not coded
3	exact COUNT is coded
4	minimum COUNT is coded
5	COMPARE operand is coded
6	SELFCHK operand is not coded "NO"

6. The ASM inner macro assembles the function byte pair only if the function flag is on in the data-type byte pair (see step 3). Flags in the function byte pair indicate functions that are to be performed:
  - a. Performance of arithmetic operations on counters to include one or more clear-and-add sequences as needed to provide for the SINK=CTR (d) specification.
  - b. Comparison of entered data for setting program logic indicators.
  - c. Control of either a buffer sink or the PRT or TMT sinks with any data format other than left justified with blank fill.

If none of these indicated functions is needed, but the function flag is on in the data bytes because the COMPARE operand is continued, the ASM inner

macro assembles the function byte pair with no flags on. The terminal control program treats this function byte pair as a no-operation.

7. The ASM inner macro assembles the minimum character-count byte pair from the COUNT operand only when the minimum count flag is on in the validity byte pair (see step 5). The byte pair represents the arithmetic value coded as COUNT=(MIN,n). The inner macro does not assemble a value of one for this function.
8. The ASM inner macro assembles the value-check group when the COMPARE operand is specified. This group consists of one or more repetitions of the following sequence:
  - a. The ASM macro assembles the value-check control byte pair from bits representing a single comparison operator as a combination of NOT (optional) with LESS, EQUAL, or GREATER (mutually exclusive) plus a logical indicator AND or OR if more comparisons follow.
  - b. ASM assembles the comparand character count byte pair from the value of the number of comparand characters, less two if the comparand is a string, and retains this value for comparison with the number of byte pairs actually assembled.
  - c. The TR and ASM inner macros process the comparand characters, and, if the resulting count does not agree with the assembled count, ASM reassembles the count.
9. The ASM inner macro assembles the self-check byte pair (SELFCHK operand) from three bits of the data source global variable.
10. If a sink of CTR (d) or the CTR operand is specified, the ASM inner macro assembles the arithmetic group. A sink of CTR (d) causes the assembly of a byte pair made up of flag bits for clear and add operations, followed by a byte pair containing the value of the counter number.

If both SINK=CTR (d) and the CTR operand are specified, the flag byte pair for sink is produced first with a chaining flag on.

If only the CTR operand is specified, the operand causes the assembly of one or more double byte pairs like those just described, except that the clear indicator is never on. The flags indicate one of the mutually exclusive operations add (ADD), subtract (SUB), multiply (MPY), divide (DIV), or divide and round (DVR).
11. The ASM inner macro assembles the compare group consisting of one or more subgroups, one for each indicator specified in the IND operand. Each subgroup comprises the following pieces:
  - a. The comparison operator byte pair— assembled from flags for the relational operators LT, EQ, and GT with (optional) NOT, plus one flag bit for the mutually exclusive connectives AND or OR if they are specified. The last comparison operator byte pair of each subgroup except the last has a chaining flag on.
  - b. The comparand character count byte pair— derived from the number of characters in a comparand, exclusive of any framing apostrophes. The assembled byte pair contains a tentative value that is to be checked later for validity.
  - c. The comparand character byte pairs— processed by the TR and ASM inner macros from the specified comparand. If the resulting actual count differs from the tentative count assembled (see b. above), ASM reassembles the count.
12. If a buffered sink is specified, or if the PRT or TMT operands are coded with editing specified other than left justified with trailing blanks, the ASM macro assembles the data sink group. If assembled, the group consists of one to five subgroups, one for each sink specified, with the subgroups for PRT and TMT being the last one or two assembled if they are required. The assembly of each subgroup is as follows:
  - a. The ASM inner macro assembles from bits in the data sink global variable the sink-type byte pair, which includes a chaining flag derived from the OR of a scratch copy of the usability flags of data sink global. The assembly of each sink subgroup causes a corresponding scratch bit to be turned off, so that when the last subgroup is assembled the chaining flag is off.
  - b. ASM assembles the start-byte-0 byte pair from the data sink global bits combined to call for centering, right justification with blank fill or zero fill, or PICTURE operand editing, which are all mutually exclusive. In addition, if the adjusted sink starting position exceeds 127, bit 6 is on. The user-specified sink starting position must be changed to 0 origin and then incremented by an offset of four.
  - c. ASM assembles a start-byte-1 byte pair to contain the remainder of the starting position after division by 128.
  - d. ASM assembles the number of sink characters into the next byte pair unless the PICTURE specification is used. For PICTURE, the assembled byte pair contains a tentative count equal to the number of picture characters exclusive of framing apostrophes. The TR and ASM inner macros process the PICTURE string assembling the picture characters and a delimiting X'7F' character, accumulating a count of the digit and insertion characters in the picture string. If this accumulated count does not equal the previously assembled count, ASM reassembles the count.

There are some assembly modifications for the PRT and TMT operands. The start-byte quadruplet is replaced by a control-byte byte pair that is similar to the start-byte-0 byte pair except that bit 4 is the UL indicator. Bit 5 is the PRT indicator and bit 6 is the TMT indicator. The UL and TMT specifications are mutually exclusive, as are the PRT and TMT operands.

The method for deriving the buffer sink character count is the same as that for buffer sources. No sink count is required for the CTR (d), PRT, or TMT specifications. Refer to step 4 for the explanation of the buffer source character count.

For detailed descriptions of the internal bytes that comprise the FD programs, proceed to the Form Description Program Organization section that follows.

## Section 3. Form Description Program Organization

### FORM DESCRIPTION PROGRAMS

The Form Description macros translate user-encoded options into FD programs that become input to the FD utility. A user-written program consists of an FDFORM macro, other FD macros, and an FDEND macro. Similarly, the generated FD program consists of an FD program header, immediate bytes and field control descriptors (FCDs), and an FD program trailer. Refer to Figure 2-18 for an illustration of the FD program structure.

The IDFASM or IJLFASM (called ASM) inner macro assembles generated FD programs for input to the FD utility. The following discussion abstracts the generated FD programs from their frames of keyed unpacked program blocks (KUPBs), which ASM generates, and concentrates on the generation of bytes that the 3735 terminal can interpret.

#### FD Program Header

An FD program header consists of the following fields: form ID, data format byte, lines form—printer, lines form—Selectric, operator message, tabs, and begin byte.

Figure 2-19 illustrates the format of the FD program header. The following list explains each field in the FD program header.

1. The FID operand routine of the FDFORM outer macro generates the form ID field. The first three bytes of an FD program header must contain a three digit identifier (FID) used to select the FD program. The ID represents the number by which the 3735 operator may select an FD program. The IDs must be in internal 3735 code and may be between 000 and 989, inclusive. The IDs of 990 through 998 are reserved for IBM supplied resident FD programs, and ID 999 is for the functional test form. If one of these special IDs is transmitted to the 3735 from the CPU, it is accepted by the terminal control program. However, if one of the special IDs is selected by the 3735 operator, the 3735 performs the special function indicated not the user program.
2. The PACKING operand routine of the FDFORM outer macro generates the data format byte (also called the packing field). The data format byte specifies the format of field data (collected under FCD control) transmitted by the 3735. The bit significant data format is as follows:

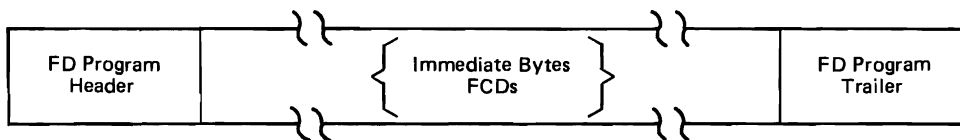


Figure 2-18. The FD Program Structure

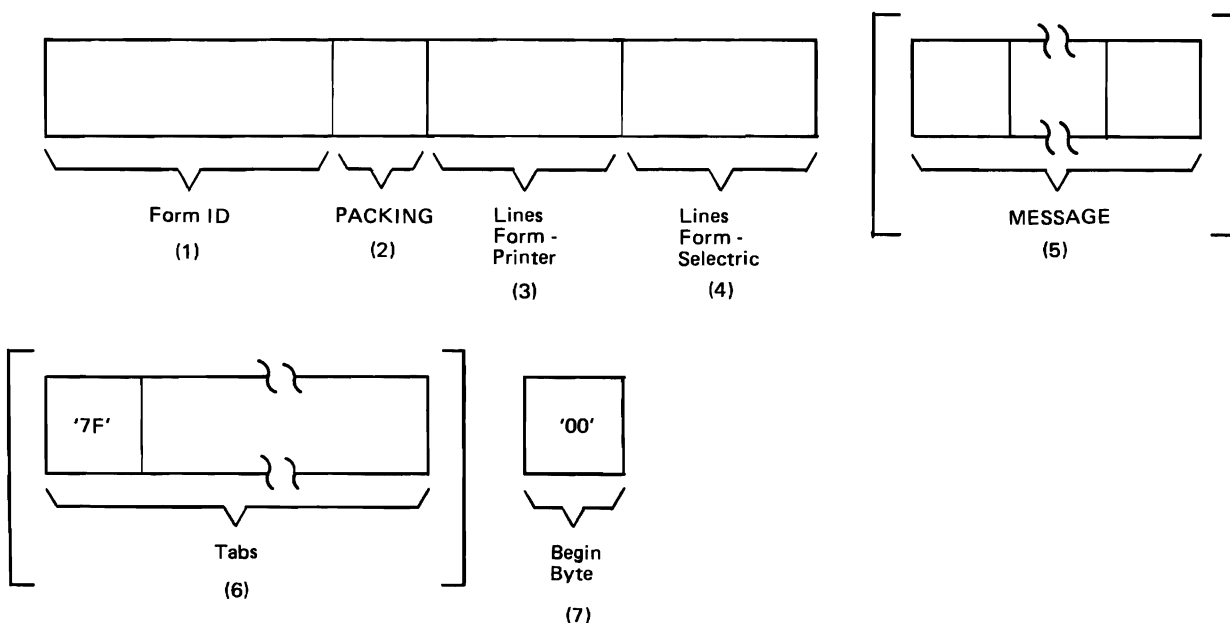


Figure 2-19. The FD Program Header

- a. Blank fill (bit 0=0, bit 1=0): indicates that variable length fields are to be padded with blanks to the right of the last character entered. For example, the 3735 transmits a seven character field as ABCbbbb if only three characters are entered, or as ABCDEFG if all seven characters are entered.
  - b. Packed with delimiter (bit 0=1, bit 1=0) strip trailing blanks: indicates that the 3735 transmits all fields with a delimiter following the last character entered. For example, the 3735 transmits a seven character field as ABC\* (with \* representing the X'1C' file separator character) if only three characters are entered, or as ABCDEFG\* if all seven characters are entered.
  - c. Packed with no delimiter (bit 0=0, bit 1=1): indicates that the 3735 transmits all fields without trailing blanks or a delimiter. For example, the terminal transmits a seven character field as ABC if only three characters are entered, or as ABCDEFG if all seven characters are entered.
3. The FDFORM macro generates the two-byte lines form—printer field (in binary) with an initial value of zero. The FDEND macro overlays the zero value with the correct value of the number of lines in the form. This field is required, whether the 3286-3 printer is attached or not.
  4. The FDFORM macro generates the two-byte lines form—Selectric field (in binary) with an initial value of zero. The FDEND macro overlays the zero value with the correct value when it does its processing. The two lines form—Selectric bytes specify a count of the total number of lines from the beginning of a Selectric application to the beginning of the same application on the next sequential form. The 3735 terminal control program decrements this count by one for each new line or line feed operation it performs. The terminal control program uses the resulting count as an index to the start of the next form whenever it ends or cancels a record. The terminal control program does not decrement a zero count, but does a carrier return to the left hand margin at the beginning of the target form.

*Note:* The operator message is not included in the lines form—Selectric byte count.

5. The MESSAGE operand routine of the IN01 inner macro generates the operator message field in character data. An optional operator message may be included in the FD program to inform the 3735 operator of form usage, set-up procedure, and so forth. The terminal operator may bypass the message printout if he desires.

The terminal control program requires that type element positional control characters be imbedded within the message if such positioning is needed. Tabs should not be used because the terminal control program sets tabs after it prints the message. The terminal control program issues a carrier return before it begins printing and a carrier return just after the printout.

The length of the message is not restricted. The terminal control program requires a message delimiter: a X'7F' if tab information follows, or a X'00' if there are not tab definitions. Message characters must be in internal 3735 code. The 3735 operator can retrieve the message only on the Selectric® I/O II device.

6. The HTAB operand routine of the IN01 inner macro generates the horizontal tabulation intervals (tabs) field beginning with the delimiter X'7F'. The terminal control program forces the type element to the left margin stop and spaces the print ball t0 times before stopping it. The 3735 operator then sets the tab and depresses the OPER key to continue. The terminal control program continues this procedure until it finds a X'00' delimiter. The t's represent binary numbers with decimal values between 1 and 127, inclusive.

The 3735 operator can bypass the tab setting procedure by depressing the ENTER key. The terminal control program does not restrict the number of tab stops. However, there cannot be tab stops at character positions greater than 128 without at least one stop at a character position less than 129.

The FD program does not use a tab operation to advance to a field whose first print position is adjacent to the last printed position of the preceding field. The Selectric device does not recognize a tab stop at character position one.

7. The begin byte generator routine of the IN01 inner macro generates the begin byte. The begin byte contains a X'00' and marks the beginning of the bytes that exercise control over the data entered on the 3735. The terminal operator can begin FCD control by performing one of the following sequences.

- The operator selects an FD program by entering an ID.
- The operator presses one of the following:
  - a. OPER — The operator receives an operator message and begins the tab setting procedure. When the tab stops are set, FCD control begins.
  - b. TAB — The operator does not receive an operator message and begins the tab setting procedure. When the tab stops are set, FCD control begins.
  - c. ENTER — The operator bypasses both the operator message and the tab setting procedure. FCD control begins immediately.

#### Immediate Bytes and Field Control Descriptors

The second part of an FD program consists of sequences of immediate bytes or field control descriptors (hereafter called FCDs), or both. Each FCD consists of a variable number of bytes, the function of each byte is defined by its format and its position within the FCD. Defining a document field requires a minimum of three bytes: a data-type byte, a character-count byte, and an end control byte. The data-type byte defines the initial data checking require-



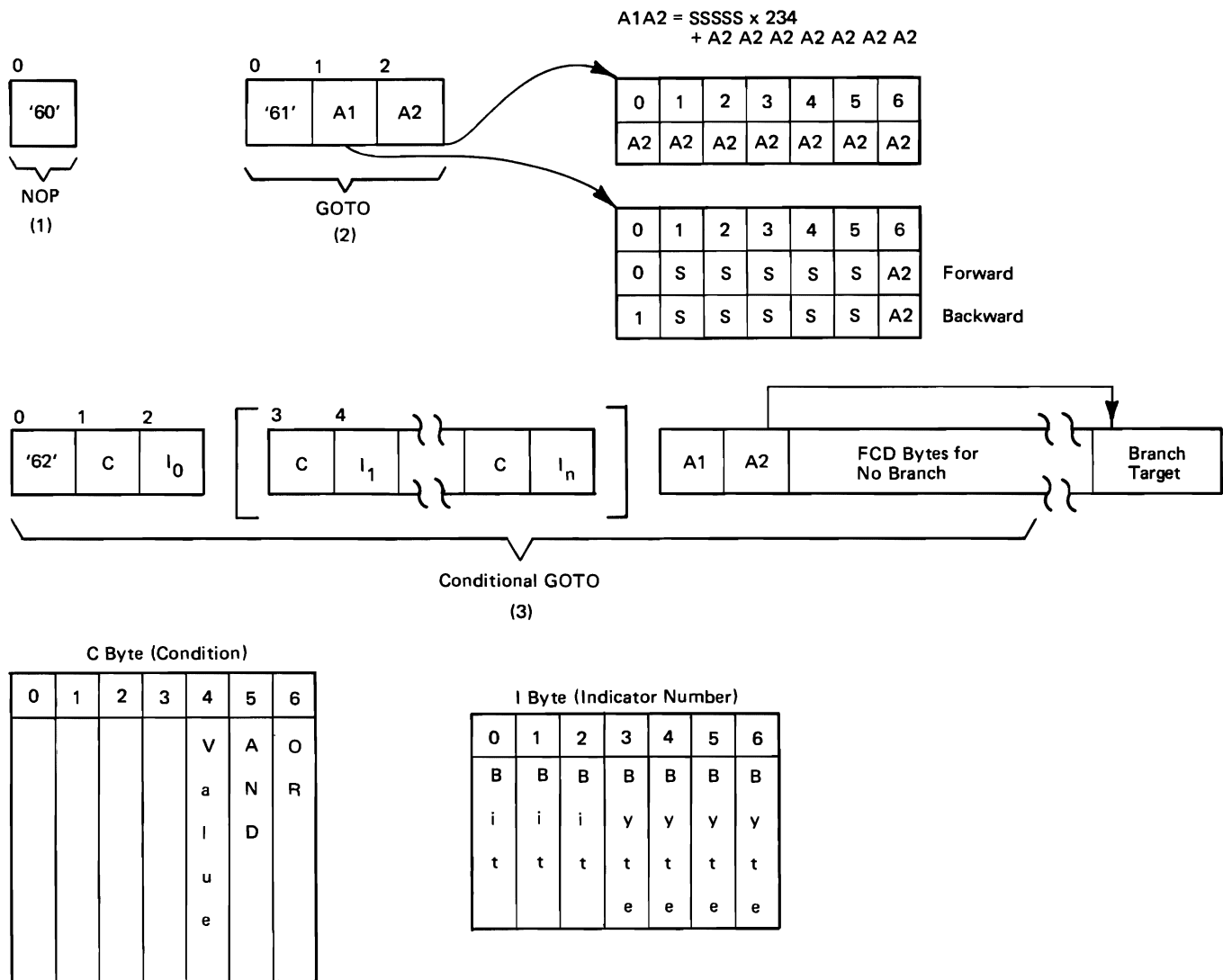


Figure 2-20. NOP, GOTO, and Conditional GOTO Immediate Bytes

ments. The character-count byte provides a binary number equal to the maximum number of characters that can be entered into the field. The end control byte provides for the movement of the Selectric carrier or platen to position the type element.

In the absence of an explicit specification, the 3735 terminal control program assumes that input data originates from the Selectric® I/O II keyboard. The FD macro programming support assumes the same default source. If a different source is used, the FCD must contain a data source byte to indicate the source.

Immediate command bytes are interspersed with FCDs to provide functions not directly related to the control of data being entered into a field. The immediate bytes may indicate an instruction which is to be completed immediately, or may indicate the location of the next byte to be examined by the terminal control program.

The FDPAGE, FDLINE, FDFIELD, FDCTRL, and FDEND macros generate these portions of an FD program.

The following describes the structure of the immediate bytes and of the FCDs.

#### Immediate Bytes

Immediate bytes cause the 3735 terminal to control devices, to branch within an FD program, or to change the contents of internal storage areas. Each immediate byte field is explained in the following list.

1. The NOP immediate byte contains a 3735 no-operation code, generated to hold space in the assembled data. This byte has no effect on program execution. Refer to Figure 2-20 for the NOP byte format.
2. The end control routine (in IN03) or the GOTO operand routine (in IN10) creates the three GOTO immediate bytes. The first byte causes an unconditional branch to the FD program byte that is indicated by the displacement in the two bytes that follow, A1 and A2. The GOTO immediate bytes provide for an

unconditional logic branch to another part of the FD program. The A1 and A2 address bytes represent the relative sector and byte displacements needed to reach the destination address. The byte displacements (relative to the A2 address byte location plus one) is calculated by multiplying the binary number SSSSS by 234 and adding the binary number A2A2A2A2A2A2A2 to the result. Refer to Figure 2-20 for an illustration of the GOTO immediate bytes.

3. The IF operand routine (in IN10) generates the conditional GOTO immediate bytes. These bytes cause the 3735 to branch to the location indicated in the A1 and A2 bytes if a specified indicator combination indicates the branch. The GOTO on condition (or conditional GOTO) immediate bytes provide conditional logic branching to another part of the FD program. The conditions for branching are based on the set/reset state of one or more indicators (I0 – In).

If the specified indicator combination is evaluated to a logical one, the GOTO branch target is to the A1 A2 address. If the combination is evaluated to zero, the terminal control program examines the FCDs following the A1 A2 address bytes. The combination may be expressed in the form  $I_0 = 0/1$  [AND/OR  $I_1=0/1 \dots$  AND/OR  $I_n = 0/1$ ]. The result of the logical expression is as if the AND functions are evaluated first. For example,  $F=I_1$  or  $I_2$  AND  $I_3$  OR  $I_4$  is evaluated as  $F=I_1$  OR  $(I_2$  AND  $I_3)$  OR  $I_4$ . The AND/OR functions are specified by setting the corresponding bit in the condition byte. The last condition byte cannot have an AND/OR bit set on. The indicator value to be tested for is specified in the value bit. The indicator number is specified in the indicator byte; the low order four bits specify one of the 16 indicator bytes, and high order three bits specify the bit within the byte. The byte number can range from X'0' to X'F', inclusive, and the bit number can range from X'0' to X'6', inclusive, yielding a total of 112 indicators.

The 3735 indicators fall into three types: general purpose, special, and feature indicators.

- a. General purpose indicators:

Indicators with an I-byte from X'00' to X'6B', inclusive, have unrestricted usage. The user may set, reset, or invert each or all of these 84 indicators. The indicator ALL functions refer to these 84 general purpose indicators. The 3735 terminal control program resets these indicators to zero at the beginning of each FD program execution.

- b. Special indicators:

The seven indicators of I-byte X'0C' are read-only indicators. The user may not set, reset, or invert these indicators. The 3735 terminal control program resets these indicators at the beginning of each FD program and sets them whenever the set-

ting condition occurs. The seven indicators are as follows:

- |       |  |
|-------|--|
| Bit 0 | Indicates playback mode. The terminal control program resets this indicator for all modes except playback.   |
| Bit 1 | Indicates 5496 end of card file. The terminal control program sets this bit when a /* is read from the 5496 card reader-punch and resets this indicator for all other device read operations.          |
| Bit 2 | Indicates inquiry timeout. The terminal control program sets this bit when an inquiry send command has not been completed successfully and resets this indicator whenever a send command is initiated. |
| Bit 3 | Reserved.  |
| Bit 4 | Reserved.  |
| Bit 5 | Not used.  |
| Bit 6 | Not used.  |

- c. Feature indicators:

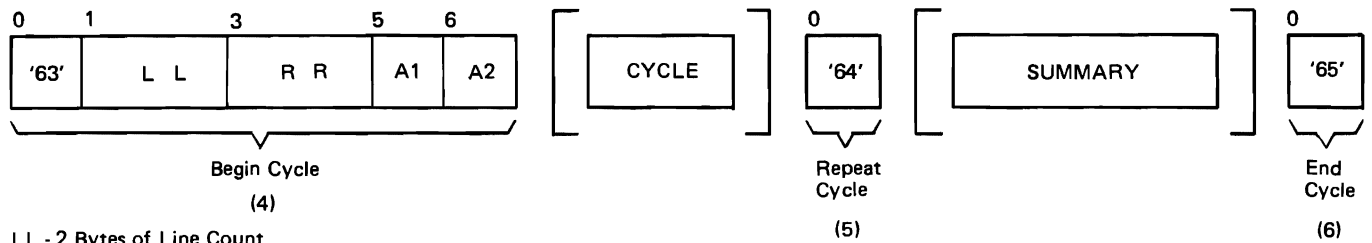
Each group of seven indicators with an I-byte from X'0D' to X'6F', inclusive, are read-only indicators representing the featured attachments that are supported for the 3735 terminal. The indicators are as follows:

- |                       |  |
|-----------------------|--|
| Indicator X'0D'       | Indicates an attached 5496                           |
| Indicator X'1D'       | Indicates an attached 3286-3                         |
| Indicator X'2D'       | Indicates an attached operator identification reader |
| Indicator X'3D'       | Reserved   |
| Indicator X'4D'–X'6F' | Not used   |

During enter-form mode, the terminal control program stores a delimiter on disk to indicate whether or not a conditional GOTO branch has been taken. The terminal control program tests indicators X'00' to X'6B', inclusive, during playback or error-correct mode to determine whether a branch was taken in enter-form mode. If contrary branching occurs, the terminal control program issues an operator message (invalid data path) and exits to local mode. Tests on indicators X'0C' to X'6F', inclusive, result in the same data path as taken in enter-form mode whether or not the new test results in a different branch/no branch condition. Refer to Figure 2-20 for an illustration of the conditional GOTO immediate bytes.

4. The begin cycle (CYCLE operand part 2) routine (in IN03) generates the seven begin-cycle immediate bytes initially in the form X'63', 8X'60'. After assembly, these bytes finally appear in the form 63 LL RR A1A2.

- a. The LL bytes represent the total Selectric form lines encompassed in the cycle. The end summary



LL - 2 Bytes of Line Count

RR - 2 Bytes of Repeats

Figure 2-21. The Begin, Repeat, and End Cycle Immediate Bytes

- routine (CYCLE operand part 5) in the IN01 inner macro updates these two bytes.
- b. The RR bytes represent the total number of cycle repetitions.
  - c. The A1A2 bytes represent the branch to the next FCD bytes to be examined. The cycle limit (CYCLE operand part 3) routine (in IN11) updates these bytes. Refer to Figure 2-21 for the formats of the cycle bytes.
5. The cycle limit (CYCLE operand part 3) routine creates the repeat cycle immediate byte to signal the point at which the cycle repeats. Each time a repeat cycle command is encountered, the terminal control program decrements the number of repeats by one and transfers control to the first FCD byte following the A2 byte. The control program ends the cycle when the number of repeats is zero.
  6. The cycle limit (CYCLE operand part 3) routine creates the end cycle immediate byte. This byte causes the 3735 to index the number of lines remaining from the LL subfield and to do a carrier return. The 3735 decrements the count of the number of lines for each Selectric printer line passed over. Following the carrier return, the terminal control program examines the immediately following FCD byte.
- Note:* The begin cycle, repeat cycle, and end cycle immediate bytes are generated in that order. The FCDs and immediate bytes between the begin cycle and repeat cycle immediate bytes correspond to the macros between the start-of-cycle macro and the limit macro, inclusive. The FCDs and immediate bytes between the repeat cycle and the end cycle immediate bytes correspond to the macros comprising a summary block.
7. The end control routine generates the two index space immediate bytes. The value of the byte n is a binary count of the number of spaces from the left margin stop to the first field of a new line on the Selectric form that has been entered using line feed motion (without carrier return). The 3735 terminal control program uses this immediate command in error correction. The index space function should follow the last line feed end control or Selectric command. Refer to Figure 2-22 for an illustration of the index space bytes.
  8. The TOTAL operand routine (in IN09) generates the seven total batch immediate bytes. The bytes cause the terminal control program to scan the records created by execution of the FD program whose identifier is given by the subfield FFF. The batch subgroup identifies the FCDs in the FD program as belonging to the same batch as the B subfield of the total batch immediate bytes. These FCDs create data that is accumulated in the counter given by the counter subfield of the total batch immediate bytes. The terminal control program does not clear the counter before the data is accumulated. Refer to Figure 2-22 for the format of the total batch bytes.
  9. The COMMAND operand routine (in IN10) generates the cancel form immediate byte that causes the terminal control program to cancel the form. Refer to Figure 2-23 for the cancel form immediate byte format.
  10. The COMMAND operand routine generates the end form immediate byte. This byte causes the terminal control program to end the form, creating a record of its execution in the CPU data directory. Refer to Figure 2-23 for the format of the end form immediate byte.
  11. The FDCTRL CTR operand routine (in IN09) creates the three clear counter immediate bytes to cause the terminal control program to clear the specified counter to +0. Refer to Figure 2-23 for the format of the clear counter bytes.
  12. The FDCTRL IND operand routine (in IN09) generates the three set indicator immediate bytes to cause the terminal control program to set (=1), reset (=0), or invert the specified indicator. Refer to Figure 2-23 for the format of the set indicator bytes.
  13. The end control routine generates the two Selectric command immediate bytes. These bytes provide for the immediate movement of the Selectric carrier and platen. Refer to Figure 2-24 for the format of these immediate bytes.
  14. The COMMAND operand routine generates the two clear STG immediate bytes to cause the terminal control program to clear the buffer. The storage buffer (STG) is a 236-byte buffer that is resident on the

3735 disk. At the beginning of each FD program, the terminal control program saves the contents of this buffer. If a record is canceled, the control pro-

gram restores the buffer to the state it was at the beginning of the record. Refer to Figure 2-25 for the format of the CLEAR STG buffer bytes.

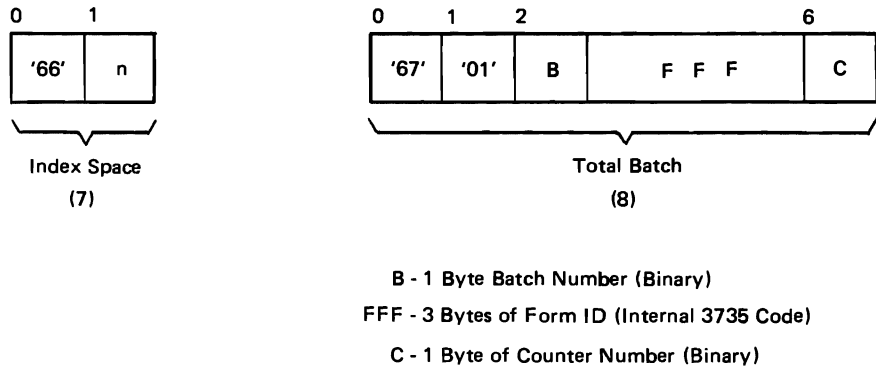


Figure 2-22. The Index Space and Total Batch Immediate Bytes

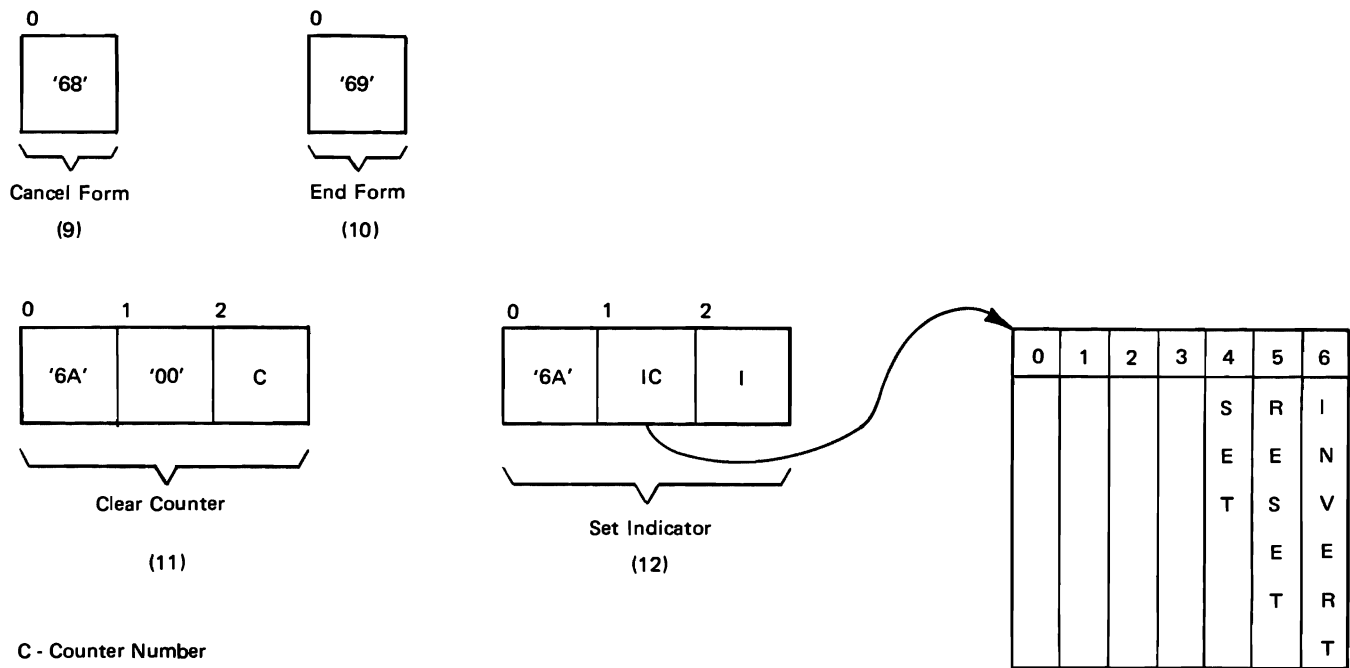


Figure 2-23. The Cancel Form, End Form, Clear Counter, and Set Indicator Immediate Bytes

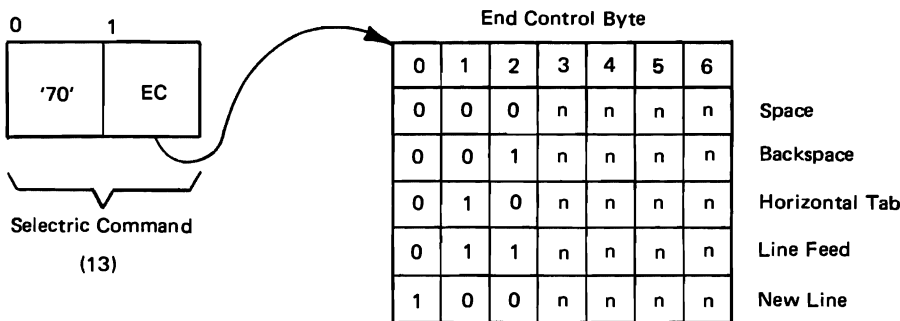


Figure 2-24. The Selectric Command Immediate Bytes

15. The **COMMAND** operand routine generates the two inquiry command immediate bytes to cause the terminal control program to clear the inquiry buffer, transmit the inquiry buffer, or discontinue the line. The inquiry buffer is a 236-byte buffer unit that is resident on the 3735 disk. The terminal control program does not save the contents of this buffer at the beginning of each record. Neither does the terminal control program restore the buffer when the operator enters error-correct mode. Refer to Figure 2-25 for the format of these bytes.
16. The **COMMAND** operand routine generates the two 5496 command immediate bytes to cause the terminal control program to clear the 5496 buffer, request input from the 5496, or send output to the 5496. The 5496 buffer is a 192-byte buffer that resides on the 3735 disk. The terminal control program does not save the contents of this buffer at the beginning of each record. Neither does the control program restore the contents of this buffer when the operator enters error-correct mode. Refer to Figure 2-26 for the formats of the 5496 commands.
17. The **COMMAND** operand routine generates the two or four line printer command immediate bytes to cause the terminal control program to clear the line printer buffer, skip lines on the line printer, or send output to the line printer. Refer to Figure 2-27 for the formats of the line printer command immediate bytes.
18. The **COMMAND** operand routine generates the two operator ID reader command immediate bytes to clear the IDR and CCR buffer, to read the IDR buffer, or to read the CCR buffer. Refer to Figure 2-28 for the formats of the IDR and CCR command immediate bytes.

The following summarizes each immediate byte field, indicating the length of each field and the format.

Type	Length	Format (Hex)
1. NOP Immediate Byte	1	60
2. GOTO Immediate Bytes	3	61A1A2
3. Conditional GOTO Immediate	Variable	62 . . .
4. Begin Cycle Immediate Bytes	7	63LLRRA1A2
5. Repeat Cycle Immediate Byte	1	64
6. End Cycle Immediate Byte	1	65
7. Index Space Immediate Bytes	2	66n
8. Total Batch Immediate Bytes	7	6701BFFFctr
9. Cancel Form Immediate Byte	1	68
10. End Form Immediate Byte	1	69
11. Clear Counter Immediate Bytes	3	6A00ctr

12. Set Indicator Immediate Bytes	3	6Aindctrl
13. Selectric Command Immediate Bytes	2	70ec
14. Clear STG Immediate Bytes	2	7140
15. Inquiry Command Immediate Bytes	2	72inqctrl
16. 5496 Command Immediate Bytes	2	73ctrl
17. Line Printer Command Immed.	2 or 4	741pctrlnn
18. IDR and CCR Command Immed.	2	75ctrl

Refer to Figures 2-20 through 2-28 for diagrams of each of these immediate bytes.

### Field Control Descriptors

FCDs correspond to **FDFIELD** or to **FDCTRL** macros and cause 3735 operations involving data input, manipulation, and output. An FCD generated each time an **FDFIELD** macro is coded consists of up to ten fields. Each of these fields is explained in the following list. Refer to Figure 2-29 for an illustration of the FCD structure.

1. The batch group is an optional immediate field generated by the **BATCH** operand routine.
2. The data source group is present for all FCDs except those corresponding to **FDFIELD** macros with **SOURCE=KBD** coded. The data source group generator routine (in **IN04**) generates the data source group from internal tables (global variables).

If a field is numeric (the numeric bit is on in the data-type byte), the data source field can have a leading plus or minus sign. Also, for a numeric field, the low order position of the field can have an overpunch to represent a negative number. The terminal control program strips off the overpunch for printing but retains the overpunch for storing right-justified in buffered sinks or on disk for transmission.

A negative source field can have a leading plus or minus sign and be right justified with leading blanks in the source field. As the terminal control program reads the source field, it strips off the leading blanks. If the print bit is on in the data-type byte, the terminal control program prints these leading blanks. The terminal control program makes any further reference to the field (arithmetic, compare, data sink, and so forth) with a character count equal to the source character count minus the leading blanks. Data source input characters are comprised only of the characters from **X'20'** to **X'7E'** in the internal 3735 code, or a cent sign, a plus-or-minus sign, or a logical-not sign. Any violation of requested validity checking (character type, length, range, self check) results in an operator error message indicating a data source error.

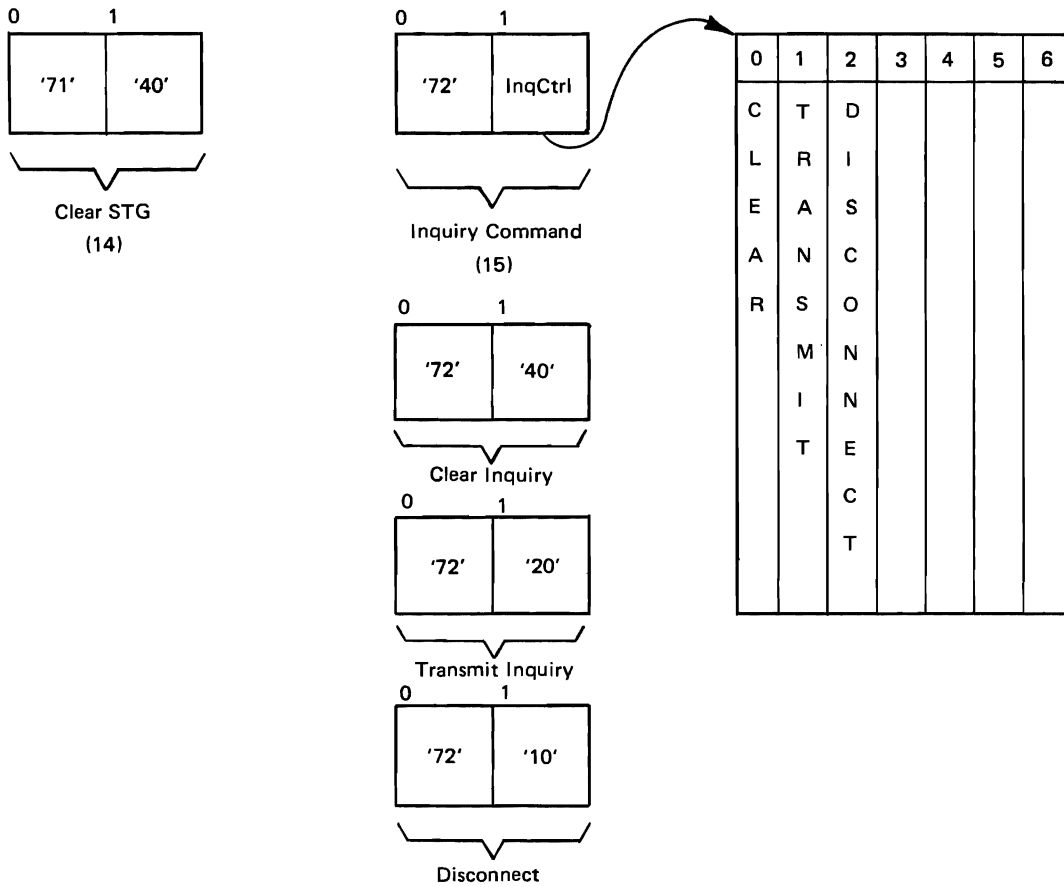


Figure 2-25. The Clear STG and Inquiry Command Immediate Bytes

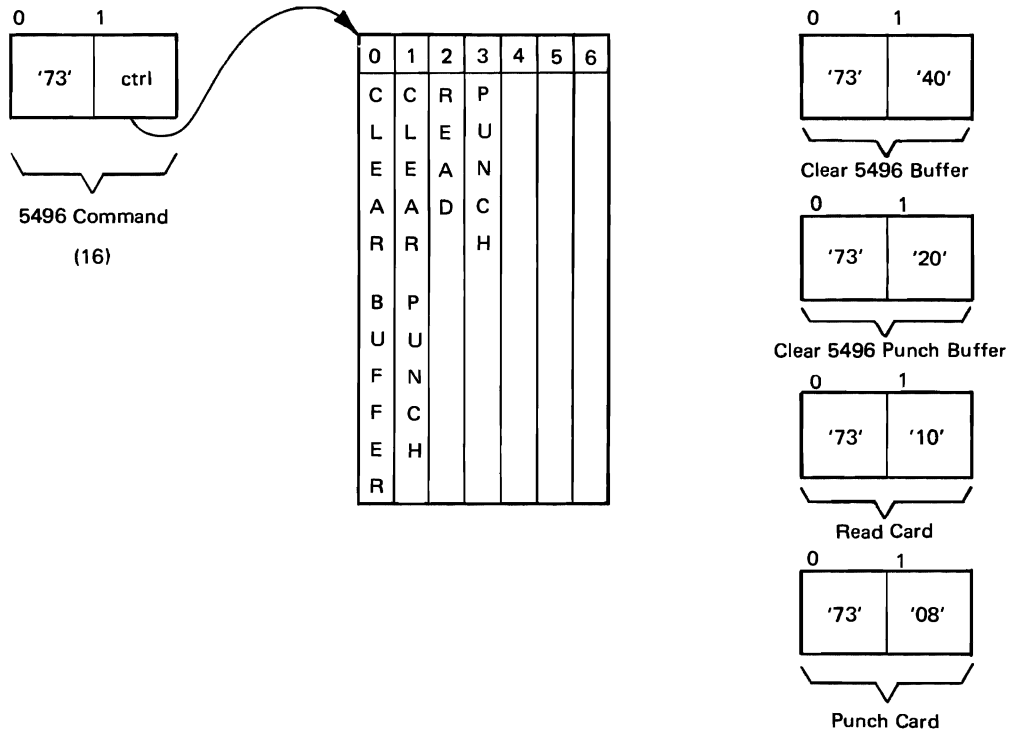


Figure 2-26. The 5496 Command Immediate Bytes

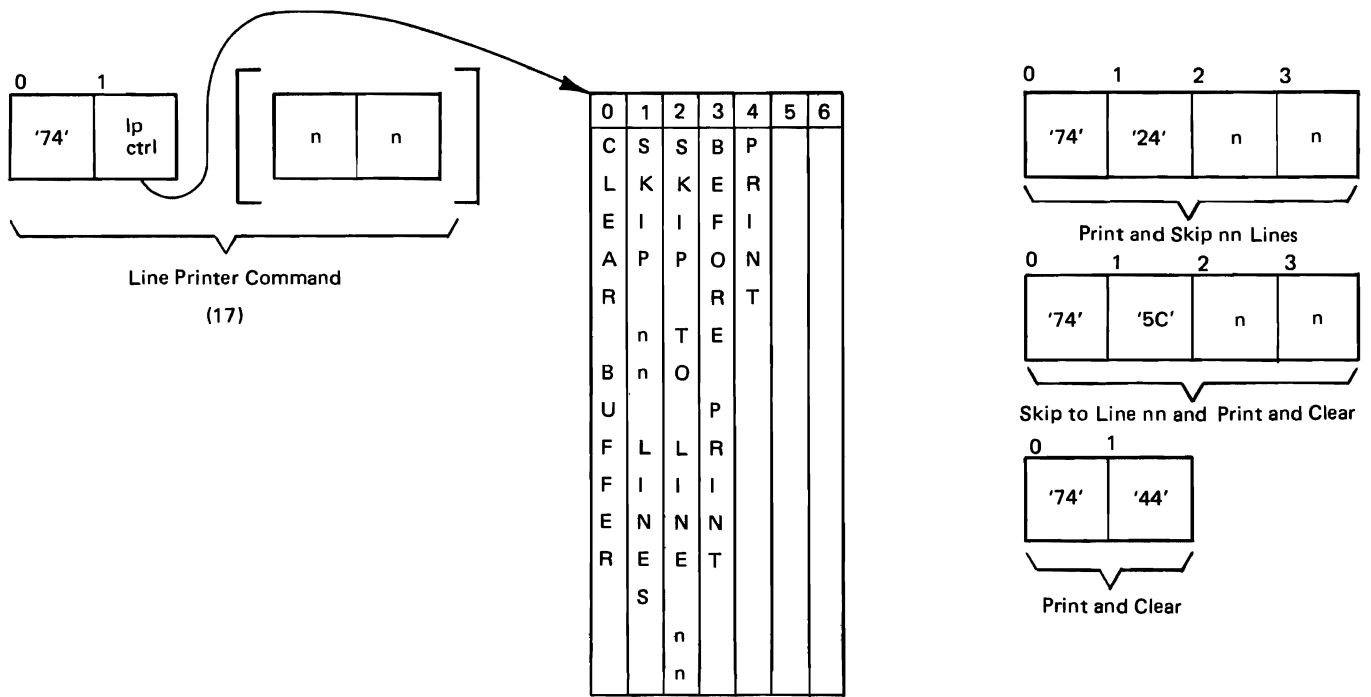


Figure 2-27. The 3286 Line Printer Command Immediate Bytes

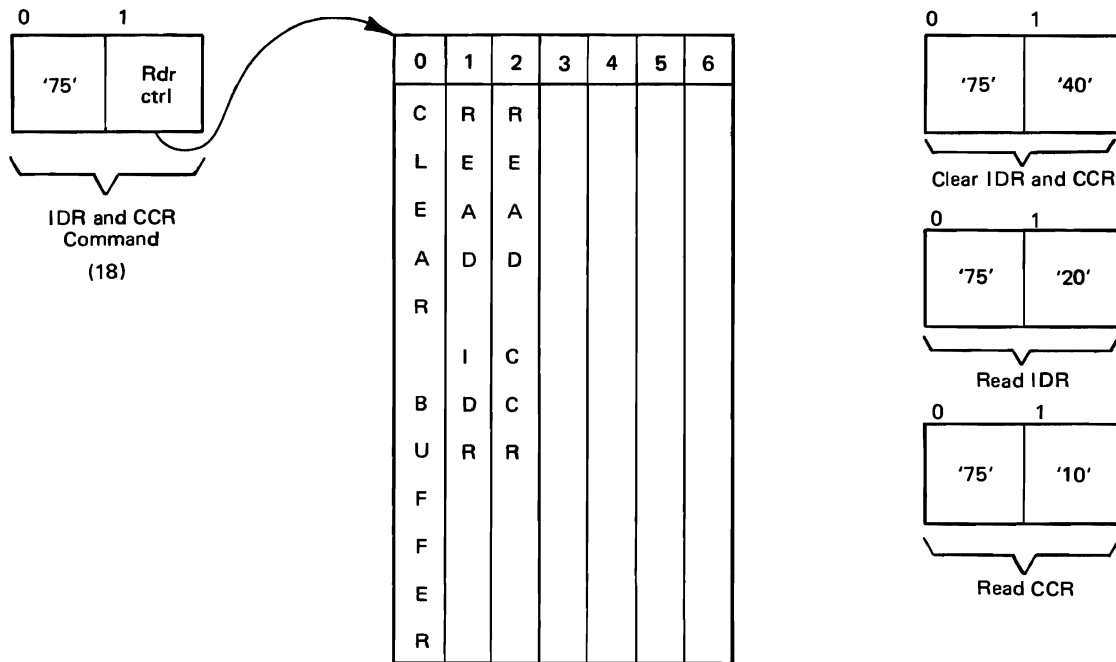


Figure 2-28. The IDR and CCR Command Immediate Bytes

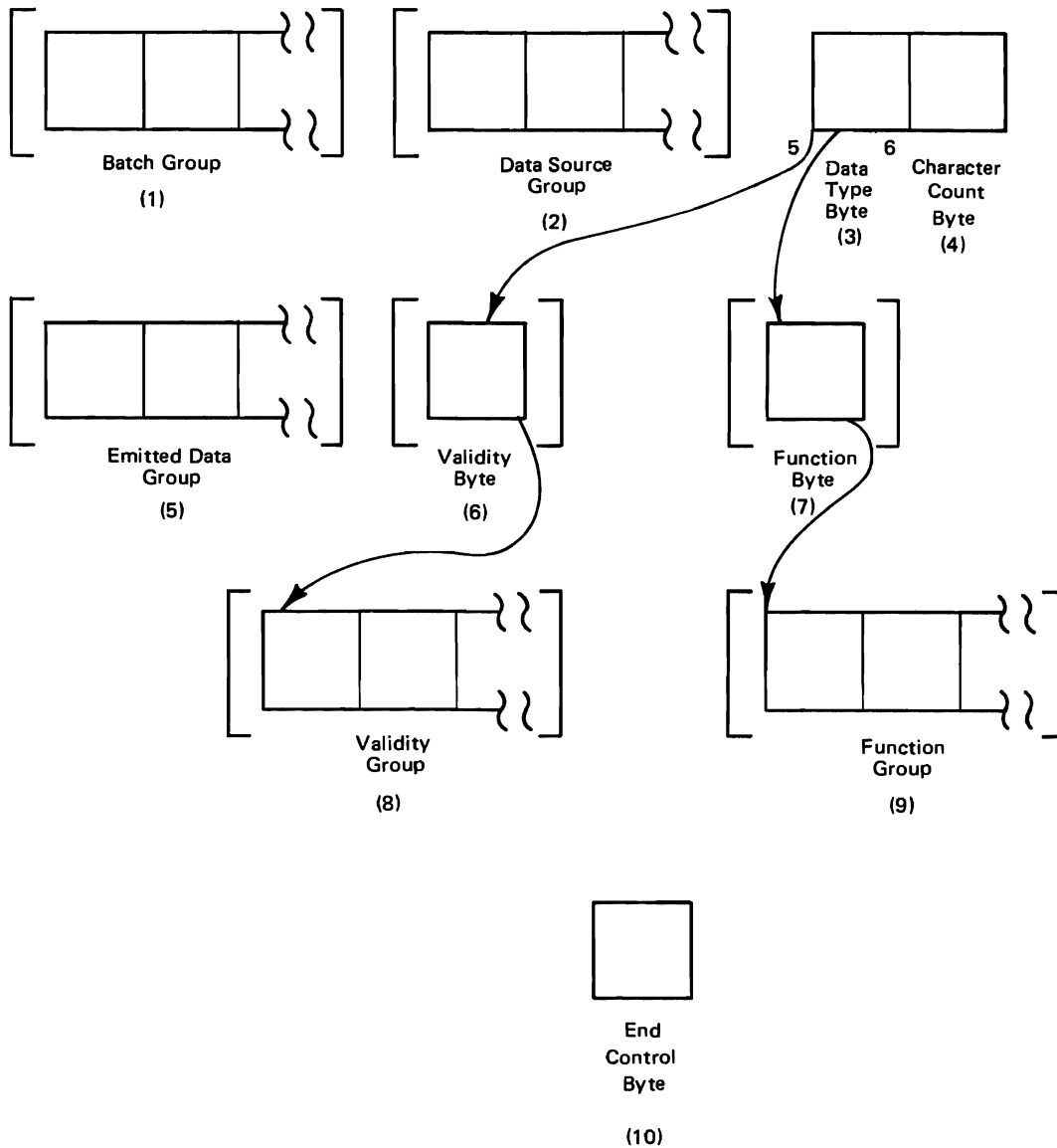


Figure 2-29. The FCD Structure

Within the data source group are nine commands to indicate functions that are to be performed. These commands indicate the following sources for input data: FD program ID, record number, emitted data, counters, storage buffer (STG), inquiry buffer (INQ), 5496 buffer, the 3286-3 buffer, and the IDR/CCR buffer. Refer to Figure 2-30 for the format of each of these source data function specifications.

a. The FD program ID command causes the terminal control program to use the FD program identification number (FID) of the active FD program as the input data source. The ID character count must be three. The terminal control program pads zeros to

the left of significant digits. The control program does not store the FD program number on disk unless the transmit bit is on in the data-type byte.

b. The record number command causes the terminal control program to use the record number assigned to the active record as an input data source. The character count must be three and zeros are padded to the left of significant digits. The terminal control program does not store the record number on disk unless the transmit bit is on in the data-type byte.

c. The emitted data command causes the terminal control program to use the FCD bytes immediately



Data Source Byte

Hex	Function
50	FD Program ID
51	Record Number
52	Emitted Data
53	Counters
54	Storage Buffer
55	Inquiry Buffer
56	5496 Buffer
57	3286-3 Buffer
5F	IDR or CCR Buffer

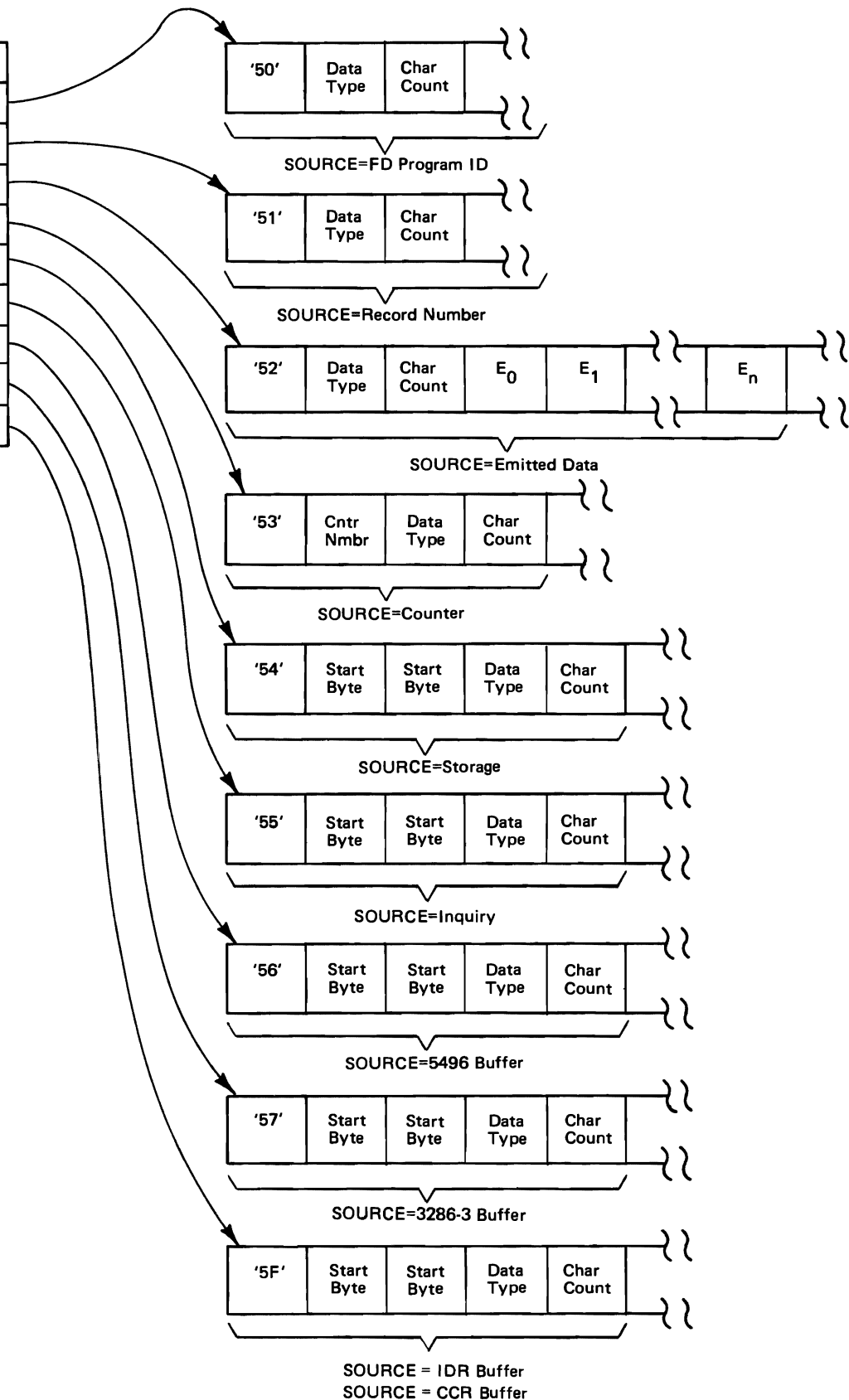


Figure 2-30. The Data Source Group

following the character count as the input data source. If validity and/or function bytes are desired, they must follow the emitted data. The terminal control program does not store emitted data on disk unless the transmit bit is on in the data-type byte.

- d. The counters command causes the terminal control program to use the counter specified by the counter address as the input data source. The character count must be ten. There are 21 ten-digit signed counters, the addresses of which are binary numbers between X'00' and X'14', inclusive. At the beginning of each FD program the terminal control program saves the contents of the counters. If a record is canceled, the control program restores the counters to their state at the beginning of the record. If the operator enters error-correct mode, the terminal control program restores the counters to their values at the beginning of the line. The control program does not save the counters on disk unless the transmit bit is on in the data-type byte. The counters are always ten digits long. If a counter is negative, the units position has an overpunch. The terminal control program takes source data from the counters in all operating modes.
  - e. The storage command causes the terminal control program to use the number of bytes specified in the character count, starting at the specified start-byte location in the storage buffer, as the input data source. The start-byte location is a two-byte binary number from four to 239, inclusive. The terminal control program always stores the source data in the storage buffer on disk. The control program takes the source data from the storage buffer in enter-form mode and error-correct mode and from the disk (data stored in enter-form mode or supplied as CPU data) during playback mode.
  - f. The inquiry command causes the terminal control program to use the number of bytes specified in the character count, starting at the specified start-byte location in the inquiry buffer, as the input data source. The start-byte location is a two-byte binary number from four to 239, inclusive. The terminal control program always stores the source data in the inquiry buffer on disk. The control program takes the source data from the inquiry buffer in enter-form mode and from the disk (data stored during enter-form mode or supplied as CPU data) during error-correct or playback mode.
  - g. The 5496 buffer command causes the terminal control program to use the number of bytes specified in the character count, starting at the specified start-byte location in the 5496 buffer, as the input data source. The start-byte location is a two-byte binary number from four to 195, inclusive. The terminal control program always stores the source data from the 5496 buffer on disk. The control program takes the source data from the 5496 buffer during enter-form mode and from the disk (data stored during enter-form mode or supplied as CPU data) during error-correct and playback modes.
  - h. The 3286-3 buffer command causes the terminal control program to use the number of bytes specified in the character count, starting at the specified start-byte location in the 3286-3 buffer, as the input data source. The start-byte location is a two-byte binary number from four to 239, inclusive. The terminal control program always stores the source data from the 3286-3 buffer on disk. The control program takes the source data from the 3286-3 buffer during enter-form mode and from the disk (data stored during enter-form mode or supplied as CPU data) during error-correct and playback modes.
  - i. The IDR or CCR command causes the terminal control program to use the number of bytes specified in the character count, starting at the specified start-byte location in the IDR or CCR buffer, as the input data source. The start-byte location is a two-byte binary number from 196 to 239, inclusive. The terminal control program always stores the source data in the IDR or CCR buffer on disk. The control program takes the source data from the IDR or CCR buffer during enter-form mode and from the disk (data stored during enter-form mode or supplied as CPU data) during error-correct and playback modes.
3. The data-type field (one byte) is a required field for all FCDs. The data-type byte generator routine (in IN06) creates this byte from internal tables (global variables). Other routines may update the data-type byte because the value of this byte is subject to change after its initial generation.
    - The data-type byte is present for all input/output fields and specifies the following:
      - the field character type
      - whether or not the field is to be printed as entered
      - whether or not the raw input data is to be transmitted to the 3735
      - whether or not any additional validity and/or function bytes follow
- Within the data-type byte are combinations of bit settings that indicate the following character checking functions:
- a. No character check (bit 0=0, bit 1=0, bit 2=0): indicates that any and all characters are accepted at the 3735.

- b. Katakana code (bit 0=1, bit 1=0, bit 2=0): indicates that kana characters and space only will be accepted at the 3735.
  - c. Alphabetic character check (bit 0=0, bit 1=1, bit 2=0): indicates that only A through Z (upper and lower case) or space are accepted at the 3735.
  - d. Numeric character check (bit 0=0, bit 1=0, bit 2=1): indicates that only 0 through 9 and an optional leading sign are accepted at the 3735. The leading sign may be either plus or minus and must be the first keyed character in the field.
  - e. Alphanumeric character check (bit 0=0, bit 1=1, bit 2=1): indicates that only A through Z (upper and lower case), 0 through 9, or space are accepted at the 3735. The terminal control program rejects a leading sign.
  - f. Transmit function: indicates that the terminal control program flags the input data for the associated field for unedited transmission to the CPU if this bit is on.
  - g. Print Selectric function: indicates that the terminal control program prints the associated input field while it is being entered if this bit is on. When the bit is off and the field is to be printed, look for a Selectric data sink byte to follow to perform that function.
  - h. Validity check: indicates that the terminal control program performs additional checks on the input data if this bit is on. A validity byte that immediately follows the character-count byte defines the additional checks.
  - i. Function check: indicates that the terminal control program specifies the presence of a function byte to control arithmetic, compare, and data sink functions if this bit is on. The function byte immediately follows the character-count byte or, if one is present, the validity byte.
- Refer to Figure 2-31 for an illustration of these bit settings.
4. The character-count byte is a required field for all FCDs. The character count generator routine (in IN06) creates this byte from internal tables (global variables) and other routines may update the byte since the value of it may not be known at the time of its initial generation.
 

The character-count byte contains a binary number equal to the exact or maximum number of characters that can be entered into a field. The maximum number of characters that can define a field is 127.

The character count of a numeric field includes an optional leading sign. The character count of a self-check field includes the self-check digit. The character count of a field that is to have the self-check digit generated does not include the self-check digit. Once the digit has been generated, any compare or data sink function should assume that the field contains the additional digit.
  5. The emitted data group is an optional field and is present for all FCDs corresponding to FDFIELD macros coded with SOURCE='string'. The emitted data group generator routine (in IN06) creates this field.

Data-Type Byte

0	1	2	3	4	5	6
K	A	N	T	P	V	F
A	L	U	R	R	A	U
T	P	M	A	I	L	N
A	H	E	N	N	I	C
K	A	R	S	T	D	T
A	B	I	M		I	I
N	E	C	I	S	T	O
A	T		T	S	E	N
	I			L	E	
	C			C	T	
				R		
				I		

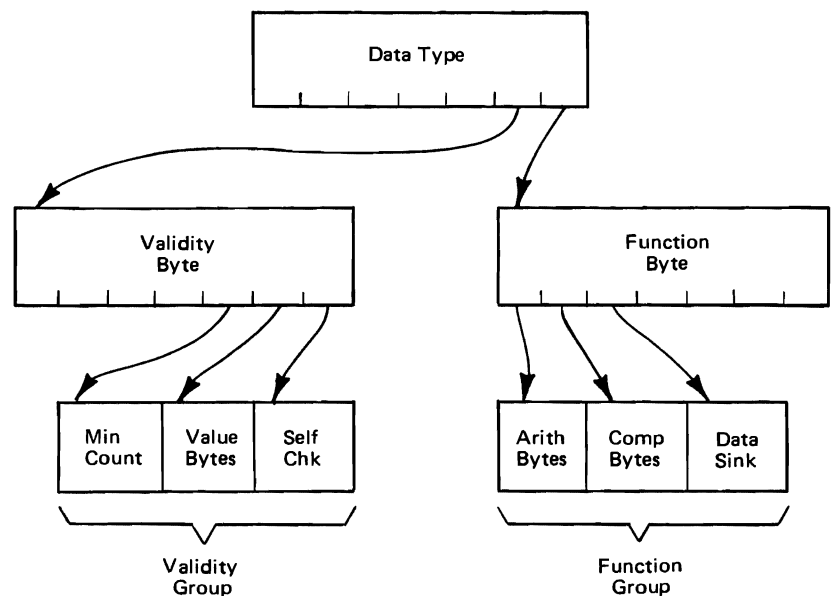


Figure 2-31. The Data-Type Byte

6. The validity byte is an optional field generated by the validity byte generator routine in the IN06 inner macro. The validity byte is present if the validity bit (bit 5) in the data-type byte is on. The validity checks are not mutually exclusive. These checks control the number of characters entered into a field and the content of the data. The validity byte indicates seven conditions: numeric (key) pad enable, enter required, mandatory field, fixed length field, minimum character count in a field, numeric/alphabetic value check, and checking of a self-check number.

The numeric-pad-enable bit enables the ten-key pad. If this bit is off, or if the validity byte is not present, the terminal control program interprets the keypad character in the normal way. Enabling the keypad changes the keypad characters to their numeric values. If a field is to be checked for numeric characters and the keypad is enabled, the 3735 accepts the ten-key pad and the regular numeric characters. Because some alphabetic characters are not usable in the numeric check, this bit should be turned on when using numeric pad.

The enter-required bit indicates that the operator must press the ENTER key or the TAB key to exit from the field.

The mandatory-field bit indicates that the operator must enter at least one character into the field. If this bit is off and no characters are entered at the 3735, the terminal does not perform the validity checks specified. If the operator tries to bypass the mandatory field, the terminal control program turns on the error indicator.

The fixed-length bit indicates that if an entry is made, the operator must enter the exact number of characters specified in the character-count byte. If this condition is violated, the terminal control program turns on this length indicator.

The minimum-count bit, assuming that no function byte is present in the FCD, indicates that the immediately following byte contains a binary number. This number is equal to the minimum number of characters that can be entered into the field, if an entry is made, before the operator can advance to the next field. The minimum-count byte is not present for a minimum of one character.

The value-check bit indicates that a check is to be made on the numeric and/or alphabetic value of a field. Refer to paragraph 8 for an explanation of the value-check byte.

The self-check bit indicates that a check is to be made of a self-checking number or that a self-check digit is to be generated. Refer to paragraph 8 for an explanation of the self-check byte. Refer to Figures 2-32 and 2-33 for illustrations of the validity byte and of the validity group.

7. The function byte is an optional field generated by the function byte generator routine in the IN06 inner macro. This byte is present if the function bit (bit 6) in the data-type byte is turned on. The function byte indicates the presence of these functions: arithmetic, compare, and data sink functions.

The arithmetic-function bit indicates an operation to be done on a counter. The field data operates on counter X and the results are stored in counter X. Refer to paragraph 9 for an explanation of the arithmetic byte in the function group.

The compare-function bit indicates compare bytes are present in the function group. Refer to paragraph 9 for an explanation of the compare bytes.

The data-sink bit indicates that data sinks are present in the function group. Refer to paragraph 9 for an explanation of the data sink bytes.

Refer to Figures 2-32 and 2-34 for illustrations of the function byte and the function group.

8. The validity group is an optional field and is present only when bits 4, 5, and 6 in the validity byte (see paragraph 6) indicate its presence.

Bit 4 on    minimum count field is present.

Bit 5 on    value bytes are present.

Bit 6 on    self-check field is present.

The value-check byte provides the ability to check the numeric or alphabetic value of a field. Refer to the conditional GOTO immediate byte explanation for the use of the AND/OR bits in the value-check byte. The greater than, equal to, and less than bits in the value-check byte are mutually exclusive. The NUMB CHAR byte represents the number of characters in the comparand.

For numeric comparisons the (numeric bit is on in the data-type byte), the comparand may contain a leading plus or minus sign. The terminal control program compares the field to the comparand as a signed numeric value and pads the shorter length number with leading zeros for the comparison.

For alphabetic or alphanumeric character string comparisons (the numeric bit in the data-type byte is off), the terminal control program pads the shorter string with trailing blanks for the comparison.

The value-check comparand characters are in internal machine code. The 3735 does not restrict the number of AND/OR functions. A 3735 with ASCII code uses the ASCII collating sequence; a 3735 with EBCDIC code uses the EBCDIC code collating sequence.

The comparand-character-count byte for numeric comparisons should include a leading sign if one is present in the comparand. Signed comparands may have a leading sign. The absence of a leading sign implies a positive number. The terminal control program uses any non-numeric character in a comparand

Validity Byte

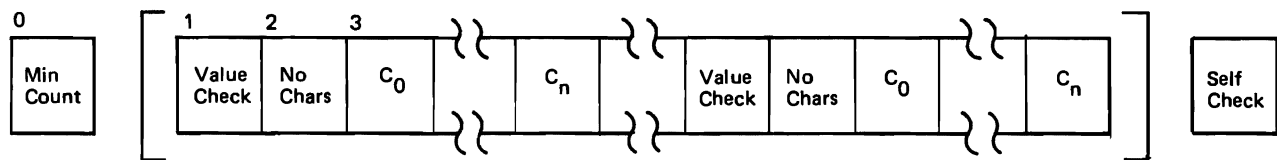
0	1	2	3	4	5	6
K	E	M	F	M	V	S
E	N	A	I	I	A	E
Y	T	N	X	N	L	L
P	E	D	E	I	U	F
A	R	A	D	M	E	
D		T		U		C
	R	O	L	M	C	H
E	E	R	E		H	E
N	Q	Y	N	C	E	C
A	U		G	O	C	K
B	I		T	U	K	
L	R		H	N		
E	E		T			
D						

Function Byte

0	1	2	3	4	5	6
A	C	D				
R	O	A				
I	M	T				
T	P	A				
H	A					
M	R	S				
E	E	I				
T		N				
I		K				
C						

Figure 2-32. The Validity and Function Bytes

Validity Group Bytes



Value Check Byte

0	1	2	3	4	5	6
L	E	G	N		A	O
E	Q	R	O		N	R
S	U	E	T		D	
S	A	A				
	L	T				
		E				
		R				

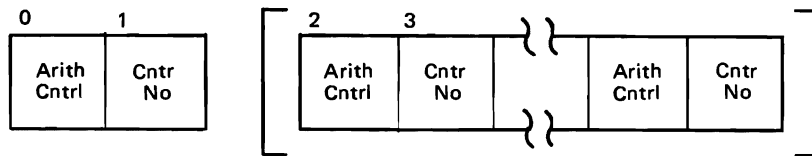
Self Check Byte

0	1	2	3	4	5	6
G	M	M				
E	O	O				
N	D	D				
E	U	U				
R	L	L				
A	O	O				
T						
E	1	1				
	0	1				

Figure 2-33. The Validity Group

## Function Group Bytes

### Arithmetic Bytes



### Arithmetic Control Byte

0	1	2	3	4	5	6
C	A	S	M	D	D	O
L	D	U	U	I	I	P
E	D	B	L	V	V	E
A		T	T	I	I	R
R		R	I	D	D	A
		A	P	E	E	T
		C	L			I
		T	Y	A	A	O
				N	N	N
				D	D	
				T	R	C
				R	O	H
				U	U	A
				N	N	I
				C	D	N
				A		I
				T		N
				E		G

Figure 2-34. The Function Group (Part 1 of 3)

value checked against a numeric field in the compare operation and assumes a value representative of the machine collating sequence.

The terminal control program automatically cancels any field that the operator enters and that fails to pass the value check. When the operator presses the OPER key to turn off the Range Check light, the terminal control program cancels the field.

The self-check byte provides a check of a self-checking number or the generation of a self-check digit. If the generate bit (bit 0) is off, the 3735 checks the field using the specified modulus routine (modulo 10 or 11). If the operator enters the field and the self-check test fails, the 3735 turns on the Self-Check light and locks the keyboard. The terminal control program automatically cancels the field when the operator presses the OPER key.

When the generate bit is on, the 3735 generates the self-check digit and appends it to the input data. Refer to Figure 2-32 for an illustration of the validity byte and to Figure 2-33 for illustrations of the validity group.

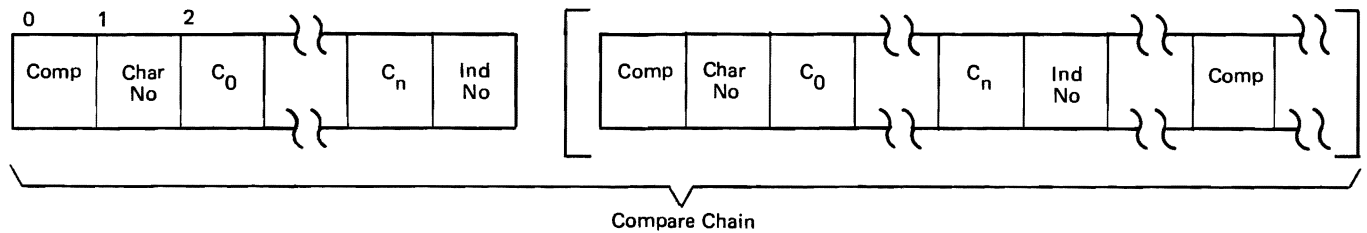
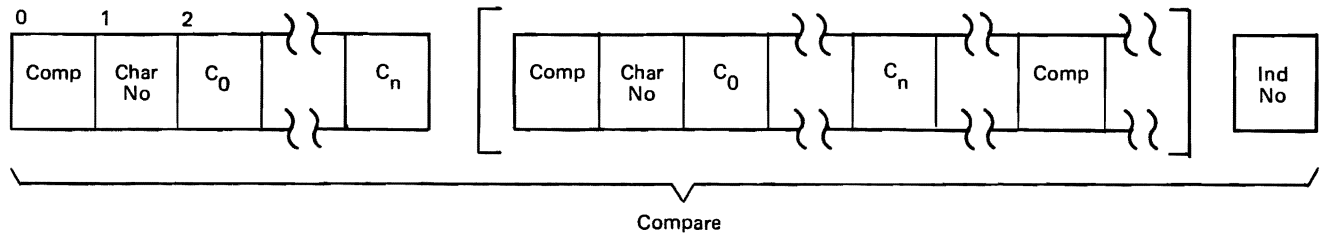
- The function group is an optional field and is present only when the bits in the function byte (see paragraph 7) indicate its presence. Bits 0, 1, and 2 in the function byte indicate that the function group contains the following bytes:

- Bit 0 on arithmetic bytes are present
- Bit 1 on compare bytes are present
- Bit 2 on data sink bytes are present

The arithmetic byte indicates that field data operates on counter X and the results are stored in counter X. The add, subtract, multiply, divide, and divide-and-round bits are mutually exclusive. The clear bit may

## Function Group Bytes

### Compare Bytes



### Compare Byte

0	1	2	3	4	5	6
L	E	G	N	C	A	O
E	Q	R	O	H	N	R
S	U	E	T	A	D	
S	A	A		I		
	L	T		N		
		E				
		R				

Figure 2-34. The Function Group (Part 2 of 3)

be on with another arithmetic function. If a counter is ever set to zero as the result of an arithmetic function, it is a plus zero. The terminal control program performs arithmetic functions in all operation modes. Refer to Figure 2-34 for an explanation of the arithmetic bytes.

Within this byte reside seven indicators:

- Bit 0 indicates a clear operation
- Bit 1 indicates an add operation
- Bit 2 indicates a subtract operation
- Bit 3 indicates a multiply operation
- Bit 4 indicates a divide and truncate operation
- Bit 5 indicates a divide and round operation
- Bit 6 indicates chaining to another arithmetic operation

If bit 0 is on, the 3735 clears the specified counter to plus zero. The counter number is X'00' to X'14'. The clear bit may be on with one other function; however, the terminal performs the clear operation first.

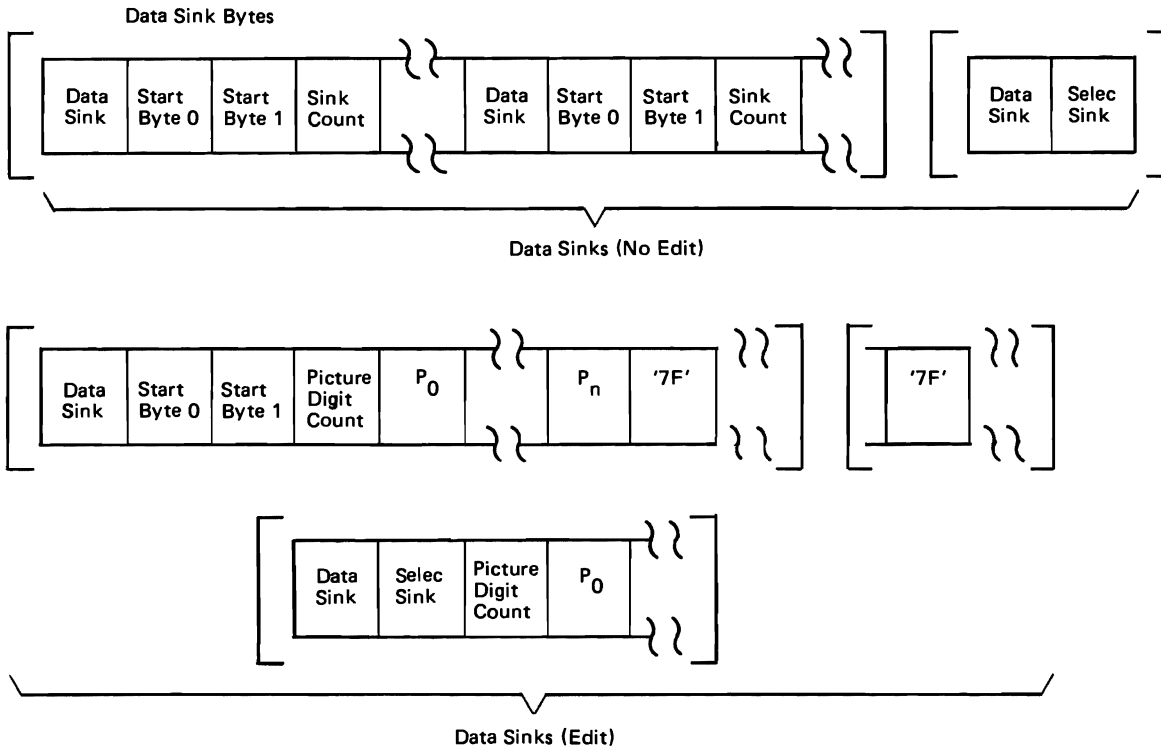
If bit 1 is on, the 3735 adds the field to the counter and stores the result in the counter specified.

If bit 2 is on, the 3735 subtracts the field from the counter and stores the result in the counter.

If bit 3 is on, the 3735 multiplies the counter by the field and stores the result in the counter.

If bit 4 is on, the 3735 divides the counter by the field and stores the result in the counter. The 3735 disregards any remainder. If an attempt is made to divide by zero, the 3735 clears the counter to plus zero.

Function Group Bytes



Data Sink Byte

0	1	2	3	4	5	6	
SINK CHAIN			0	0	0	0	Data Sink
			0	0	0	1	Selectric
			0	0	1	0	Storage
			0	0	1	1	Inquiry
			0	1	0	0	5496 Buffer
							3286-3 Buffer

Start Byte 0

0	1	2	3	4	5	6
C	R	R	E			B <sub>1</sub>
E	I	I	D			
N	G	G	I			
T	H	H	T			
E	T	T				
R	B	Z				
	L	E				
	A	R				
	N	O				
	K					

Selectric Data Sink Byte

0	1	2	3	4	5	6
C	R	R	E	U	P	T
E	I	I	D	N	R	R
N	G	G	I	D	I	R
T	H	H	T	E	N	A
E	T	T		R	N	S
R				L	S	M
	B	Z		I	E	I
	L	E		N	E	T
	A	R		E		
	N	O				
	K					

Start Byte 1

0	1	2	3	4	5	6
B <sub>2</sub>	B <sub>2</sub>	B <sub>2</sub>	B <sub>2</sub>	B <sub>2</sub>	B <sub>2</sub>	B <sub>2</sub>

Figure 2-34. The Function Group (Part 3 of 3)



If bit 5 is on, the 3735 divides the counter by the field and stores the result in the counter. The terminal uses any remainder to half-adjust the quotient. If an attempt is made to divide by zero, the 3735 clears the counter to plus zero.

If bit 6 is on, the 3735 looks for another arithmetic control byte and continues performing arithmetic functions.

The compare bytes provide the ability to check the numeric or alphabetic value of a field. Refer to the conditional GOTO immediate explanation for the use of the AND/OR bits in the compare bytes. The compare bytes use the results of the comparisons to set or reset an indicator. If the results of the compare logical function is true, the 3735 sets the specified indicator. If the expression is false, the terminal resets the indicator. The terminal control program executes the compare function in all operation modes.

Within a compare byte reside seven indicators:

Bit 0	indicates a less than comparison
Bit 1	indicates an equal comparison
Bit 2	indicates a greater than comparison
Bit 3	indicates a not qualification of bits 0, 1, or 2
Bit 4	indicates a compare chain
Bit 5	indicates an and logic operation
Bit 6	indicates an or logic operation

If the compare-chain bit is on, the 3735 uses the logical expression evaluated to this point to set or reset the indicator specified in the byte following the last comparand byte for that segment of the chain. Once the indicator is set or reset, the 3735 treats the following compare function as if it were the only compare function.

The 3735 uses the data sink bytes to store data in the buffered sinks and to control printing and/or editing. The terminal performs the data sink functions in the order they are encountered and uses the sink chain bit (bit 0) to indicate additional data sink functions.

If the sink count or picture digit count for numeric fields (the numeric bit is on in the data-type byte) is less than the number of digits to be placed in the sink, the 3735 uses the low-order digits of the source. Refer to Figure 2-34 for detailed illustrations of the data sink bytes.

For all buffer sinks, the data sink byte has several fields following it: the starting-byte location in the buffer in which the resultant data is to be stored, the size of the sink field (maximum of 127), and optional picture bytes. The high-order four bits of the starting-byte location specify left justify, right justify with leading blanks, right justify with leading zeros, center, or edit. These functions are mutually exclusive. The 3735 stores no data on disk (as a part of the created

record) for the buffered sink functions. The terminal stores only into the 5496 buffer during playback mode.

The 3735 does the left-justify function when all sink function bits are off in the start-byte-zero byte. The terminal stores the input field in the buffer left justified. If the sink count is greater than the source count (character count), the 3735 pads the buffer field with trailing blanks. If the input field is numeric (the numeric bit is on in the data-type byte) and contains a negative sign overpunch in the low-order-digit position, the 3735 transfers the overpunched digit to the sink field if the overpunched digit is stored in the rightmost sink field character position.

For centering, the 3735 centers the input source field into the sink field. If any odd character is to be to the right of the raw data, the terminal should see the right-blank bit in the start-byte-zero byte. The center function centers the input field into the sink. The 3735 considers any blanks in the data as characters in the field to be centered. If the input field is numeric and contains a negative sign overpunch in the low-order-digit position, the 3735 transfers the overpunched digit to the sink field if, and only if, the overpunch is stored in the rightmost sink field character position.

For right-justify with leading blank, the 3735 right justifies the input source field with leading blanks and stores the field in the sink field. The terminal replaces any leading zeros in the source field with leading blanks. If the input field is numeric and contains a negative sign overpunch in the low-order-digit position, the 3735 stores the overpunch in the rightmost sink field character position.

For right-justify with leading zero, the 3735 right justifies the input source field with leading zeros and stores it in the sink field. The terminal replaces any leading blanks in the source field with leading zeros. If the input field is numeric and contains a negative sign overpunch, the 3735 stores the overpunch digit in the rightmost sink field character position.

For the edit function, the picture digit count replaces the sink count byte. This count is equal to the number of picture bytes that have associated digit positions. This count equals the sum of 9s and zero suppression characters in the picture stream. All picture bytes are represented by 3735 internal codes with the following exceptions:

- The blank insertion byte should be a X'20';
- The radix symbol is used for stopping zero suppression only, and should be an uppercase V if no suppression characters follow, or a lowercase v if suppression characters do follow.

The 3735 must initiate a drifting sequence with two contiguous drifting characters.

The only way to display the source field sign is to use the picture bytes S, +, or -. Otherwise, the 3735 drops any overpunched digit and assumes any leading sign is a leading zero.

The Selectric sink function controls the printing and or storing for transmission of edited data. For explanation of the right justify, left justify, center, and edit functions, refer to the description of the buffered sinks.

The Selectric data sink functions are as follows:

- Underline: if the underline and print bits are on, the 3735 underlines the field. If the right blank bit is on, the terminal only underlines significant (nonblank) characters. If the center bit is on, the 3735 underlines only characters that are centered. If the left justify bit is on (other bits are off), the 3735 does not underline trailing blanks. If the edit function is used, the 3735 underlines the results of the picture stream.
- Print Selectric: The requested function is to send output to the Selectric device.
- Transmit: The 3735 stores the edited data on disk for transmission to the CPU.

10. The end control byte, generated by the end control routine in the IN03 inner macro, is a required field. The 3735 spaces the Selectric print element to the end of the field before executing the end control functions of space, backspace, or line feed. The terminal control program executes horizontal tabbing and new line functions without spacing to the end of the field. If tabs are set within a variable length field that is not mandatory, code an end control byte of

X'00' followed by an immediate tab command. Even if the print bit in the data-type byte is turned off, the 3735 always executes the end control function. Refer to Figure 2-35 for a summary of the end control byte functions.

Refer to Figure 2-29 for a summary of the FCD structure and to Figures 2-30 through 2-35 for detailed diagrams of each FCD field.

#### FD Program Trailer

An FD program trailer consists of an end form byte, an FD program delimiter, and, optionally, padding to complete a sector (KUPB). The FDEND macro generates these bytes when the user codes an FDEND macro in his source deck. Refer to Figure 2-36 to see the format of the trailer.

#### FD Program Maintenance

This section describes some of the problems that may occur at some time in the generation of a form processing system.

The trouble-shooting information presented in the following paragraphs may be further augmented by referring to the diagnostic messages and MNOTE messages contained in Part 4, Appendix C of this book and to the sample program in Part 4, Appendix D.

#### Trouble Shooting

Troubles may occur anywhere during the building and generating of a system for processing forms. Difficulties may appear at macro assembly time, utility processing time, during the transmission of data to the 3735, and during the execution of FD programs. The following discussions explain some of the conditions that may occur and describe steps to be taken to correct these situations.

#### End Control Byte

0	1	2	3	4	5	6	Function
0	0	0	n	n	n	n	Space (0000000 indicates space out to end of field before doing next motion-end control noop)
0	0	1	n	n	n	n	Backspace
0	1	0	n	n	n	n	Horizontal Tab
0	1	1	n	n	n	n	Line Feed (0110000 indicates carrier return)
1	0	0	n	n	n	n	New Line
1	0	1					Reserved
1	1	0					Reserved
1	1	1					Reserved

Figure 2-35. The End Control Byte

### **Part 3. Logic of the Form Description Utility**

## Contents

<b>Section 1: Introduction</b> . . . . .	3-3	OS Method of Operation . . . . .	3-14
FD Utility Purpose and Function . . . . .	3-3	DOS Method of Operation . . . . .	3-17
System Requirements . . . . .	3-3	Linkage Editor Step Functions . . . . .	3-17
Physical Characteristics of the FD Utility . . . . .	3-3	OS Method of Operation . . . . .	3-17
Characteristics for OS Systems . . . . .	3-3	DOS Method of Operation . . . . .	3-21
Characteristics for DOS Systems . . . . .	3-3	Storage Step Functions . . . . .	3-21
Operational Considerations . . . . .	3-4	OS Method of Operation . . . . .	3-21
Input and Output of the FD Utility . . . . .	3-4	DOS Method of Operation . . . . .	3-21
OS Control Information . . . . .	3-4	<b>Section 3: Program Organization</b> . . . . .	3-35
DOS Control Information . . . . .	3-5	FD Utility Organization . . . . .	3-35
<b>Section 2: Method of Operation</b> . . . . .	3-7	Control Step Organization . . . . .	3-35
FD Utility General Operation and Data Descriptions . . . . .	3-7	IDFCT Organization . . . . .	3-35
General Logical Flow . . . . .	3-7	IJLFCT Organization . . . . .	3-40
Input and Output Data Descriptions . . . . .	3-7	Linkage Editor Step Organization . . . . .	3-43
Input Data . . . . .	3-7	Storage Step Organization . . . . .	3-44
Output Data . . . . .	3-10	IDFST Organization . . . . .	3-44
Processing Data Description and Flow . . . . .	3-10	IJLFST Organization . . . . .	3-47
Input Object Module Data . . . . .	3-10	IJLFLOAD Organization . . . . .	3-49
Object Modules IDFST, IJLFST, IJLFLOAD, . . . . .	3-12	IJLFUPDT Organization . . . . .	3-51
and IJLFUPDT . . . . .	3-12	Message Modules IDFM01 and IJLFM01 Organization . . . . .	3-54
IDFST . . . . .	3-12	FD Utility Flowcharts . . . . .	3-54
IJLFST, IJLFLOAD, IJLFUPDT . . . . .	3-13	<b>Section 4: Directory</b> . . . . .	3-76
Overlay Segments of the Storage Step . . . . .	3-13	<b>Section 5: Data Area Layouts</b> . . . . .	3-77
FD Utility Functions . . . . .	3-14	<b>Section 6: Diagnostic Aids</b> . . . . .	3-78
Control Step Functions . . . . .	3-14		

## Illustrations

<i>Figure</i>	<i>Title</i>	<i>Page</i>	<i>Figure</i>	<i>Title</i>	<i>Page</i>
3-1	OS FD Utility Logical Flow . . . . .	3-8	3-17	OS FD Utility Auxiliary Storage Residency . . . . .	3-36
3-2	DOS FD Utility Logical Flow . . . . .	3-9	3-18	DOS FD Utility Auxiliary Storage Residency . . . . .	3-36
3-3	Card Image Input Format Descriptions . . . . .	3-10	3-19	Module IDFCT Physical Organization . . . . .	3-37
3-4	Input Module Physical Organization . . . . .	3-11	3-20	Module IDFCT Hierarchy of Routines . . . . .	3-38
3-5	OS Input CSECT Example . . . . .	3-11	3-21	Module IJLFCT Physical Organization . . . . .	3-41
3-6	DOS Input CSECT Example . . . . .	3-12	3-22	Module IJLFCT Hierarchy of Routines . . . . .	3-42
3-7	KUPB Format . . . . .	3-12	3-23	Module IDFST Physical Organization . . . . .	3-45
3-8	Typical Input FD Program . . . . .	3-13	3-24	Module IDFST Hierarchy of Routines . . . . .	3-46
3-9	OS FD Utility Data Flow . . . . .	3-15	3-25	OS REPLTAB Format . . . . .	3-47
3-10	DOS FD Utility Data Flow . . . . .	3-16	3-26	Module IJLFST Physical Organization . . . . .	3-48
3-11	OS Control Step Method of Operation (3 Parts) . . . . .	3-18	3-27	Module IJLFST Hierarchy of Routines . . . . .	3-48
3-12	DOS Control Step Method of Operation (3 Parts) . . . . .	3-22	3-28	Module IJLFLOAD Physical Organization . . . . .	3-50
3-13	SEGTAB Contents . . . . .	3-25	3-29	Module IJLFLOAD Hierarchy of Routines . . . . .	3-50
3-14	ENTAB Contents . . . . .	3-25	3-30	Module IJLFUPDT Physical Organization . . . . .	3-52
3-15	OS Storage Step Method of Operation (3 Parts) . . . . .	3-26	3-31	Module IJLFUPDT Hierarchy of Routines . . . . .	3-53
3-16	DOS Storage Step Method of Operation (5 Parts) . . . . .	3-30	3-32	Modules IDFM01 and IJLFM01 Organization . . . . .	3-54

The Form Description (FD) utility is a service program that prepares unpacked Form Description programs for transmission to one or more IBM 3735 Programmable Buffered Terminals. The program may be executed under either the Operating System (OS) or the Disk Operating System (DOS).

### FD UTILITY PURPOSE AND FUNCTION

The FD utility is the IBM support program that prepares the Form Description unpacked programs for transmission to the 3735 terminals. The utility operates basically the same under OS and DOS. It reads the output of the Form Description macro assembly from a card reader or an equivalent sequential input device. It checks the input programs' integrity and arranges the programs into 476-byte blocks. The utility then writes the arranged program blocks into a user-specified data set that is later made available to a user-written application program.

The aims of the FD utility are accomplished through the logical organization of the program in three sequential job steps: Control step, Linkage Editor step, and Storage step.

The Control step verifies the correctness of the object module input (except that data generated in columns 17-72 of the TXT card images) and generates the necessary Linkage Editor control statements for the next step.

The Linkage Editor step performs the linkage editing of the input object module, under control of the control statements generated in the first step. The Linkage Editor step also includes the object module IDFST (under OS) or IJLFST (under DOS) in the input. This module then becomes the root segment of the overlay program executed in the Storage step.

The third and last step, the Storage step, is the execution of the linkage edited overlay program. The data originally generated by the macro assembly is brought into main storage as overlay segments and put out to the user's private data set in 476-byte blocks. Additionally, an optional JCL parameter (OS) or control cards (DOS) may be used to allow the storage of Form Description programs whose names are the same as those of programs already existing in the user's data set.

### SYSTEM REQUIREMENTS

The main storage required by the FD utility is that required for the minimum OS or DOS Linkage Editor available to the system. The main storage required by the utility, exclusive of the Linkage Editor step, is no more than 10K bytes (OS) or 12K bytes (DOS). The utility requires the presence of three I/O devices: a card reader or equivalent device, a

printer, and a direct access storage device. The utility uses no more than 10 tracks of 2311 storage, or the equivalent, for program residence. The user's auxiliary storage requirements depend on the number and size of the forms being described.

To include the FD utility in his operating system, the user must copy it from the component library on which it is distributed. This function may require the allocation of additional space. Before the FD utility is executed, the system programmer must ensure that suitable input and output data sets are designated through correct coding of the job control statements.

### PHYSICAL CHARACTERISTICS OF THE FD UTILITY

#### Characteristics for OS Systems

The FD utility program contains three modules: IDFCT, IDFST, and IDFM01. These modules may reside in either a private (user) job/step library or the system link library (SYS1.LINKLIB). The module IDFCT is executed in the first, or Control, step. The module IDFST is executed in the third, or Storage, step. However, it must be linkage edited with the output of the first step before it can be executed as the GO module. The linkage editing is performed during the second, or Linkage Editor, step of the utility. The linkage editing is necessary because IDFST is the root segment of an overlay program whose overlay segments are available only at the end of the first step. The message module, IDFM01, contains the messages that are written out during the execution of the first and third steps. These messages may be written to either the system printer or a system console.

The module that is loaded and executed in the third step, that is, IDFST and the linkage edited overlay segments, is an overlay program. Its function is to write the data that is contained in the overlay segments into the user's partitioned data set. After the module has been executed, it serves no further purpose. Therefore, it should be regarded as a temporary data set and should be deleted at the end of the third step.

#### Characteristics for DOS Systems

The FD utility program contains five relocatable modules: IJLFCT, IJLFST, IJLFLOAD, IJLFUPDT, and IJLFM01. IJLFCT and IJLFM01 reside in a core image library; IJLFST, IJLFLOAD, and IJLFUPDT may reside in either a private (user) relocatable library or the system relocatable

library. The module IJLFCT, which is executed in the first, or Control, step, must be linkage edited during system generation. The modules IJLFST, IJLFLOAD, and IJLFUPDT are executed in the third, or Storage, step. However, they must be linkage edited with the output of the first step before they can be executed. This linkage editing is performed during the second, or Linkage Editor, step of the utility. The linkage editing is necessary because IJLFST is the root phase of an overlay program whose overlay phases are available only at the end of the first step; IJLFST loads either IJLFLOAD or IJLFUPDT to process these overlay phases. The message module, IJLFM01, contains the messages that are written out to the system printer during the execution of the first and third steps. IJLFM01 contains no relocatable code; it must be linkage edited during system generation to allow its subsequent loading by IJLFCT, IJLFLOAD, and IJLFUPDT.

The modules that are loaded and executed in the third step, that is, IJLFST, IJLFLOAD, IJLFUPDT, and the linkage edited overlay segments, comprise an overlay program. The function of this overlay program is to write the data that is contained in its overlay phases into the user's indexed sequential data set. Since there are no external inputs to this module, it is a temporary data set and is effectively deleted at the end of the third step. The Linkage Editor step places this temporary data set in the unused core image library space, but does not catalog it. The user must ensure that this library space is adequate.

## OPERATIONAL CONSIDERATIONS

The FD utility is scheduled through the job input stream and encompasses three sequential job steps, as previously discussed. There is one optional feature: in OS, the REPLACE parameter; in DOS, RPLACE control statements. These options are discussed fully in Part 3, Section 2, "Method of Operation." The FD utility program creates one or more executable Form Description unpacked programs (FD programs) from the one or more object modules that are produced by the assembly of the Form Description macros (FD macros). The utility also places the FD programs in a user-defined partitioned data set (under OS) or indexed sequential data set (under DOS), which serves the user as a library of FD programs. The use of the FD utility involves the Linkage Editor, since the object modules produced by the FD macros contain backward references resulting from ORG statements.

If the utility encounters an uncorrectable error, the action it takes depends upon which step of the three step sequence is being executed. The action may be either the immediate termination of the program or, if the Storage step is executing, the deletion of a partially created sequence of FD programs from the user's data set, then termination. In both cases, however, a message explaining

the action and its cause is written to either the system printer or the system console. The utility produces a diagnostic listing at the end of each program step. The listing produced by the last step includes the name of every FD program that was added to, deleted from, or not added to the user's data set.

The utility's response to environmental errors arising from improper input, such as a card missing or out of sequence, is to write a message to the Control step diagnostic listing, print the listing, then terminate. Any input following the erroneous card is not processed. The response to job control errors, such as the insufficient allocation of space in a data set, is to terminate processing and to note the error condition in the diagnostic listing. The response to implementation errors, such as errors in system control blocks, is those error recovery and termination options provided by OS and DOS.

## Input and Output of the FD Utility

The object modules produced by the FD macros are the principal input to the FD utility program. These modules were designed to make the input as self-describing as possible. That is, the records themselves contain all the information needed for their processing in the three steps of the FD utility. The physical organization of these object modules is 80-byte card images. Two levels of logical organization exist within the physical organization of the object modules. The primary level of logical organization is by control section (CSECT); the secondary level is by keyed Form Description unpacked program block (KUPB). Each CSECT consists of from one to six KUPBs (under OS) or from one to three KUPBs (under DOS), plus an end-of-assembly indicator that indicates the presence or absence of additional CSECTs. Each KUPB is a 486-byte record which is comprised of four subfields: name, count, data, and end-of-form. The physical descriptions of CSECTs and KUPBs are contained in Part 3, Section 2, "Method of Operation."

The principal output data set of the FD utility is the user's library of FD programs. This library is a partitioned data set (under OS) or an indexed sequential data set (under DOS) whose members are the individual FD programs. These members are known by either the name used in the name field of the FDFORM macro or by the temporary name that the FD Utility assigns if FD program replacement is not specified in the user's JCL. The output data set is described in more detail in the section entitled "Method of Operation."

## OS Control Information

Execution of the FD utility program is sequential: the Control step is executed first; the Linkage Editor step, second; and the GO module (Storage step), third and last. The control is mediated through the OS Job, Task, and Data

Management facilities. In particular, the Job Control facilities allow the passing of intermediate data sets between job steps. Therefore, correctly coded job control statements are imperative to the correct functioning of the FD utility program.

Because the control is mediated by OS Job Management, the FD utility may be executed one step at a time, or even as steps following an assembly of the FD macros. Since each step in the program sets a nonzero return code if it detects an error condition, cataloged procedures that include appropriate COND keyword guards may be designed to suit the user's program integrity requirements.

The data flow among the program units includes a variety of data types:

- The input object module data
- The object module IDFST
- The overlay segments of the GO module

The physical descriptions of these data types and a discussion of the processing performed upon them by the FD utility are contained in Part 3, Section 2, "Method of Operation."

#### **DOS Control Information**

Execution of the FD utility program is sequential: the Control step is executed first; the Linkage Editor step, second; and the Storage step, third and last. The control is mediated

through the DOS Job Control and Data Management facilities. In particular, the Job Control facilities allow the passing of intermediate data sets by planned changes of device assignments. For example, SYSPCH, an output data set of the Control step, becomes (through Job Control intervention) SYSLNK, an input data set to the Linkage Editor step. Therefore, correctly coded job control statements are imperative to the correct functioning of the FD utility.

Because the control is mediated by the DOS Job Control facility, the FD utility may be executed one step at a time (one step per job). However, it may also be executed as sequential job steps following an assembly of the FD macros. In this case, since each step in the program would be executed unconditionally, even if errors were encountered in previous steps, the user should insert a PAUSE statement after each step. He should then allow the next step to proceed only after he validates the correctness of the previous step. Furthermore, to avoid processing delays, the user should perform the FD macro assembly separately from the execution of the FD utility.

The data flow among the program units includes a variety of data types:

- The input object module data
- The object modules IJLFST, IJLFLOAD, and IJLFUPDT
- The overlay segments of the Storage step

The physical descriptions of these data types and a discussion of the processing performed upon them by the FD utility are contained in Part 3, Section 2, "Method of Operation."





**FD UTILITY GENERAL OPERATION AND DATA DESCRIPTIONS**

**General Logical Flow**

The assembly of the FD macros produces one or more card-image object modules that contain FD programs. By themselves, these modules are not executable in a 3735 terminal; they must be (1) validated, (2) linkage edited, (3) arranged into a usable format, and (4) placed in a user-defined FD program library. The purpose of the FD utility is to perform these four major tasks. To accomplish this purpose, the utility is divided into three sequential job steps: Control, Linkage Editor, and Storage.

The Control step, which is executed first, performs the following functions:

- Validates the input data
- Creates Linkage Editor control statements
- Writes the validated input data and control statements to a utility data set
- Writes a diagnostic listing of the Control step results to an output data set

Next, the Linkage Editor step, which is the second job step executed, performs four more functions:

- Creates the overlay program that is executed in the Storage step
- Resolves backward origins and, under OS, resolves a V-type address constant
- Writes the newly created overlay program to a temporary library
- Writes a diagnostic listing of the Linkage Editor step results to an output data set

Finally, the Storage step, the third job step to be executed, performs the following functions:

- Loads the overlay segments from the temporary library where the Linkage Editor step placed them
- Creates new or replacement members (FD programs) and stores them in the user's FD program library
- Writes a diagnostic listing of the Storage step results to an output data set

Of the four major tasks mentioned previously, the Control step accomplishes the first one; the Linkage Editor step, the second; and the Storage step, the last two. Figures 3-1 and

3-2 represent the logical flow of the FD utility operating under OS and DOS, respectively.

**Input and Output Data Descriptions**

*Input Data*

The object modules produced by the FD macros are the principal input to the FD utility program. These modules were designed to make the input as self-describing as possible. That is, the records themselves contain all the information needed for their processing in the three steps of the FD Utility. The physical organization of these object modules is as 80-byte card images. Each module contains three record types: ESD, TXT, and END. Figure 3-3 describes the format of each type. Figure 3-4 is a graphic representation of a module's physical organization.

Within the physical organization of the object modules, there are two levels of logical organization. The primary level of logical organization is by control section (CSECT); the secondary level is by keyed Form Description unpacked program block (KUPB).

Under OS, each CSECT is  $(6 \times 486) + 4 = 2920$  bytes long, except the last one. The last CSECT is  $(n \times 486) + 4$  bytes long, where  $n$  is an integer with a value of 1, 2, 3, 4, 5, or 6. Under DOS, each CSECT is  $(3 \times 486) + 6 = 1464$  bytes long, except the last one. The last CSECT is  $(n \times 486) + 6$  bytes long, where  $n$  is an integer with a value of 1, 2, or 3. This means that a CSECT consists of from one to six or one to three KUPBs plus a four-byte or six-byte end-of-assembly indicator under OS and DOS, respectively. The end-of-assembly indicator is used to indicate the presence or absence of additional CSECTs. The indicator is set to all zeros for all but the last CSECT, for which it is set to all ones. In addition to the length restrictions of CSECTs, there are also name restrictions. Under OS, the CSECT names are in the sequence IDF1000, IDF1001, IDF1002, etc. Under DOS, the CSECT names are in the sequence IJLF1000, IJLF1001, IJLF1002, etc. Figures 3-5 and 3-6 illustrate the composition of OS and DOS CSECTs.

The secondary level of logical organization within the input object modules is by KUPB. Each KUPB consists of a 10-byte key field and a 476-byte unpacked program block. Within the key field are two subfields: (1) an eight-byte name subfield and (2) a two-byte count subfield. Within the unpacked program block field are two more subfields: (3) a 470-byte data subfield and (4) a six-byte end-of-form subfield.

Figure 3-1. OS FD Utility Logical Flow

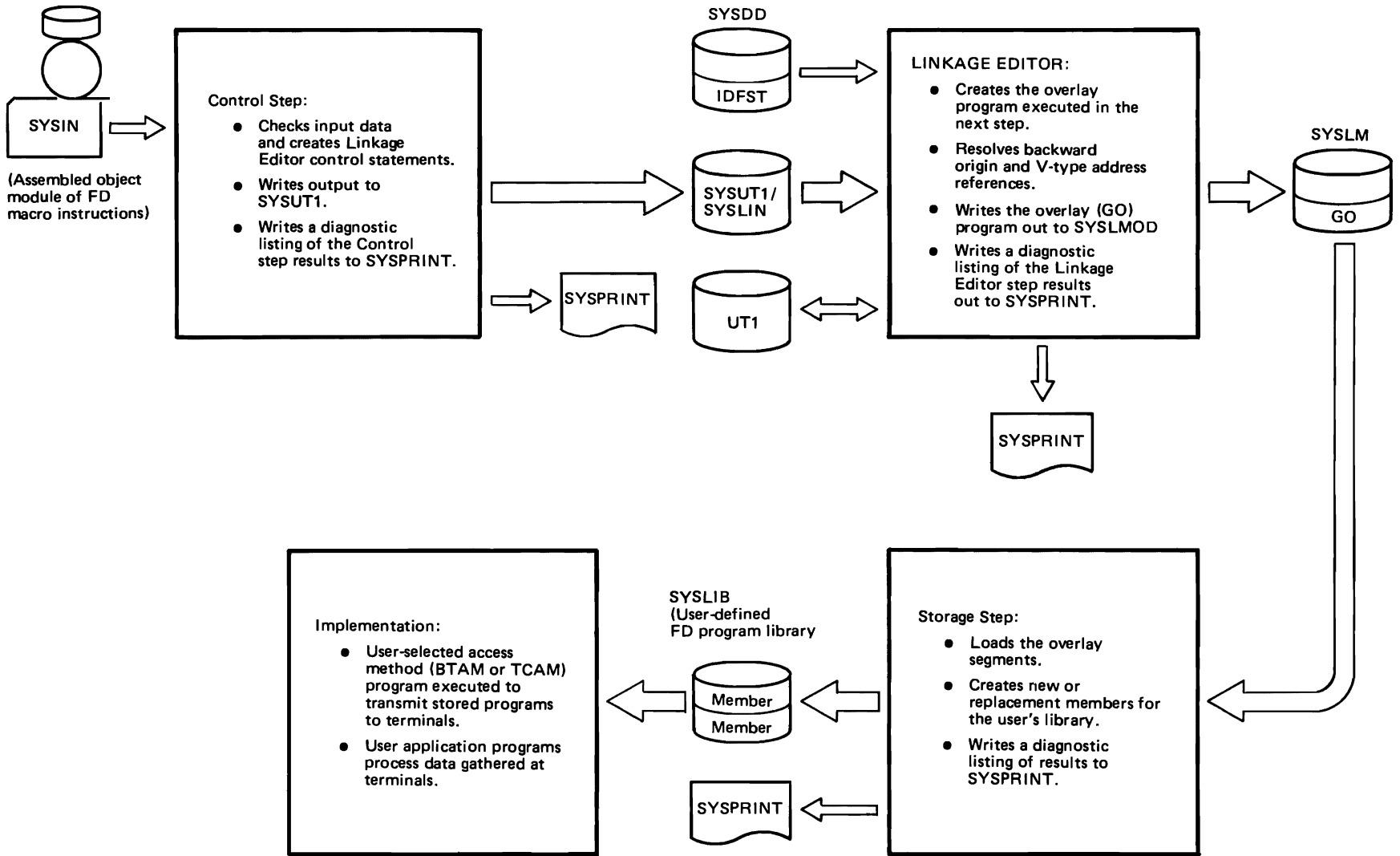
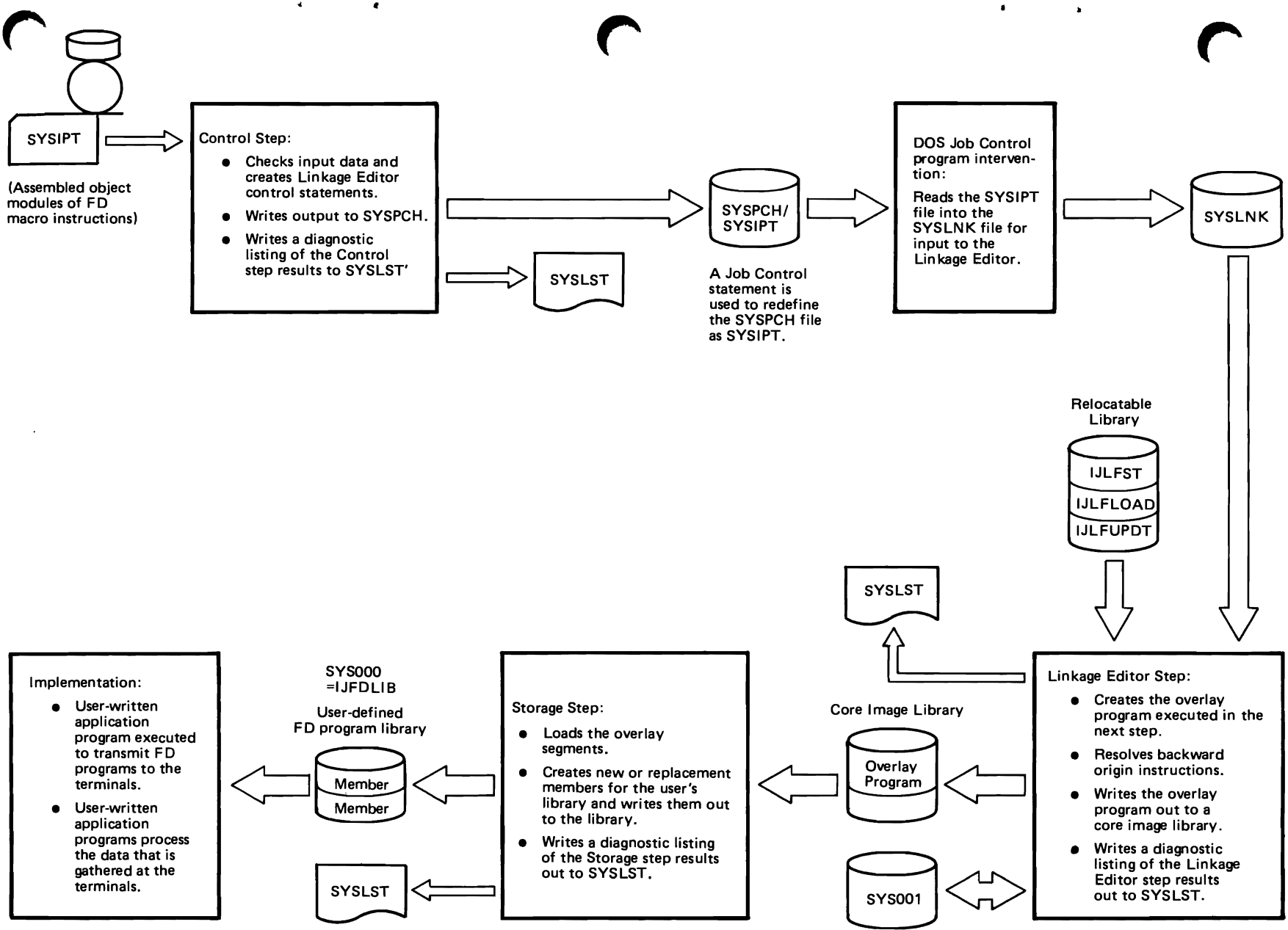


Figure 3-2. DOS FD Utility Logical Flow



CARD TYPE	COLUMNS	CONTENTS
ESD	1	12-2-9 punch
	2-4	Card type, which should be ESD
	5-10	Blank
	11-12	Variable field count. The number of bytes of information in the variable field (columns 17-64)
	13-14	Blank
	15-16	ESDID of the first CSECT definition in the variable field
	17-64	Variable field. From one to three ESD items, each in the following format: 8 bytes - CSECT name, left justified and right padded with blanks, as needed 1 byte - ESD type code, which should be a hexadecimal 00 3 bytes - CSECT address 1 byte - Blank 3 bytes - CSECT length
	65-72	Blank
	73-76	Deck ID
	77-80	Card sequence number
TXT	1	12-2-9 punch
	2-4	Card type, which should be TXT
	5	Blank
	6-8	Relative address of the first instruction on the card
	9-10	Blank
	11-12	Byte count. The number of bytes in the information field (columns 17-72)
	13-14	Blank
	15-16	ESDID
	17-72	56-byte information field
	73-76	Deck ID
77-80	Card sequence number	
END	1	12-2-9 punch
	2-4	Card type, which should be END
	5-39	Blank
	40-62	Version of the assembler and the date and time of the assembly } optional
	63-72	Blank
	73-76	Deck ID
	77-80	Card sequence number

Figure 3-3. Card Image Input Format Descriptions

The contents of the four subfields are as follows:

1. The name subfield contains the form name, left justified and padded to the right with blanks, as needed. This is the name by which the FD program will be known in the user's FD program library.
2. The count subfield commences with a binary zero and is incremented by one in each KUPB within a single FD program. However, if there was an error in the assembly of the program, the first subfield contains a binary -1. If the assembled program is incomplete, this subfield contains a binary -2.

3. The data subfield contains the actual instructions that are sent to the 3735 to control the terminal's actions during forms creation and data capture.
4. The data in the six-byte end-of-form subfield is also sent to the 3735. This subfield is set to all zeros in every KUPB except the last one in an FD program. In the last KUPB, it is set to all ones to indicate the end of the FD program.

Figure 3-7 illustrates the format of a KUPB. Figure 3-8 shows a complete FD program as it would appear in the input data set.

#### Output Data

The principal output data set of the FD utility is the user's library of FD programs. This library is a partitioned data set (under OS) or an indexed sequential data set (under DOS) whose members are the individual FD programs. The members of this library may be stored under either of the following name types:

- The name used in the name field of the FDFORM macro, that is, the high-order eight bytes of the key field in all the KUPBs of a particular program.
- The temporary name assigned by the utility if FD program replacement is not specified in the user's JCL or control cards.

Temporary names are chosen from the lowest available name in the sequence IDFTempo through IDFTemp9 (under OS) or IJLFtm00 through IJLFtm09 (under DOS). In any case, the FD program names are listed in the diagnostic listing produced in the Storage step. Any storage of a program under a temporary name is noted in this listing. Standard system facilities may be used later to rename or delete the programs stored under temporary names.

#### Processing Data Description and Flow

Data flow among the program units includes a variety of data types:

- The input object module data
- The object module IDfst (OS) or IJLFST, IJLFLOAD, and IJLFUPDT (DOS)
- The overlay segments of the Storage step

The following paragraphs describe the changes these data types undergo during the execution of the FD utility.

#### Input Object Module Data

The input object module data undergoes changes during processing until its incorporation in the executable module of the Storage step. Initially, the data consists of 80-byte card images representing an object module deck. These card images are in the form of ESD, TXT, and END records. The Control step reads the individual card images into and out of main storage during validation, generates Linkage

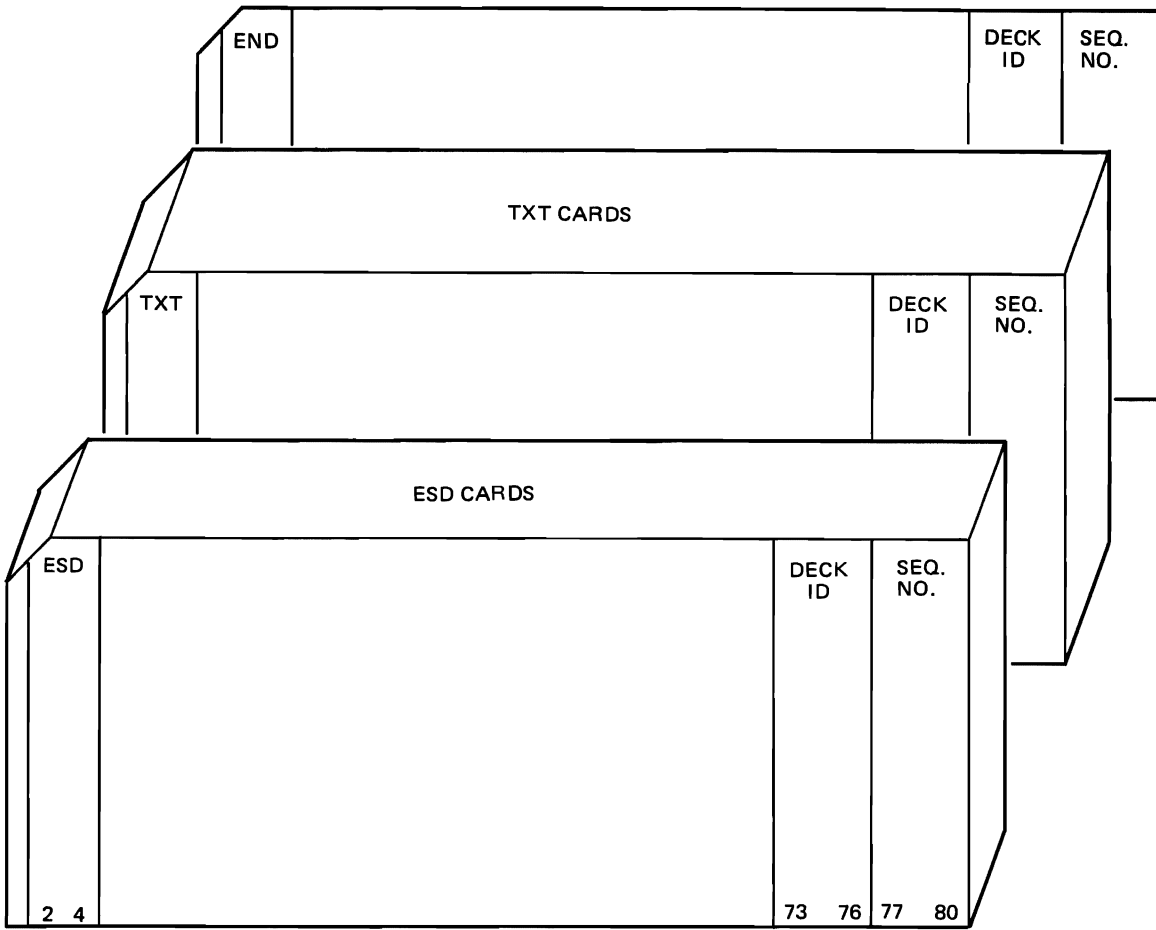
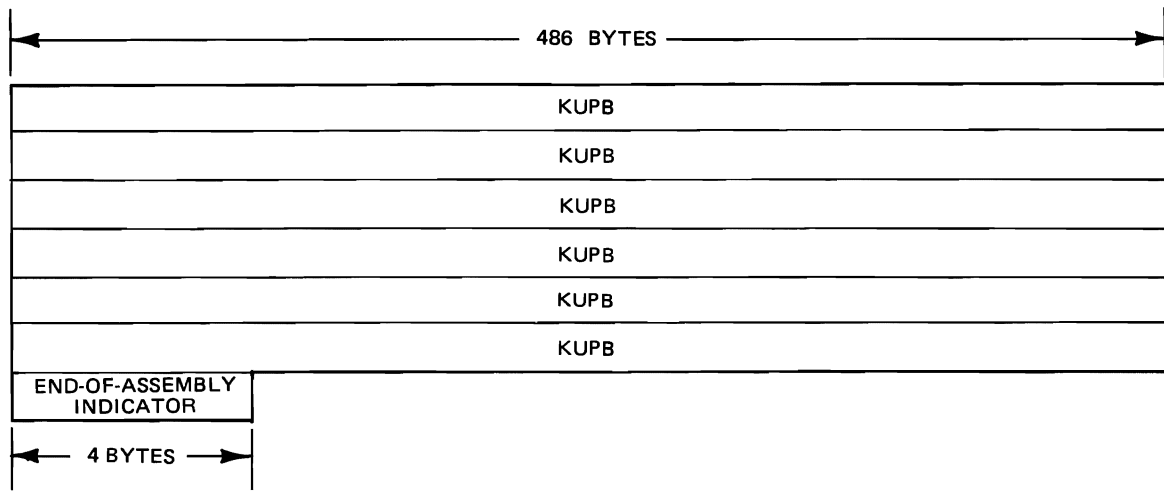
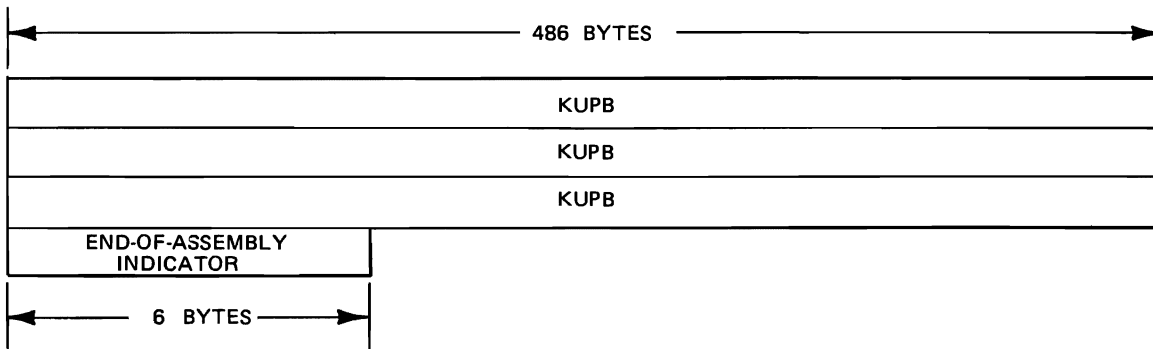


Figure 3-4. Input Module Physical Organization



NOTE: The last input CSECT may have six or less KUPBs and the four-byte end-of-assembly indicator is set to all ones.

Figure 3-5. OS Input CSECT Example



NOTE: The last input CSECT may have three or less KUPBs and the six-byte end-of-assembly indicator is set to all ones.

Figure 3-6. DOS Input CSECT Example

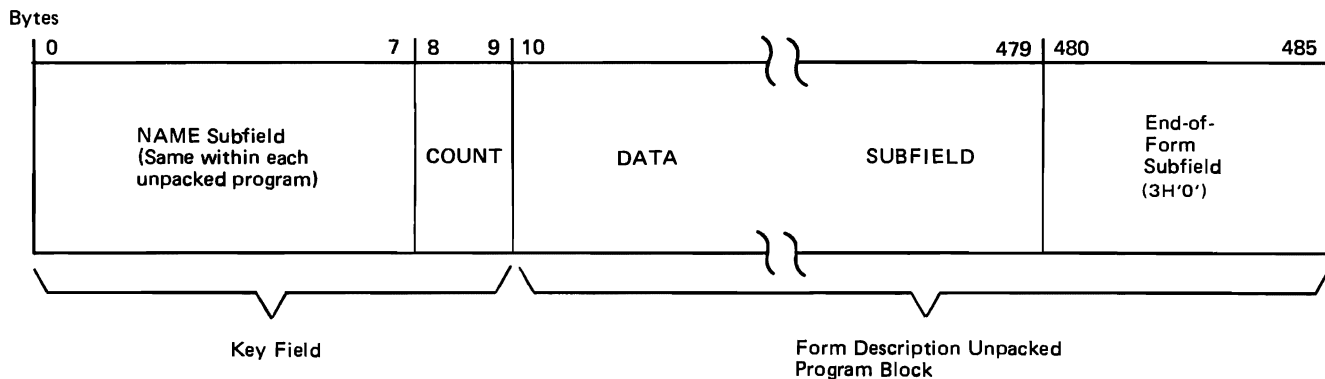


Figure 3-7. KUPB Format

Editor control statements, and writes the card images and control statements to a utility file in auxiliary storage. The Linkage Editor, using the control statements generated in the Control step, transforms the card images into segments of an overlay program; each CSECT becomes one overlay segment. In the process, the card image input data becomes Linkage Editor output records. The ESD, deck identification, and sequence information that was validated in the Control step is removed during linkage editing. The TXT information, that is, the data in columns 17-72 of the TXT cards, remains unchanged, except that backward origins are resolved. The result of this processing is that the KUPBs retain the structure shown in Figure 3-7, but they become parts of overlay segments that may be loaded into main storage. Under OS, the overlay segments possess segment numbers 2, 3, 4 . . . n; the root segment (IDFST) is numbered 1. Under DOS, the segments' phase names are the same as the input CSECTs' names. Besides KUPBs, the only other data in each overlay segment is the four-byte or six-byte end-of-assembly indicator.

*Object Modules IDFST, IJLFST, IJLFLOAD, and IJLFUPDT*

*IDFST*: As previously stated, the object module *IDFST*, which is used in the OS version of the FD utility, is not executable until the Linkage Editor step has successfully finished executing. The Linkage Editor creates two tables for the module and resolves one V-type address constant contained in it. This V-type constant is a pointer to the load address of the first CSECT generated by the FD macros, that is, *IDF1000*. The two tables created are the segment table, *SEGTAB*, and the entry table, *ENTAB*. The Linkage Editor places the segment table at the start of *IDFST* and the entry table at its end, thus producing an executable load module. The load module *IDFST* and the linkage-edited CSECTs of the original input data, each of which is an individual segment of an overlay program, are written into a temporary load module library as a single overlay program. This program, which still bears

*=V(10F1000)*  
*to = 01004 to*  
*... + ...*

NAME	0	DATA	3H'0'
NAME	1	DATA	3H'0'
NAME	2	DATA	3H'0'
NAME	3	DATA	3H'0'
NAME	4	DATA	3H'0'
NAME	5	DATA	3H'0'
NAME	6	DATA	3H'0'
NAME	7	DATA	3H'0'
NAME	8	DATA	3H'0'
NAME	n-1	DATA	3H'0'
NAME	n	DATA	3H'1'

Figure 3-8. Typical Input FD Program

the name IDFST, is executed in the Storage step as the GO module.

For more detailed information about the resolution of the V-type address constant and the creation of the segment and entry tables, refer to the subsection of this section entitled "Linkage Editor Step Functions."

**IJLFST, IJLFLOAD, and IJLFUPDT:** As previously stated, the object modules IJLFST, IJLFLOAD, and IJLFUPDT, which are used in the DOS version of the FD utility, are not executable until the Linkage Editor step has successfully finished executing. The Linkage Editor creates the phases IJLFST, IJLFLOAD, and IJLFUPDT from the modules. The phases IJLFST, IJLFLOAD and IJLFUPDT and the previously linkage edited CSECTs of the input data, each of which is an individual phase of an overlay program, are written into auxiliary storage as a single overlay program. This program is executed in the Storage step.

#### Overlay Segments of the Storage Step

Each overlay segment of the Storage step consists of KUPBs and the end-of-assembly indicator. The segments are called into main storage by the issuing of the SEGWT supervisor

call (OS) or the LOAD macro (DOS). Each segment is brought into main storage when the processing of the previous segment is complete, provided that the end-of-assembly indicator shows that another valid segment is available. The segment loading occurs during the larger process of writing the FD programs to the user's library, but is independent from it.

The KUPBs within each segment are examined serially. The data portions of the KUPBs, that is, bytes 10-485, are successively written to the user's library until the end-of-form subfield indicates that the last KUPB in the program has been encountered. Under OS, the next Storage step function is to issue a STOW macro; under DOS, this action is not necessary. If the return from the STOW indicates that the FD program is the duplicate of an FD program that already exists in the user's library, the utility may construct and consult a replacement table. (Under DOS, this replacement table is always constructed before the program attempts to process any KUPBs.) The purpose of this consultation is to determine whether the new FD program should replace the older program in the user's library. If the program name is not found in the replacement table or if the user did not employ the replacement parameter, the

utility attempts to store the new FD program under a temporary name. The Storage step resumes processing with the next KUPB, if one is available. Otherwise, it examines the end-of-assembly indicator to determine whether there is another valid segment to be processed.

The flow of data during the execution of the FD utility is shown in Figure 3-9 for OS and Figure 3-10 for DOS.

## FD UTILITY FUNCTIONS

### Control Step Functions

The primary tasks of the Control step are (1) the validation of the input records produced by the FD macros and (2) the generation of Linkage Editor control statements and the concatenation of these statements with the input records. To accomplish these tasks, the Control step performs several interrelated functions during its execution. These functions are described in the following paragraphs.

The first function of the Control step is the opening of the system input and output files and the utility file that receives the validated input records. Under OS, the program then determines whether these three files were opened successfully. If the file openings were successful, the program proceeds to the validation of the input records. Under DOS, this checking of the file openings is performed by the operating system.

There are three input record types: ESD, TXT, and END. Although the functions of these records are different, as are the fields within each one, two fields are common to all three types. These fields are (1) deck ID, columns 73-76, and (2) card sequence number, columns 77-80. The Control step validates these fields in all three record types; for TXT and END records, these are the only fields validated. In addition to validating these fields, the program also ensures that the record types are in the proper order. That is, an ESD record must be followed by another ESD record or TXT record; a TXT record must be followed by another TXT record or an END record; an END record must be followed by an ESD record (indicating that another FD program module follows) or an end-of-file indication. Other than the restrictions imposed by the system assembler, there are no limitations on the number of ESD and TXT records allowed in a single FD program module. However, only one END record per module is allowed.

As previously stated only the deck ID and card sequence number fields of the TXT and END records are validated. On the other hand, the Control step validates the ESD records more rigorously. In any ESD record, there are from one to three ESD items specified in the 'variable' field, columns 17-64; only the last ESD record within a module may contain less than three ESD items. The fields that the Control step validates in each ESD item are as follows:

- Name

The eight-byte name of the external symbol, which is the name of a CSECT generated by the FD macros. For the

first ESD item of an input module, this field must contain IDF1000 (OS) or IJLF1000 (DOS). The value of each name field in succeeding ESD items must be one greater than the value of the name field in the immediately preceding ESD item. If multiple input modules are present, the Control step renames the CSECTs of these modules so that the sequential progression is maintained in the output data set.

- Type

The one-byte ESD type code, which should be a hexadecimal 00.

- Address

The three-byte address of the external symbol (CSECT) within the input module. For the first ESD item of an input module, this field must contain 0. The value of each address field in succeeding ESD items must be a progressive multiple of 2920 (OS) or 1464 (DOS).

- Length

The three-byte length field of the external symbol (CSECT). Under OS, this value should be 2920 in all ESD items except the last one in a module. The value of the length field in the last ESD item may be 490, 976, 1462, 1948, 2434, or 2920. Under DOS, the value should be 1464 in all ESD items except the last one in a module. The value of the length field in the last ESD item may be 492, 978, or 1464.

The Control step validates these fields in each ESD item. After it validates an ESD item, it uses the 'variable field count' field to determine the presence or absence of additional ESD items. If no more items are available, the program reads in the next record which may be either another ESD record or a TXT record.

The OS and DOS versions of the FD utility operate differently to create the necessary Linkage Editor control statements. These different methods of operation are described in the following paragraphs.

### OS Method of Operation

In validating the input, the OS Control step reads a single card image into main storage, validates that card image, then writes it out to the utility file. At no time during Control step processing are there two or more card images in main storage simultaneously. When all of the records in the entire input file have been validated and written out, the Control step creates and writes out Linkage Editor control statements. The purpose of these control statements is to provide the Linkage Editor, which is called into execution in the next job step, with the information necessary to produce an overlay program. The control statements are written to the same utility file as the validated input records. The first control statement created and written out is an INCLUDE statement. Its purpose is to define the object module that will become the root segment of the overlay program. This module is IDFST. The Control step also



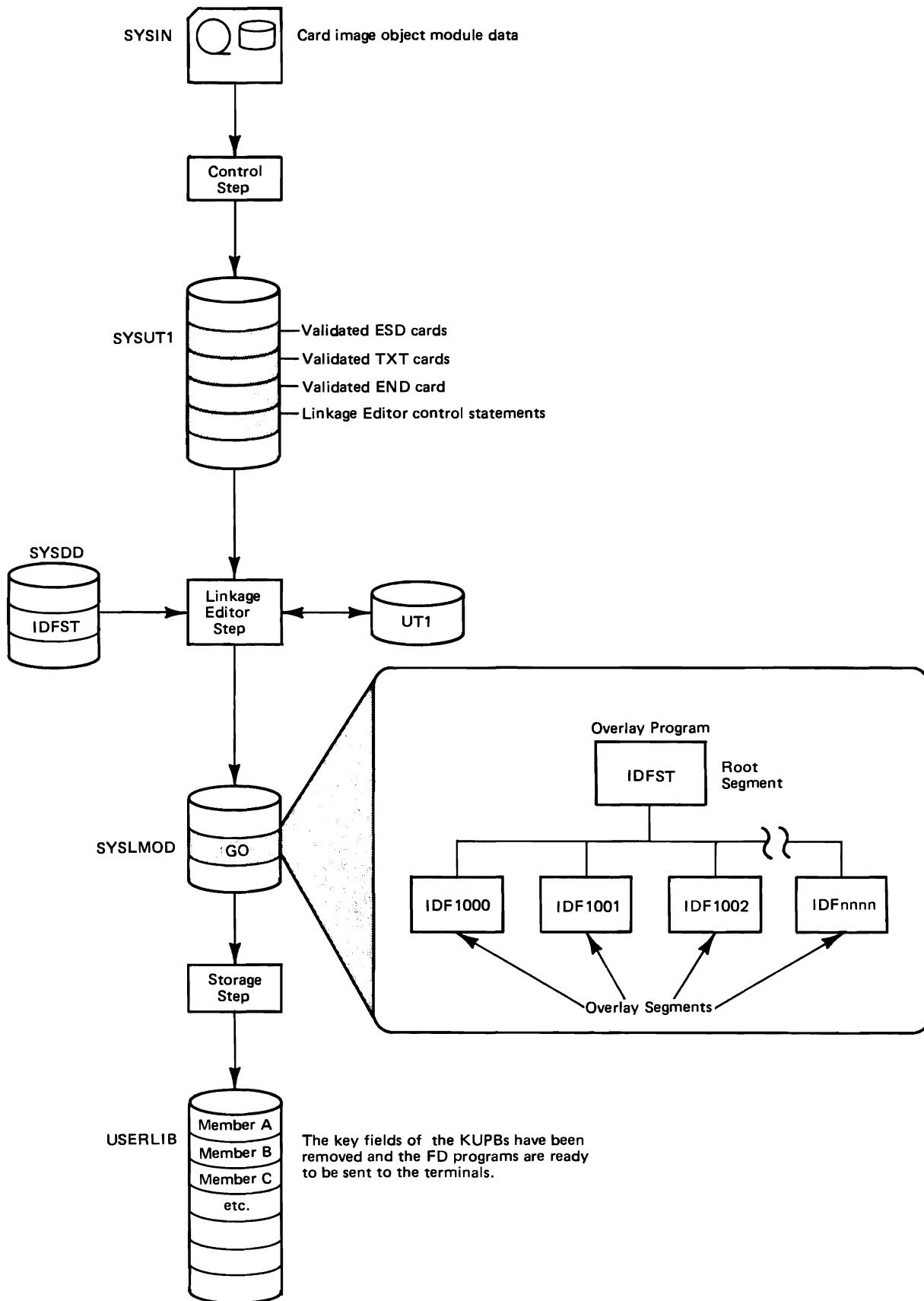


Figure 3-9. OS FD Utility Data Flow

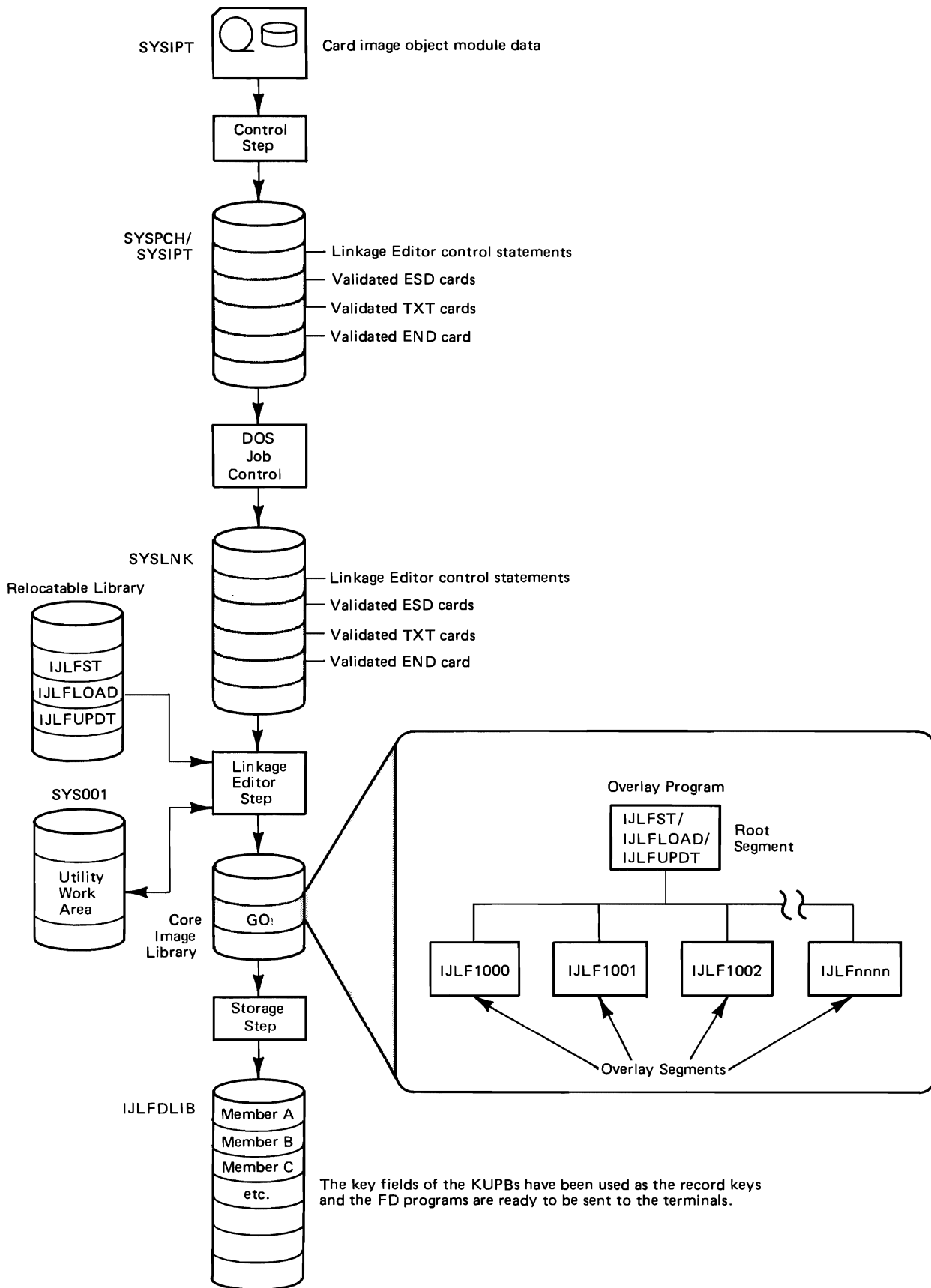


Figure 3-10. DOS FD Utility Data Flow

creates and writes out one OVERLAY statement and one INSERT statement for each ESD item (input CSECT) that was validated. The purpose of these statements is to cause the Linkage Editor to transform each input CSECT into a segment of an overlay program. When all the necessary control statements have been created and written out, the Control step writes a message to the system printer stating that the step has been completed successfully, then terminates.

If a data error is encountered during the execution of the Control step, the step itself writes an error message out to the system printer, then continues processing with the next available module. Figure 3-11 represents the OS Control step's method of operation.

#### *DOS Method of Operation*

In validating the input, the DOS Control step reads a single ESD card image into main storage and validates that card image, but it does not write out the card image. Instead, it continues to read and validate the ESD records until the first TXT card is encountered. At that point, the program creates and writes out the necessary Linkage Editor control statements. Next, it recreates and writes out the ESD records, then reads in and writes out the remaining TXT and END records of the input module. If multiple input modules are present, the entire process is repeated for each module.

The purpose of the control statements is to provide the Linkage Editor, which is called into execution in the next job step, with the information necessary to produce an overlay program. The control statements are written to the same file as the validated input records. The first control statements created and written out are PHASE and INCLUDE statements for the modules IJLFST, IJLFLOAD, and IJLFUPDT, which are the processing phases of the overlay program executed in the Storage step. Next, the program writes out a PHASE and an INCLUDE statement for each input CSECT. The purpose of these statements is to cause the Linkage Editor to transform each input CSECT into an overlay phase. If multiple input modules are present, the phase names inserted in the PHASE statements for the CSECTs of these modules are changed to maintain a sequential progression in the output file.

When all the necessary control statements have been created and written out, the Control step writes a message to the system printer stating that the step has been completed successfully, then terminates. If an error is encountered in the input to the Control step, the step itself writes an error message out to the system printer, then terminates. Figure 3-12 represents the DOS Control step's method of operation.

#### **Linkage Editor Step Functions**

The primary task of the Linkage Editor step is the creation of an executable overlay program from the object module

IDFST (OS) or IJLFST (DOS) and the validated input data. While the methods of operation used to accomplish this task are similar in the OS and DOS versions of the Linkage Editor, enough differences exist to merit a separate discussion of each method. These methods are described in the following paragraphs.

#### *OS Method of Operation*

The first function of the Linkage Editor step is to read the input records from SYSLIN (formerly SYSUT1) and place them in a work file. When all of the object module records have been read, the Linkage Editor encounters the first of the control statements that were generated by the Control step. The first statement, the INCLUDE statement, causes the Linkage Editor to fetch the object module IDFST from either SYS1.LINKLIB or a user's module library and place it in the work file.

The next set of control statements read in, the OVERLAY and INSERT statements, causes the Linkage Editor to perform the following functions:

- Transform the CSECTs, one by one, into segments of an overlay program and assign segment numbers
- Resolve backward origins within the CSECTs
- Construct a segment table at the front and an entry table at the rear of IDFST to produce the root segment of the overlay program executed in the Storage step

The segment table, SEGTAB, consists of a 28-byte heading plus  $n$  four-byte entries, where  $n$  is the number of overlay segments that were created from an equal number of input CSECTs. The purpose of the segment table is to contain information about the relationship of the segments in the overlay program. Also, during the execution of the overlay program, this table indicates which segment are either in main storage or waiting to be loaded. Figure 3-13 illustrates the contents of the segment table.

The entry table, ENTAB, consists of two 12-byte entries. The first entry is created in response to a four-byte V-type address constant that is contained within IDFST. The second entry is the standard last ENTAB entry, which is used for linkage to the Overlay Supervisor. Figure 3-14 illustrates the contents of the entry table. For more information about the generation and composition of the segment and entry tables, refer to the *IBM System/360 Operating System Linkage Editor Program Logic Manual*, order number GY28-6610.

The four-byte V-type address constant mentioned in the preceding paragraph is a pointer to the load address of the first CSECT generated by the FD macros, that is, of IDF1000. It is resolved by setting its low-order three bytes to the address of the entry table. The high-order byte is set to a hexadecimal 02, which is the segment number of the segment containing the single CSECT IDF1000.

When the Linkage Editor has finished transforming all of the input CSECTs into overlay segments, it writes out the

Figure 3-11. OS Control Step Method of Operation (Part 1 of 3)

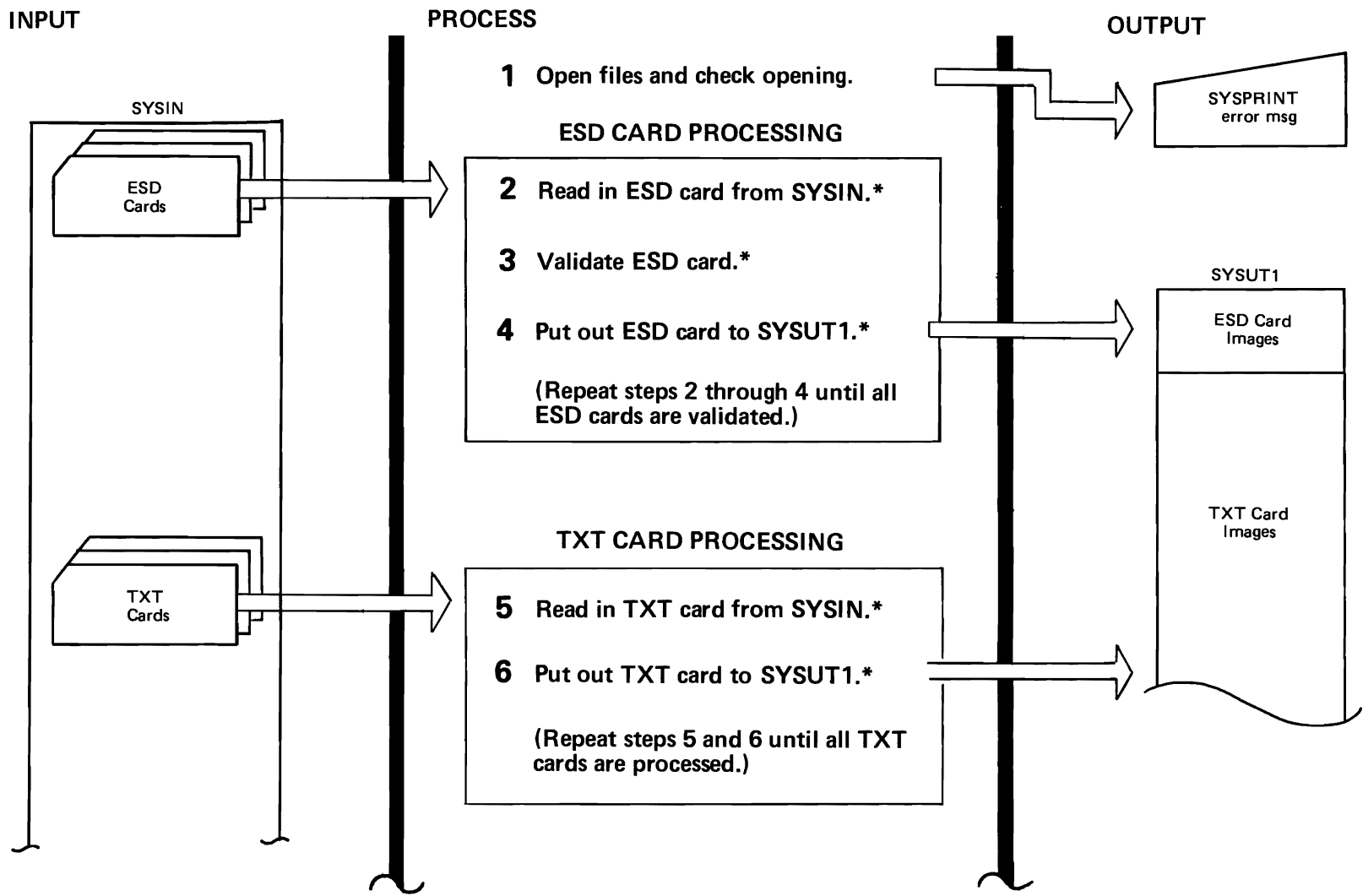
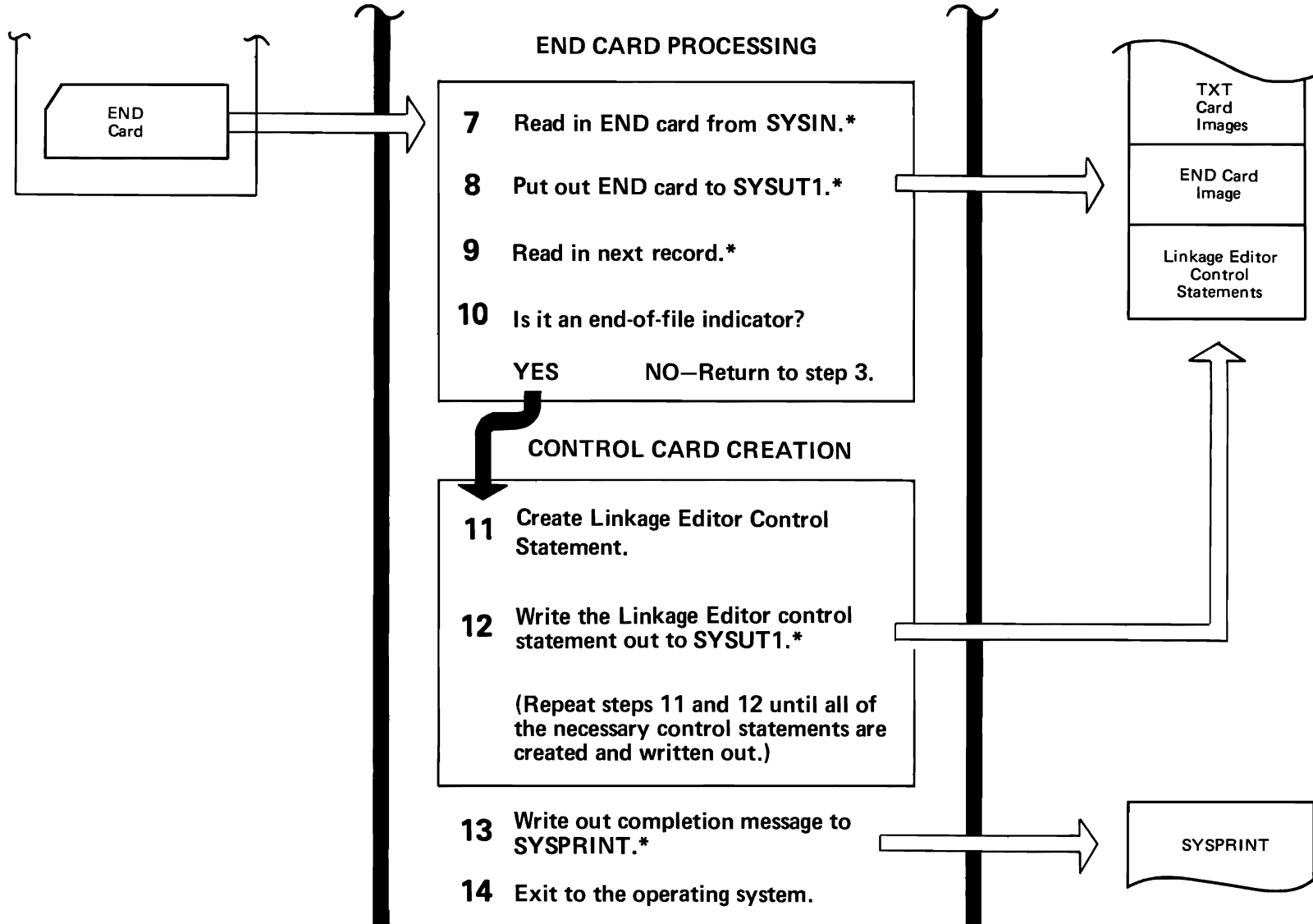


Figure 3-11. OS Control Step Method of Operation (Part 2 of 3)



\*If an error is encountered in an input card field, the program writes an error message out to SYSPRINT. If an unrecoverable I/O error is encountered, the program writes an error message out to SYSPRINT, if that file is available, or to the operator console, then terminates.

Description	Routine	Chart
<b>1</b> Opens and checks the opening of the SYSIN, SYSPRINT, and SYSUT1 files.	CNTLINIT	0A1
<b>2</b> Reads the ESD card into the input area, OBJCARD, and validates the deck ID, card sequence number, and card type fields.	GETCHECK	0A4
<b>3</b> Validates the name, type, address, and length fields of every ESD item on the card.	ESDPROC	0A1
<b>4</b> Puts out the card to the sequential file SYSUT1.	PUTCARD	0A3
<b>5</b> Reads the TXT card into the input area, OBJCARD, and validates the deck ID, card sequence number, and card type fields.	GETCHECK	0A4
<b>6</b> Puts out the card to the sequential file SYSUT1.	PUTCARD	0A3
<b>7</b> Reads the END card into the input area, OBJCARD, and validates the deck ID, card sequence number, and card type fields.	GETCHECK	0A4
<b>8</b> Puts out the card to the sequential file SYSUT1.	PUTCARD	0A3
<b>9</b> Reads in the next record, which should be an end-of-file indicator or the first ESD card of the next input module.	GETCHECK	0A4
<b>10</b> <ul style="list-style-type: none"> <li>● YES - Transfers control to step 11.</li> <li>● NO - Returns control to ESDPROC for the processing of the new object module.</li> </ul>	TXTPROC GETCHECK	0A3 0A4
<b>11</b> Creates the necessary INCLUDE control statement for IDFST and the OVERLAY and INSERT statements for the input CSECTs.	CNTLSTMT	0A3
<b>12</b> Puts out the statement to the sequential file SYSUT1.	PUTCARD	0A3
<b>13</b> Informs the user of the successful completion of the Control step.	MGSFAN/ DIAGWTR	0A4 0A4
<b>14</b> Closes the SYSIN, SYSPRINT, and SYSUT1 files and returns control to the operating system.	CNTLEND	0A3

Figure 3-11. OS Control Step Method of Operation (Part 3 of 3)

entire overlay program (GO module) to the output module library SYSLMOD or to a user-defined output module library. Next, it writes a message to SYSPRINT stating that the step has been completed successfully, then terminates.

*Note:* The proper functioning of the GO module executed in the Storage step depends upon the construction of the segment and entry tables in accordance with

the current Linkage Editor practices. If the format of either of these tables is altered by a subsequent release of the Operating System, code changes in the module IDFST may be necessary. If the table formats are altered and the code changes are made, the new version of the FD utility will not execute properly under an older release of the Operating System.

### *DOS Method of Operation*

The first function of the DOS Linkage Editor is to read the control statements and validated object module records in from SYSLNK and place them in the SYS001 work file. The Linkage Editor begins processing the control statements as encountered. The first six control statements are the PHASE and INCLUDE statements for the object modules IJLFST, IJLFLOAD, and IJLFUPDT. These statements cause the Linkage Editor to fetch the object modules from a relocatable library and place them in the work file. The next control statements are the PHASE and INCLUDE statements for the following input CSECTs. These statements cause the Linkage Editor to transform the CSECTs, one by one, into overlay phases and to resolve backward origins within the CSECTs. When the Linkage Editor has finished processing the control statements, it writes out the entire overlay program to the core image library. Finally, it writes out a storage map to SYSLST, stating that the step has been completed successfully, then terminates.

### **Storage Step Functions**

The primary tasks of the Storage step are to arrange the FD programs into a usable format and to place them in a user-defined FD program library. Although the purposes of the Storage step are similar in the OS and DOS versions of the utility, the methods used to achieve the purposes differ widely. These different methods of operation are described in the following paragraphs.

### *OS Method of Operation*

The first function of the OS Storage step is the opening of the system output file and the user's FD program library file. The program then determines whether these two files were opened successfully. If the file openings were successful, the program proceeds to the validation of the parameters passed from the user's JCL. If the parameters are valid, the program will construct a replacement table when it encounters a duplicate FD program. The purpose of the table is to specify the action the Storage step should take if a new FD program bears the same name as an FD program already present in the user's library. Two alternatives exist: (1) the program may replace the old program in the user's library or (2) the new program may be stored under a temporary name.

The user specifies the replacement of old FD programs through the JCL PARM feature. If the user wishes to replace all old duplicate programs with the new programs, he should code PARM='REPLACE' on the EXEC card that calls for IDFST. In some cases, the user may wish to replace only some particular old programs. To accomplish this, he must name those programs specifically by coding PARM='REPLACE=(name1, name2, . . . , namen)'. The names entered must be the same as the names entered in the

name field of the FDFORM macros and a maximum of twenty names may be specified. If the user does not name the programs to be replaced or does not employ the PARM feature, the duplicate programs are stored as new members under temporary names. Temporary names are chosen from the lowest available name in the sequence IDFTEMP0 through IDFTEMP9. If these temporary names have been exhausted, the program is not stored. In any case, the FD program names and the actions taken by the Storage step are listed in the diagnostic listing produced by the Storage step.

The next function of the Storage step is to load and process the overlay segments, one segment at a time. The program serially examines the KUPBs within each segment. It checks the name and count subfields of each KUPB, then writes out the 476-byte unpacked program block field to the user's library. By writing out only the unpacked program block field, the Storage step effectively deletes the 10-byte key field, thereby arranging the FD programs into a usable format. The process of reading in and writing out the overlay segments continues until the end-of-form indication is detected. The appropriate form of the STOW macro is then issued. If the return from the STOW indicates that the FD program is the duplicate of an FD program that already exists in the user's library, the utility will construct and consult the replacement table. The purpose of this consultation is to determine whether the new FD program should replace the older program in the user's library. If the program name is not found in the replacement table or if the user did not employ the replacement parameter, the utility attempts to store the new FD program under a temporary name. The reading and writing of the FD programs continues until the end-of-assembly indication is encountered. As a part of the FD program processing, the Storage step composes and writes out a diagnostic listing to the system printer. This listing states the names of all FD programs encountered and the actions taken by the program in each case (that is, stored under its proper name, stored under a temporary name, or not stored at all). The program then terminates.

If an error condition is encountered at any time during the execution of the Storage step, the step itself writes an error message out to the system printer, if available, or to the operator console. It ceases processing at the point of the error and terminates. Figure 3-15 represents the OS Storage step's method of operation.

### *DOS Method of Operation*

The first function of the DOS Storage step is the opening of the SYSIPT and SYSLST files. The next function is the validation and processing of the user-coded FD utility control statements in the SYSIPT file. There are three types of control statements that the user may code: (1) DEVICE, (2) OPTION, and (3) RPLACE.

Figure 3-12. DOS Control Step Method of Operation (Part 1 of 3)

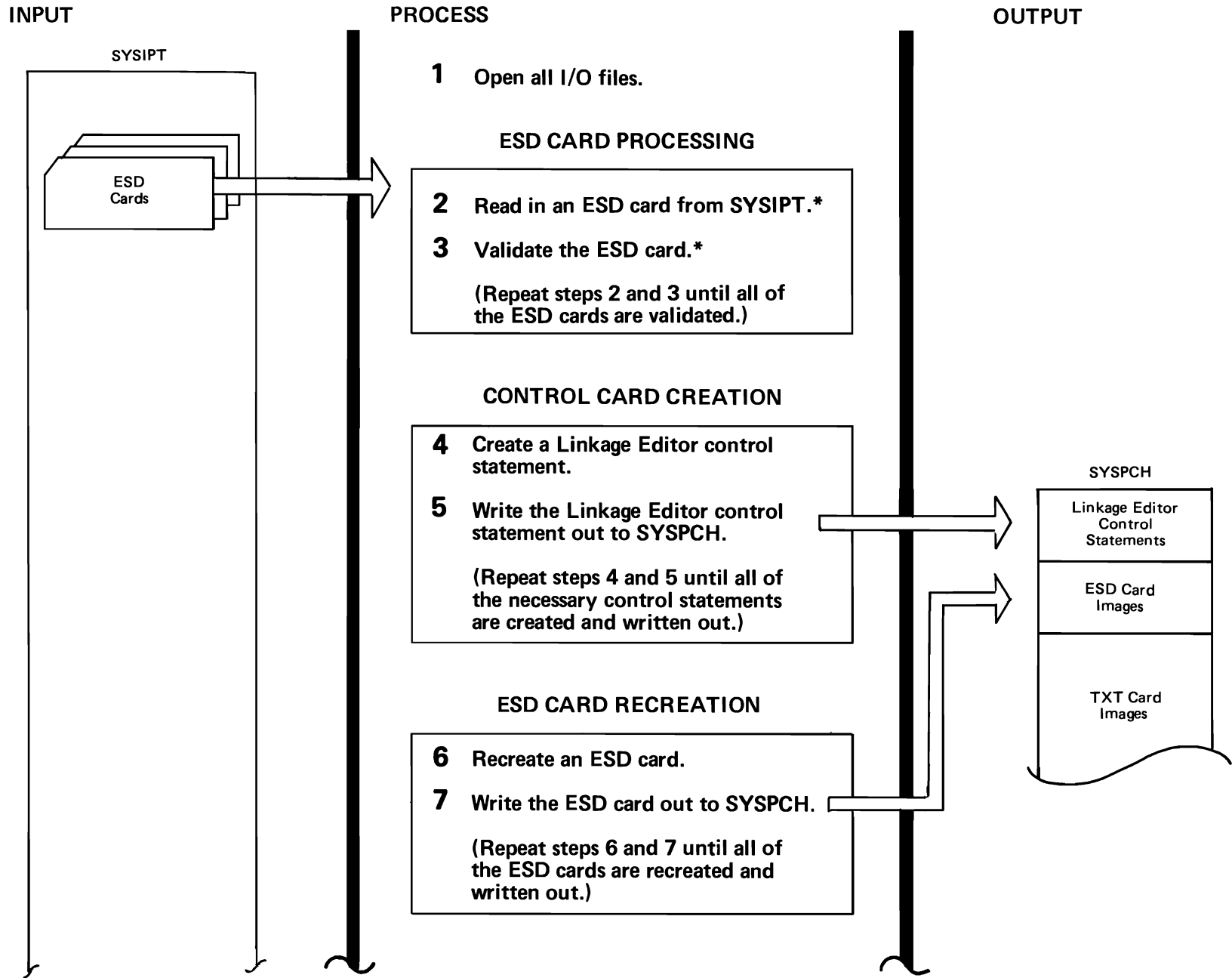
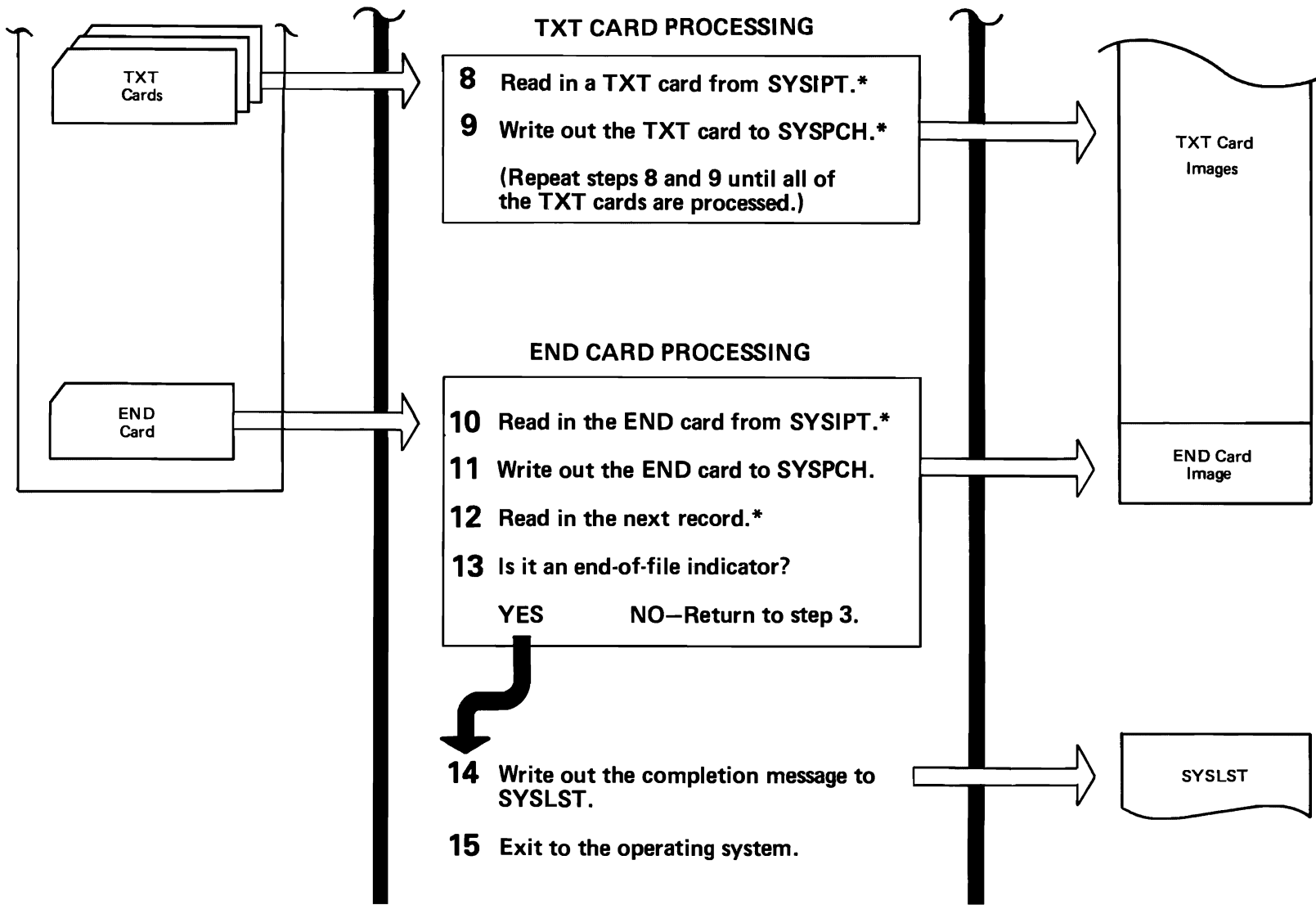




Figure 3-12. DOS Control Step Method of Operation (Part 2 of 3)



\*If an error is encountered in an input card field, the program writes an error message out to SYSLST, then terminates. If an unrecoverable I/O error is encountered, the DOS Supervisor terminates the Control step and returns control to the DOS Job Control program.

Description	Routine	Chart
<b>1</b> Opens the SYSIPT, SYSPCH, and SYSLST files.	CNTLINIT	DA1
<b>2</b> Reads the ESD card into an I/O buffer and validates the deck ID, card sequence number, and card type fields.	GETCHECK	DA3
<b>3</b> Validates the name, type, address, and length fields of every ESD item on the card.	ESDPROC	DA1
<b>4</b> Creates the necessary PHASE and INCLUDE control statements for the object module IJLFST and for the input CSECTs.	CNTLSTMT	DA1
<b>5</b> Writes out the Linkage Editor control statement to the sequential file SYSPCH.	PUTCARD	DA3
<b>6</b> Recreates an ESD card in an I/O buffer, using predictable values and decrementing the count of ESD cards to be created.	CNTLSTMT	DA1
<b>7</b> Writes out the newly recreated ESD card to the sequential file SYSPCH.	PUTCARD	DA3
<b>8</b> Reads the TXT card into an I/O buffer and validates the deck ID, card sequence number, and card type fields.	GETCHECK	DA3
<b>9</b> Writes out the TXT card to the sequential file SYSPCH.	PUTCARD	DA3
<b>10</b> Reads the END card into an I/O buffer and validates the deck ID, card sequence number, and card type fields.	GETCHECK	DA3
<b>11</b> Writes out the END card to the sequential file SYSPCH.	PUTCARD	DA3
<b>12</b> Reads in the next record, which should be an end-of-file indicator or the first ESD card of the next input module.	GETCHECK	DA3
<b>13</b> <ul style="list-style-type: none"> <li>● YES - Transfers control to step 14.</li> <li>● NO - Returns control to ESDPROC for the processing of the new object module.</li> </ul>	TXTPROC GETCHECK	DA2 DA3
<b>14</b> Informs the user of the successful completion of the Control step.	MSGFAN/ DIAGWTR/ PUTCARD	DA2 DA2 DA3
<b>15</b> Closes the SYSIPT, SYSPCH, and SYSLST files and returns control to the operating system.		DA2

Figure 3-12. DOS Control Step Method of Operation (Part 3 of 3)

TEST Indicator	Address of Data Control Block (DCB) used to load module *		
Address of note list *			
Last segment number of region 1	Highest segment no. in storage-region 1	Last segment number of region 2	Highest segment no. in storage-region 2
Last segment number of region 3	Highest segment no. in storage-region 3	Last segment number of region 4	Highest segment no. in storage-region 4
Zero	(Not used in the Fixed-Task Supervisor) *		
(Not used in the Fixed-Task Supervisor) *			
Previous segment number for segment 1 *	Zero		Status Indicator
Previous segment number for segment 2	Address of entry table entry (when caller chain exists) *		Status Indicator
: : : :			
: : : :			
Previous segment number for segment N	Address of entry table entry (when caller chain exists) *		Status Indicator
← 4 bytes →			

TEST indicator – specifies that this module is “under test” using TESTRAN. (Bit 1) Initialized by program fetch.

Highest segment no. in storage – is initially set to 00 except for region 1 which is initially set to 01 by linkage editor.

Status indicator – indicates the status of this segment with the two last bits of the entry table address field as follows:

- 00 – segment is in main storage as a result of a branch to the segment.
- 10 – segment is in main storage, no caller chain exists.
- 01 – segment is not in main storage, but is scheduled to be loaded.
- 11 – segment is not in main storage.

The status indicator for segment 1 is initially set to 10, all the rest are initially set to 11.

\* set to zero by linkage editor

Figure 3-13. SEGTAB Contents

Unconditional branch to last entry-BC 15, DISP (15,0)	Address of referred to symbol		“to” seg number	Previous Caller (zero initially)
SVC 45	L 15,4(0,15) Loads GR15 with the value of the ADCON	BCR 15, 15	“from” seg no	Address of segment table (SEGTAB)
← 2 bytes →	← 2 bytes →	← 2 bytes →	← 2 bytes →	← 1 byte →
				← 3 bytes →

DISP – is the displacement, in bytes, of this entry from the last entry.

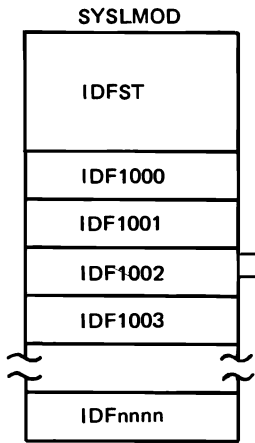
“to” segment number – is the number of the segment containing the symbol being referred to.

“from” segment number – is the number of the segment that contains this entry table.

Figure 3-14. ENTAB Contents

Figure 3-15. OS Storage Step Method of Operation (Part 1 of 3)

INPUT



PROCESS

- 1 Open files and check opening.\*
- 2 Validate the passed parameters and create the replacement table.

SEGMENT PROCESSING

- 3 Load an overlay segment from SYSLMOD.\*
  - 4 Determine the replacement status of the new FD program.
  - 5 Write out the unpacked program block field of a KUPB to the SYSLIB file.\*
- (Repeat step 5 until all of the KUPBs within the segment have been processed or until the end-of-form indication is encountered. Repeat steps 3 and 5 until all of the KUPBs within a single FD program have been processed.)

OUTPUT

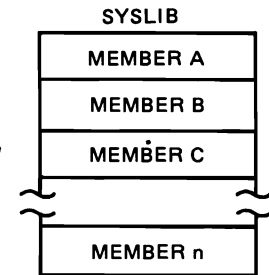
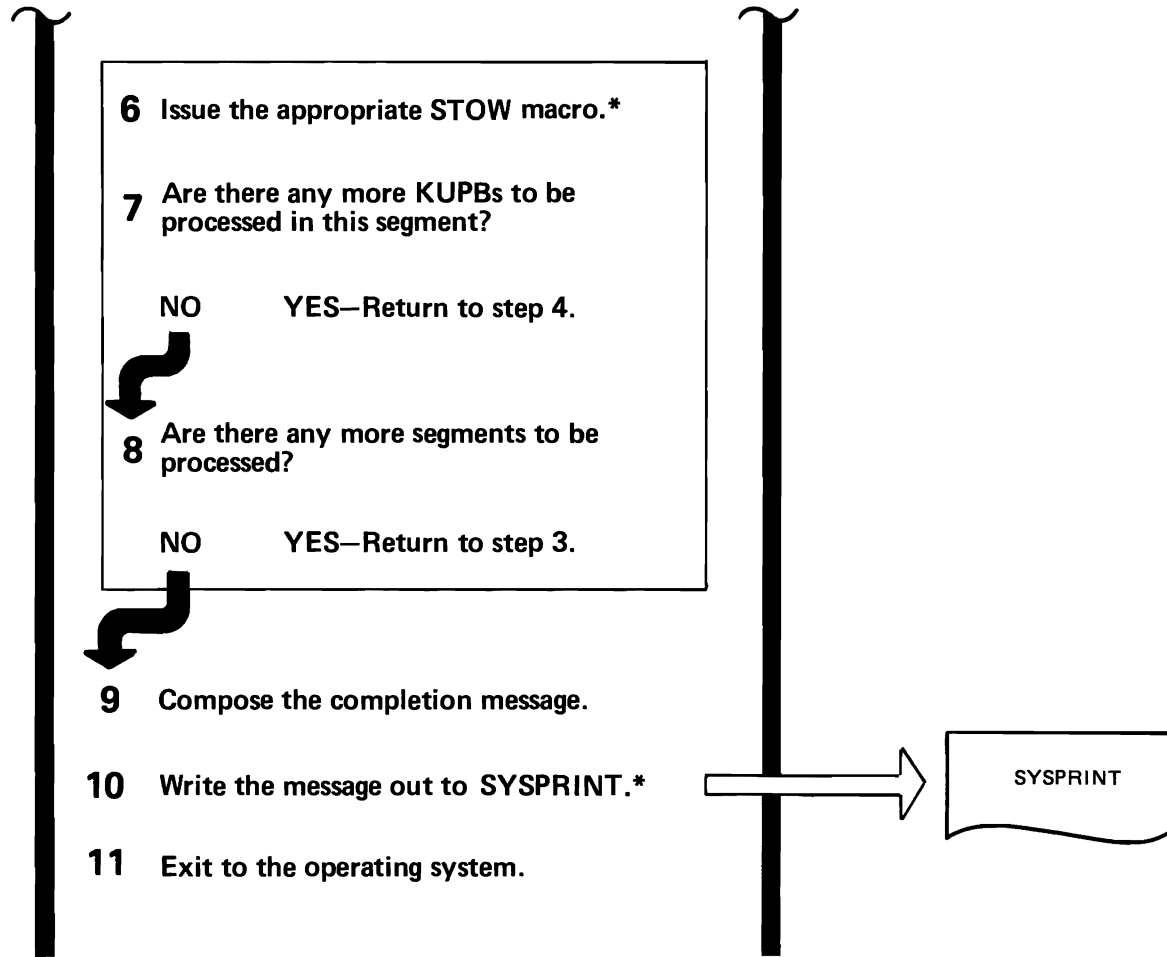


Figure 3-15. OS Storage Step Method of Operation (Part 2 of 3)



\*If an unrecoverable I/O error is encountered, the program writes an error message out to **SYSPRINT**, if that file is available, or to the operator console. It then terminates.

	Description	Routine	Chart
1	Opens and checks the opening of the SYSLIB (user's library) and SYSPRINT files.	STGINIT	OB1
2	Ensures that the passed parameters are in the proper format, then builds the replacement table, REPLTAB.	PARMPROC	OB1
3	Issues a SEGWT supervisor call to bring the overlay segment into main storage.	SEGPROC	OB4
4	Consults REPLTAB to determine the replacement status of the FD program.	NEWMEM	OB1
5	Examines the count subfield of the KUPB to ensure that it is in the proper sequence, then writes out the 476-byte unpacked program block field to the user's FD program library.	NEWMEM	OB1
6	Uses the previously determined replacement status to determine the form of the STOW macro to be used, then issues the STOW command.	NEWMEM	OB1
7	Resumes processing with the next KUPB, if one is available in the current segment.	NEWMEM	OB1
8	Resumes processing with the next segment, if one is available.	NEWMEM	OB1
9	Composes the completion message, which contains the names of all FD programs processed, their lengths (number of KUPBs), and the action taken by the Storage step.	MSGFAN/ DIAGWTR	OB4 OB4
10	Informs the user of the successful completion of the Storage step.	DIAGWTR	OB4
11	Closes the SYSLIB and SYSPRINT files and returns control to the operating system.	STGEND	OB4

Figure 3-15. OS Storage Step Method of Operation (Part 3 of 3)

- The DEVICE control statement is used to specify the type of DASD on which the user's FD program library resides. It must be coded in the following format:

```
// DEVICE=2311
or // DEVICE=2314
```

- The OPTION control statement is used to specify the type of operation to be performed by the Storage step. It must be coded in the following format:

```
// OPTION=LOAD
or // OPTION=LOADFST
or // OPTION=UPDATE
```

- The RPLACE control statement is used to specify which old FD programs are to be replaced by new (duplicate) FD programs during an update operation. It must be coded in the following format:

```
// RPLACE
or // RPLACE=name
```

where "name" is the same as the name of an FD program specified in the name field of that program's FDFORM macro.

The user specifies the LOAD option as the first step of a two-step operation when he is creating a new FD program

library. The selection of this option during the execution of the phase IJLFST causes IJLFST to fetch and execute the overlay phase IJLFLOAD. This phase initializes the new library by loading the first FD program from the core image library and writing it out to IJFDLIB (the user's ISAM FD program library). The second step is accomplished when the user specifies the LOADFST ("load first") option during a second execution of IJLFST. The selection of this option causes IJLFST to fetch and execute the overlay phase IJLFUPDT. This phase, when executing under the LOADFST option, loads and writes out the remaining FD programs, thus completing the creation of the new FD program library. When the user wishes to update an existing FD program library, that is, to add new FD programs or replace old ones, he specifies the UPDATE option. The selection of this option causes IJLFST to load and execute the overlay phase IJLFUPDT. IJLFUPDT, when executing under the UPDATE option, performs the necessary additions and/or replacements and revises the library's index to reflect these changes. Replacements are performed in accordance with the operand (s) specified in the RPLACE control statement (s).

As previously stated, the user uses the RPLACE control statements during an update operation to control the replacement of old FD programs by new duplicates. The use of these control statements in the input stream causes IJLFST to construct a replacement table, which is passed to IJLFUPDT when that phase is loaded. If the user wishes to replace all old duplicate FD programs with the new programs, he should code "// RPLACE". In some cases, however, the user may wish to replace only some particular old programs. To accomplish this, he must name those programs specifically by "// RPLACE=name", as described previously. In this format, only one FD program name may be specified on a control statement, and a maximum of twenty FD programs may be so specified. If the user does not name the programs to be replaced or does not employ the RPLACE control statement at all, the duplicate programs are stored as new members under temporary names. Temporary names are chosen from the lowest available name in the sequence IJLFTM00 through IJLFTM09. If these temporary names have been exhausted, the program is not stored. In any case, the FD program names and the actions taken by the Storage step are listed in the diagnostic listing produced by the Storage step.

The processing of the FD programs in the Storage step is basically the same in each type of operation (LOAD,

LOADFST, or UPDATE). The overlay phases created in the Linkage Editor step are loaded and processed one by one. The program processes the KUPBs within each phase serially. It checks the name and count subfields of a KUPB, then writes out the 476-byte unpacked program block field to IJFDLIB with a 10-byte key field, thereby arranging the FD programs into a usable format. If IJLFLOAD encounters any type of error during this processing, it issues a descriptive error message and terminates immediately. IJLFUPDT, on the other hand, responds to an error within an FD program by flushing that program, issuing an error message, and continuing processing with the next available FD program. However, environmental errors, such as a data area overflow, cause IJLFUPDT to issue a descriptive error message and terminate.

The response of the Storage step to the detection of a duplicate FD program depends upon the type of operation being performed. If the duplicate is detected during a LOAD operation, the program issues an error message and terminates. If a LOADFST operation is in progress, the program automatically replaces the old FD program with the new one. If the UPDATE operation is being performed when the duplicate FD program is encountered, the program consults the previously constructed replacement table. The purpose of this consultation is to determine whether the new FD program should replace the older program in the user's library. If the user has specified // RPLACE, the older program is replaced automatically. However, if // RPLACE was not specified, then the program must search the replacement table for the name of the duplicate FD program. If the program finds the FD program's name in the replacement table, it replaces the older FD program with the new one. If the FD program's name is not found or if the user did not employ any // RPLACE control statements, the program attempts to store the FD program under a temporary name, as described previously.

The loading and writing of the FD programs continues until all of the overlay phases have been processed. As a part of this processing, the Storage step composes and writes out a diagnostic listing to the system printer. This listing states the names of all FD programs encountered and the action taken by the Storage step in each case (that is, stored under its proper name, stored under a temporary name, or not stored at all). When all processing has been completed, the program writes out a message stating the fact, then terminates. Figure 3-16 represents the DOS Storage step's method of operation.

Figure 3-16. DOS Storage Step Method of Operation (Part 1 of 5)

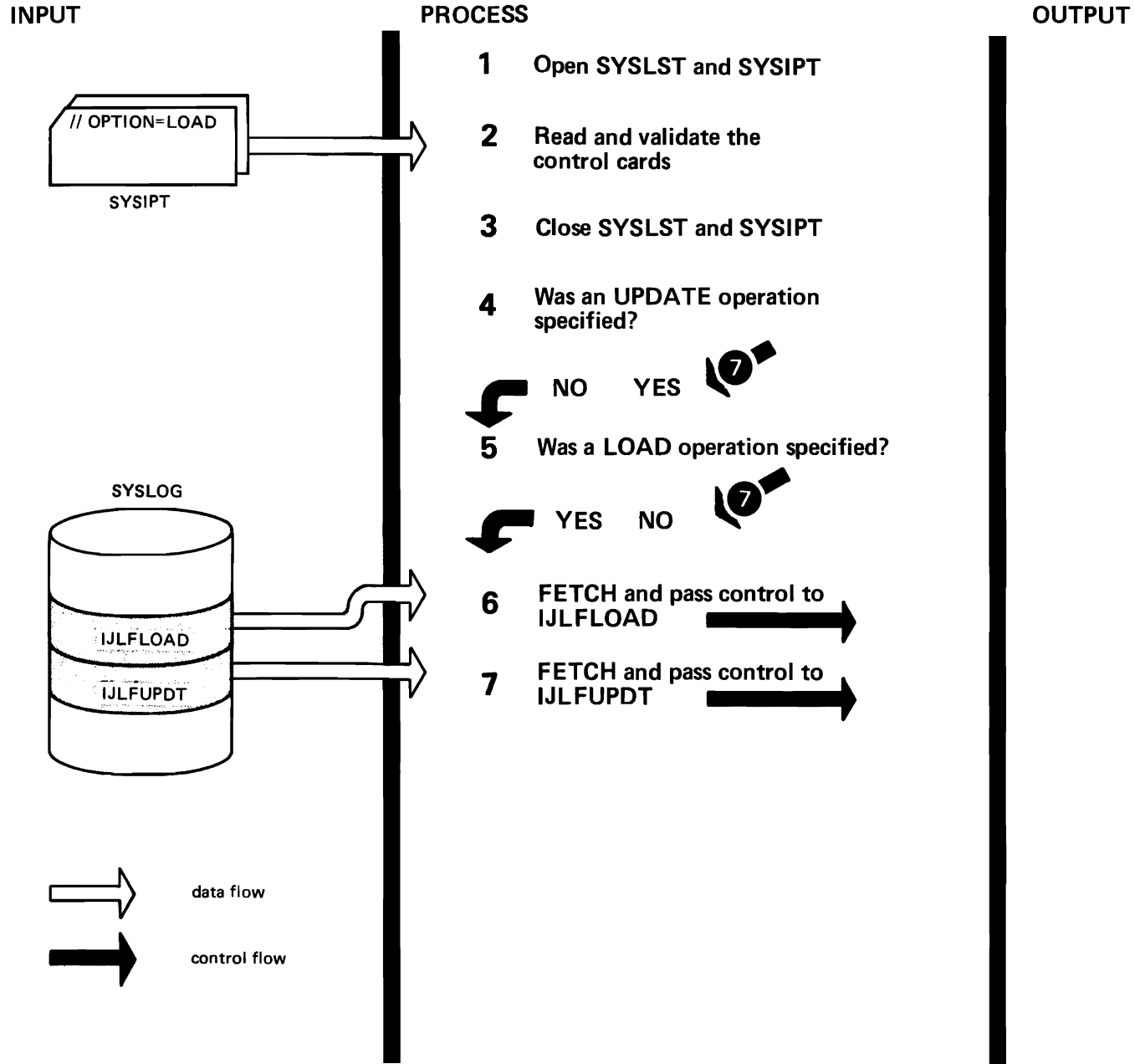
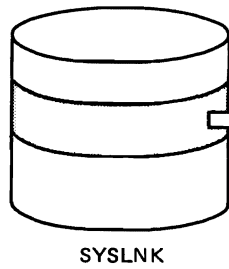




Figure 3-16. DOS Storage Step Method of Operation (Part 2 of 5)

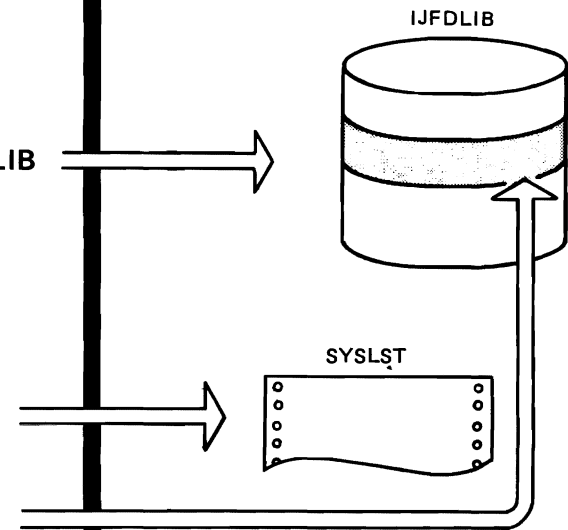
INPUT



PROCESS

- 1 Open IJFDLIB and SYSLST
- 2 Load an overlay phase into main storage from SYSLNK
- 3 Validate the name and count subfields of a KUPB within the phase
- 4 Write the validated KUPB out to IJFDLIB  
(Repeat steps 3 and 4 until all of the KUPBs within the phase have been validated and written out. Repeat steps 2 through 4 until all of the phases on SYSLNK have been processed.)
- 5 Write a message out to SYSLST indicating successful completion
- 6 Write an EOF indication out to IJFDLIB
- 7 Close IJFDLIB and SYSLST
- 8 Exit to the operating system

OUTPUT

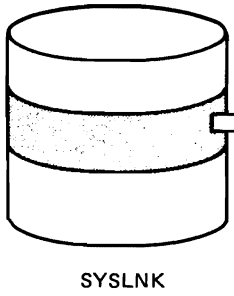


Description	Routine	Chart
<p><b>2</b> Each overlay phase is loaded into a 1464-byte area within IJLFLOAD named LOADAREA. The LOAD macro is used to accomplish this task.</p>	LOADRTN	DC1
<p><b>3</b> If either of the fields is invalid, control is passed to the message Fan-in routine, MSGFAN, for the printing of an explanatory error message, and then to the End-of-Job routine, EOJRTN.</p>	RCDCHK	DC1
<p><b>4</b> The program first issues a SETFL macro to initialize the output data area, then writes the KUPB out. If the record is not written out properly, the program analyzes the error, then passes control to MSGFAN and EOJRTN.</p>	ISMPUT	DC2

Figure 3-16. DOS Storage Step Method of Operation (Part 3 of 5)

Figure 3-16. DOS Storage Step Method of Operation (Part 4 of 5)


INPUT



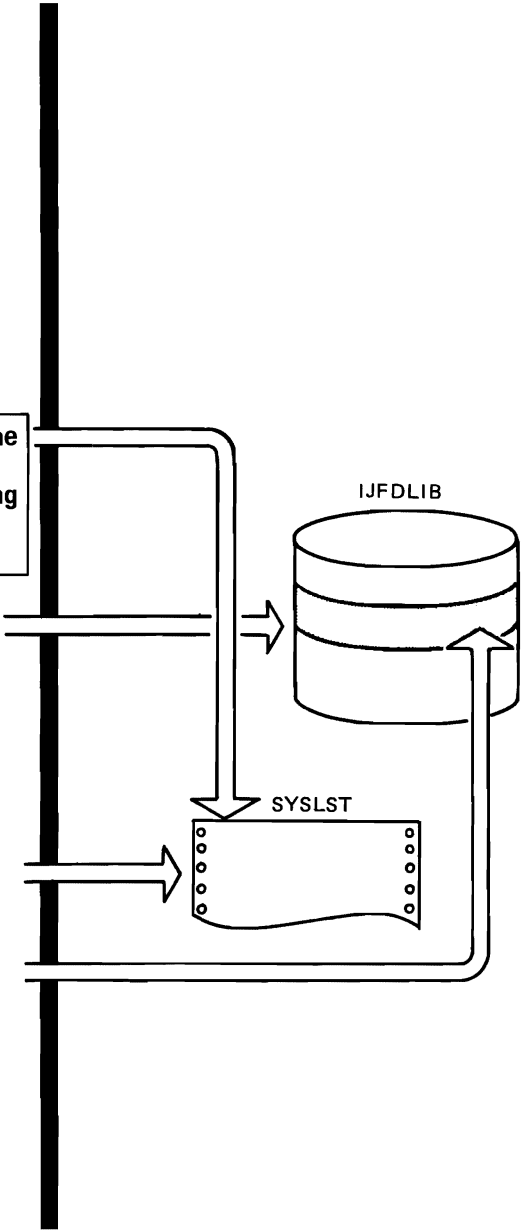
PROCESS

- 1 Open IJFDLIB and SYSLST
- 2 Load an overlay phase into main storage from SYSLNK
- 3 Validate the name and count subfields of a KUPB within the phase
- 4 Were the fields valid?  
YES NO
- 5 Write the validated KUPB out to IJFDLIB  
  
(Repeat steps 3 through 5 until all of the KUPBs within the phase have been validated and written out. Repeat steps 2 through 5 until all of the phases on SYSLNK have been processed.)
- 6 Write a message out to SYSLST indicating successful completion
- 7 Write an EOF indication out to IJFDLIB
- 8 Close IJFDLIB and SYSLST
- 9 Exit to the operating system

Print out a message defining the error, flush the erroneous FD program, and resume processing with the next FD program at step 2.



OUTPUT



Description	Routine	Chart
<p><b>2</b> Each overlay phase is loaded into a 1464-byte area within IJLFUPDT named LOADAREA. The LOAD macro is used to accomplish this task.</p>	LOADRTN	DD2
<p><b>5</b> The program first determines whether or not the KUPB is a duplicate. If it is, and it is supposed to be replaced, the program replaces it in IJFDLIB. If it is a duplicate, but is not supposed to be replaced, the program stores it under a temporary name. If the KUPB is not a duplicate, the program checks for the following I/O errors: (1) DASD error, (2) wrong length record, (3) EOF written out, (4) no output record found, and (5) overflow area full. If any of these errors is found and cannot be corrected, the program prints out an error message and terminates.</p>	ISMPUT	DD2 & DD3

Figure 3-16. DOS Storage Step Method of Operation (Part 5 of 5)

## FD UTILITY ORGANIZATION

The aims of the FD utility, which were stated in the previous section, are accomplished through the logical organization of the program in three sequential job steps: Control step, Linkage Editor step, and Storage step. The Control step verifies the correctness of the object module input (except that data generated in columns 17-72 of the TXT card images) and generates the necessary Linkage Editor control statements for the next step. The Linkage Editor step performs the linkage editing of the input object modules, under control of the control statements generated in the first step. The Linkage Editor step also includes the object module IDFST (under OS) or IJLFST, IJLFLOAD, and IJLFUPDT (under DOS) in the input. The third and last step, the Storage step, is the execution of the linkage-edited overlay program. The data originally generated by the FD macro assembly is brought into main storage as overlay segments and put out to the user's private data set in 476-byte blocks. Additionally, an optional JCL parameter (OS) or control statement (DOS) may be used to allow the storage of FD programs whose names are the same as existing members of the user's data set.

Under OS, the program is comprised of three modules:

IDFCT	The Control step load module;
IDFM01	The message module that contains the messages put out to SYSPRINT or the operator console;
IDFST	The object module that becomes the root segment of the overlay program executed in the Storage step.

Under DOS, the program is comprised of five modules:

IJLFCT	The Control step phase;
IJLFM01	The message module that contains the messages put out to SYSLST;
IJLFST	The object module that becomes the root phase of the overlay program executed in the Storage step;
IJLFLOAD	The object modules that become Storage step overlay IJLFUPDT phases for the processing of the FD programs.

Figures 3-17 and 3-18 illustrate the residency of these modules in auxiliary storage for OS and DOS, respectively.

### Control Step Organization

Although the organization of the modules IDFCT and IJLFCT is similar, enough differences exist to merit a separate discussion of each. These modules are described in the following paragraphs.

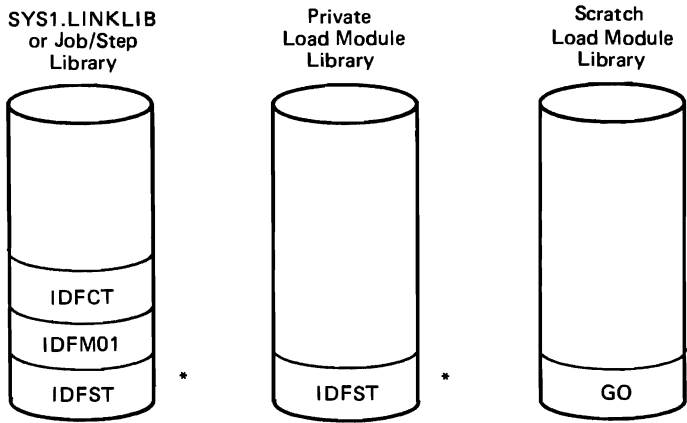
### IDFCT Organization

The load module IDFCT is logically organized in 10 routines, an 80-byte card storage area, and three data control blocks (DCBs). Figure 3-19 represents the physical organization of IDFCT. The logical flow of the 10 routines is graphically represented in Charts OA1 through OA5 at the rear of this section. Figure 3-20 illustrates the hierarchy of the routines within IDFCT.

*CNTLINIT - Control Initial:* The control initial routine opens and tests the opening of the SYSIN, SYSUT1, and SYSPRINT files. Control passes automatically to the ESD processor routine unless one of the files fails to open properly. If the opening of either SYSIN or SYSUT1 fails, CNTLINIT gives control to the message fan-in routine so that a message may be written to the system printer. If the opening of SYSPRINT fails, CNTLINIT writes a message out to the operator console, then terminates.

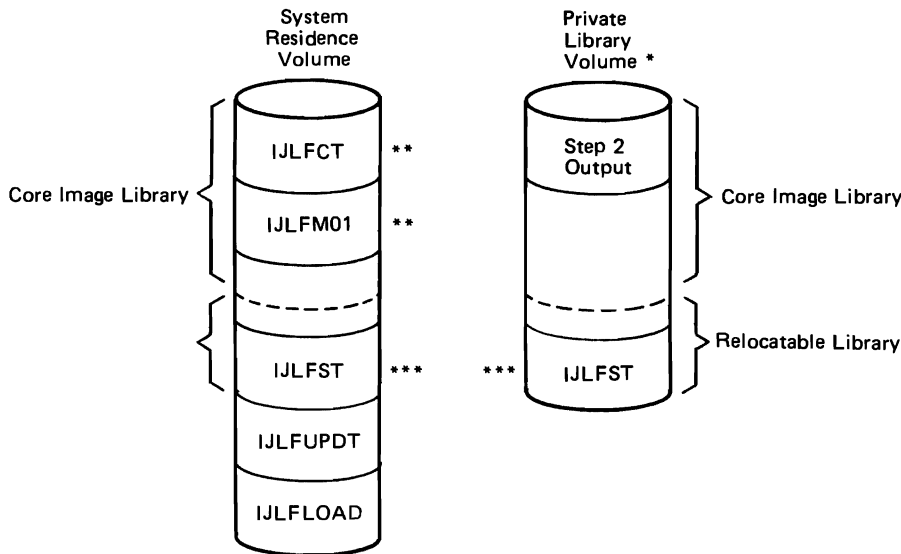
*ESDPROC - ESD Processor:* The ESD processor routine receives control from CNTLINIT at the start of processing. It may also receive control from the TXT processor routine (TXTPROC) if multiple input modules are present. ESDPROC first invokes the get and check an input record routine (GETCHECK) to read in a card from SYSIN. If the card that is read in is a TXT card, the ESD processor transfers control to TXTPROC. Otherwise, it validates the name field, type code, address field, and length field of each ESD item on the card. Next, the routine invokes the put out a card routine (PUTCARD) to put the validated card out to SYSUT1, then again invokes GETCHECK to read in another card. Finally, when all of the ESD cards have been validated and the first TXT card has been read in, it gives control to TXTPROC. If ESDPROC encounters an error during processing, it ceases processing immediately and gives control to the appropriate point in the message fan-in routine so that a descriptive error message may be written out to the SYSPRINT file.

*TXTPROC - TXT Processor:* The TXT processor routine receives control from the ESD processor routine after the first TXT card has been read into main storage. It invokes PUTCARD to put out that first TXT card, then alternately invokes GETCHECK and PUTCARD to read in and put out the remaining TXT cards. TXTPROC performs this reading and writing function until it encounters an END card. At that point, it puts out the END card and reads the next record. If the next record read is the end-of-file



\* IDFST may reside in either of these libraries.

Figure 3-17. OS FD Utility Auxiliary Storage Residency



\* If the Private Library Volume is not used, all of the modules are written on the System Residence Volume.

\*\* These modules are linkage edited during system generation. IJLFST may reside on either volume.

\*\*\* This module is linkage edited during the FD Utility Linkage Editor step.

Figure 3-18. DOS FD Utility Auxiliary Storage Residency

indication, the routine passes control to the create control statements routine (CNTLSTMT). However, if the next record is an ESD card, then multiple input object decks are present and the routine sets the multiple object deck indicator (NEWJOB) to one and the address counter (ADDRCTR) to zero, then returns control to ESDPROC. On the other hand, if the next record read is neither an end-of-file indication nor an ESD card, then an error condition exists. This error is detectable by GETCHECK. If it occurs,

TXTPROC does not regain control from GETCHECK. Rather, GETCHECK passes control to the appropriate point in the message fan-in routine so that a descriptive error message may be written out to SYSPRINT.

*CNTLSTMT - Create Control Statements:* The create control statements routine receives control from the end-of-file subroutine that is contained within TXTPROC. By the time it gains control, all ESD, TXT, and END cards have

Control Initial	(CNTLINIT)
ESD Processor	(ESDPROC)
TXT Processor	(TXTPROC)
Create Control Statements	(CNTLSTMT)
Control End	(CNTLEND)
Message Fan-in	(MSGFAN)
Diagnostic Writer	(DIAGWTR)
Get and Check an Input Record	(GETCHECK)
Put Out a Card	(PUTCARD)
Write Operator	(WTORTN)
Control Synchronous I/O Error	(CTLSYNER)
Card Storage	(OBJCARD)
SYSIN DCB	(INPUT)
SYSUT1 DCB	(OUTPUT)
SYSPRINT DCB	(PRINT)

Figure 3-19. Module IDFCT Physical Organization

been written out to SYSUT1. The purpose of CNTLSTMT is to create and write out the Linkage Editor control statements necessary to produce the overlay program (GO module) that is executed in the Storage step. As soon as CNTLSTMT creates a new control statement, it invokes PUTCARD to write it out to the SYSUT1 file. The first statement created is the INCLUDE control statement, which requests the Linkage Editor to use an additional data set as input. The INCLUDE statement specifies the ddname of the DD statement that describes the object module IDFST. The format of this statement is as follows:

```
⊘ INCLUDE SYSDD(IDFST)
```

When this statement has been created and written out, CNTLSTMT creates and writes out a series of OVERLAY

and INSERT statements. The format of these statements is as follows:

```
⊘ OVERLAY A
⊘ INSERT IDFnnnn
where IDFnnnn is the name of an input CSECT that will
become an overlay segment.
```

CNTLSTMT creates only one INCLUDE statement, but it creates one OVERLAY statement and one INSERT statement for each input CSECT. When all of the necessary Linkage Editor control statements have been created and written out, CNTLSTMT passes control to the message fan-in routine.

*GETCHECK - Get and Check an Input Record:* The get and check an input record routine is invoked by ESDPROC and

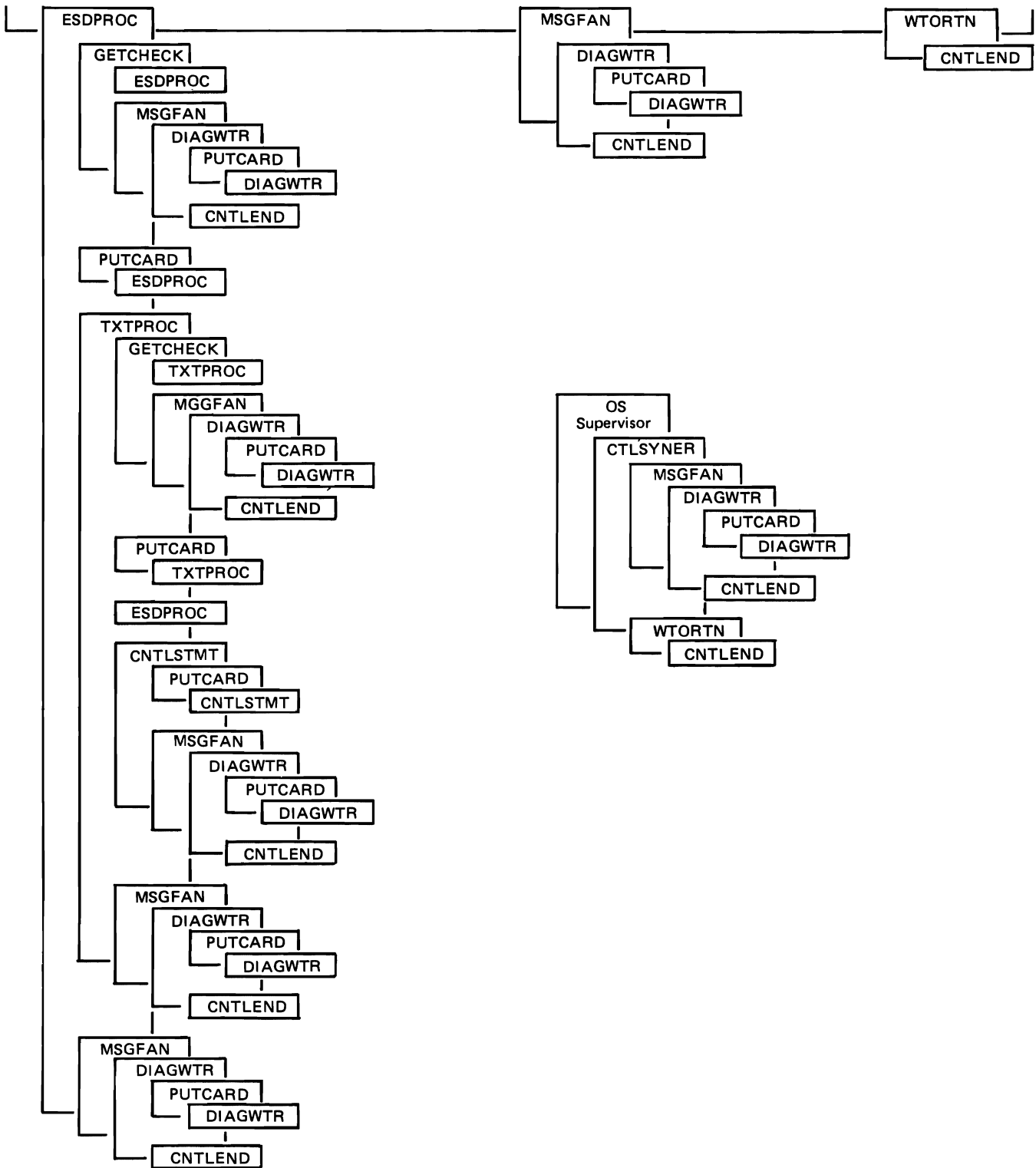


Figure 3-20. Module IDFCT Hierarchy of Routines



TXTPROC to read in the next sequential record from SYSIN and to check three fields of the record. The three fields checked are the deck ID, columns 73-76; card sequence number, columns 77-80; and card type, columns 2-4. GETCHECK validates the deck ID field to ensure that the four-character deck identifier is the same on every card of the input object deck. It validates the card sequence number field to ensure that no input records are out of place or missing. Finally, GETCHECK checks the card type field to ensure that the card types themselves are in the correct sequence. That is, an ESD card must be followed by another ESD card or a TXT card; a TXT card must be followed by another TXT card or an END card; an END card must be followed by an ESD card (indicating that another FD program module follows) or an end-of-file indication. If no errors are detected in the three fields, GETCHECK returns control to the routine that called it. However, if GETCHECK does detect an error, it ceases processing immediately and passes control to the appropriate point in the message fan-in routine so that a descriptive error message may be written out to SYSPRINT. If an end-of-file indicator or an irrecoverable I/O error is encountered during the reading of SYSIN, GETCHECK does not retain control. These conditions are handled by the end-of-file subroutine of TXTPROC and the control synchronous I/O error routine (CTLSYNER), respectively. GETCHECK uses the Queued Sequential Access Method (QSAM); the GET macro that is used to read in the records operates in the move mode.

*PUTCARD - Put Out A Card:* The put out a card routine is invoked by ESDPROC, TXTPROC, and CNTLSTMT to write out a record to the SYSUT1 file. After the record has been put out successfully, PUTCARD returns control to the routine that called it. If an irrecoverable I/O error is encountered during the writing of a record, PUTCARD does not retain control. This error condition is handled by CTLSYNER. The PUTCARD access method is QSAM; the PUT macro that is used to write out the records operates in the move mode.

*CTLSYNER - Control Synchronous I/O Error:* The control synchronous I/O error routine is invoked when an irrecoverable error is encountered during an input or output operation. There are three entry points to the routine: SYNAD1, SYNAD2, and SYNAD3. These entry points are actually symbolic addresses that correspond to the three I/O files used by the Control step. SYNAD1 corresponds to SYSIN; SYNAD2, to SYSUT1; and SYNAD3, to SYSPRINT. The purpose of CTLSYNER is to determine the cause and type of error that occurred and to create an error message that explains the problem. The routine uses the SYNADAF macro to define the error and construct the error message. This macro examines the following data areas and records the pertinent information.

- The contents of the general registers
- The data event control block (DECB)
- The exceptional condition code
- The status and sense indications

After examining this data, the macro creates a descriptive error message that may be written out by either the diagnostic writer routine or by CTLSYNER itself. When the error message has been created, CTLSYNER modifies the return address so that control will not be given to the instruction that immediately follows the GET or PUT macro instruction that encountered the I/O error. Rather, CTLSYNER gives control to the appropriate point in MSGFAN, if the SYSPRINT file is available, or to the control end routine, if the SYSPRINT file has sustained an irrecoverable error.

*MSGFAN - Message Fan-In:* The message fan-in routine may be invoked by any of the following routines: CNTLINIT, ESDPROC, TXTPROC, GETCHECK, CTLSYNER, and CNTLEND. MSGFAN consists of a series of branch and link (BAL) instructions, each of which corresponds to a message contained in the message module, IDFM01. The routine may be entered at any one of these BAL instructions. The purpose of MSGFAN is to establish a binary number and place it in register 3. This binary number equals the displacement between the start of MSGFAN and the point at which the entry to the routine was made. The number is used later by DIAGWTR for scanning a second-level index in IDFM01 to retrieve the variable detail line (s) of the message. When the binary number has been established in register 3, the routine transfers control to DIAGWTR.

*DIAGWTR - Diagnostic Writer:* The diagnostic writer routine is invoked by the message fan-in routine when all of the input record processing has been completed successfully or when an error condition has been detected and the SYSPRINT file can be used. First, DIAGWTR issues a LOAD macro to bring the message module, IDFM01, into main storage. Next, the routine moves the three variable characters of the message ID into the output area, OBJCARD. DIAGWTR then uses the binary number located in register 3, which is the number placed there by MSGFAN, to recover the detail line (s) of the message. Finally, the routine composes the rest of the header line, including the remainder of the message ID, then writes the header and detail lines out to the SYSPRINT file. At this point, DIAGWTR passes control to the control end routine, CNTLEND.

*CNTLEND - Control End:* The control end routine is invoked by either DIAGWTR or WTORTN. It may be called when all of the input record processing has been completed successfully or when any type of error condition

has been encountered. The purpose of CNTLEND is to close the SYSIN, SYSUT1, and SYSPRINT files. When these files have been closed, the routine gives control to the operating system.

**OBJCARD - Card Image Storage Area:** The format of the 80-byte card image storage area is the same as the format of the ESD card described in Figure 3 in Part 3, Section 2, "Method of Operation." This storage area is used to contain ESD, TXT, and END cards during validation, as well as Linkage Editor control statements and the messages produced by the diagnostic writer routine.

**Input DCB:** The INPUT data control block describes the characteristics of the SYSIN file as follows:

```
DDNAME=SYSIN
LRECL=80
MACRF=(GM)
DSORG=PS
EODAD=EOFRTN
SYNAD=SYNAD1
RECFM=FB
```

**Print DCB:** The PRINT data control block describes the characteristics of the SYSPRINT file as follows:

```
DDNAME=SYSPRINT
MACRF=(PM)
DSORG=PS
SYNAD=SYNAD3
```

**Output DCB:** The OUTPUT data control block describes the characteristics of the SYSUT1 file as follows:

```
DDNAME=SYSUT 1
LRECL=80
MACRF=(PM)
RECFM=F
DSORG=PS
SYNAD=SYNAD2
BLKSIZE=80
```

#### **IJLFCT Organization**

The phase IJLFCT is logically organized in nine routines, three DTFDI macros, and one device independent module (DIMOD), plus five 81-byte I/O and work areas. Figure 3-21 represents the physical organization of IJLFCT. The logical flow of the nine routines is graphically represented in Charts DA1 through DA4 at the rear of this section. Figure 3-22 illustrates the hierarchy of the routines within IJLFCT. The routine, macros, and the DIMOD are described in the following paragraphs.

**CNTLINIT - Control Initial:** The control initial routine opens the SYSIPT, SYSPCH, and SYSLST files. Control passes automatically to the ESD processor routine unless one of the files fails to open. If one or more of the files

fails to open, the DOS Supervisor terminates the Control step and returns control to the DOS Job Control program.

**ESDPROC - ESD Processor:** The ESD processor receives control from the control initial routine at the start of processing. It may also receive control from the TXT processor routine if multiple input modules are present. ESDPROC first invokes the get and check an input record routine (GETCHECK) to read in a card from SYSIPT and place it in one of two input/output buffers. The routine then validates the name field, type code, address field, and length field of each ESD item on the card. Next, it invokes GETCHECK to read in another card and place it in the other I/O buffer. Again, ESDPROC validates the ESD items, then transfers control to GETCHECK. This reading and validating process continues until the first TXT card is encountered. Each card that is read in is placed in the I/O buffer opposite from the most recently validated card. Also, ESDPROC maintains a count of the ESD items that it validates. When the first TXT card is encountered, one I/O buffer contains that TXT card and the other I/O buffer contains the last ESD card. At this point, ESDPROC gives control to the create control statements routine (CNTLSTMT). If ESDPROC encounters an error during processing, it gives control to the appropriate point in the message fan-in routine (MSGFAN) so that a descriptive error message may be written out to the SYSLST file.

**CNTLSTMT - Create Control Statements:** The create control statements routine receives control from the ESD processor routine. By the time it gains control, all of the ESD items have been read in and validated, the count of ESD items has been completed, and the first TXT card has been encountered. Additionally, the last ESD card is contained in one I/O buffer and the first TXT card is contained in the other. The purpose of CNTLSTMT is to create and write out to SYSPCH the Linkage Editor control statements necessary to produce the overlay program that is executed in the Storage step. Also, since every ESD card (except the last one) is destroyed during the ESDPROC processing, CNTLSTMT must reconstruct these records and write them out to the SYSPCH file. CNTLSTMT creates the control statements and reconstructs the ESD card images in the I/O buffer that contains the last ESD card. Thus, the last ESD card is destroyed (but later reconstructed), whereas the first TXT card remains unaffected in the other I/O buffer. CNTLSTMT employs the put out a card routine (PUTCARD) for all of its writing operations.

The first six statements created by CNTLSTMT are PHASE and INCLUDE statements for the object modules IJLFST, IJLFLOAD, and IJLFUPDT. The PHASE statements provide the Linkage Editor with phase names and origin points for the phases. The INCLUDE statements indicate to the Linkage Editor which object modules are to

Control Initial	(CNTLINIT)
ESD Processor	(ESDPROC)
Create Control Statements	(CNTLSTMT)
TXT Processor	(TXTPROC)
Get and Check an Input Record	(GETCHECK)
Put Out a Card	(PUTCARD)
Message Fan-in	(MSGFAN)
Diagnostic Writer	(DIAGWTR)
Control End	(CNTLEND)
I/O Buffer 1	(OBJCARD1)
I/O Buffer 2	(OBJCARD2)
SYSIPT DTFDI Macro	(IJSYSIN)
SYSPCH DTFDI Macro	(IJSYSPH)
SYSLST DTFDI Macro	(IJSYSLS)
DIMOD	(IJJFCBIC)

Figure 3-21. Module IJLFCT Physical Organization

be included for editing. The formats of the statements are as follows:

```

PHASE IJLFST,S
INCLUDE IJLFST
PHASE IJLFLOAD, S+144
INCLUDE IJLFLOAD
PHASE IJLFUPDT, S+144
INCLUDE IJLFUPDT

```

Next, CNTLSTMT creates and writes out a series of PHASE and INCLUDE statements for the input FD program CSECTs. One PHASE statement and one INCLUDE statement is generated for each input CSECT. The format of these statements is as follows:

```

PHASE IJLFnnnn, +0
INCLUDE , (IJLFnnnn)

```

where nnnn is the number of an input CSECT that will become an overlay segment.

When all the necessary Linkage Editor control statements have been created and written out, CNTLSTMT begins reconstructing the previously destroyed ESD card images and writing them out to SYSPCH. Since all of the ESD items were verified by ESDPROC, all of the data within those items (except the length field of the last ESD item) is predictable and can be reconstructed. Also, since ESDPROC maintained a count of the ESD items, CNTLSTMT can reconstruct the exact number of ESD items required. As soon as CNTLSTMT constructs an ESD card in the I/O buffer, it writes it out to SYSPCH, placing it behind the control statements, then reconstructs the next ESD card in the same I/O buffer. When the last ESD card has been reconstructed and written out, CNTLSTMT passes control to the TXT processor routine.

*TXTPROC - TXT Processor:* The TXT processor routine receives control from the create control statements routine.

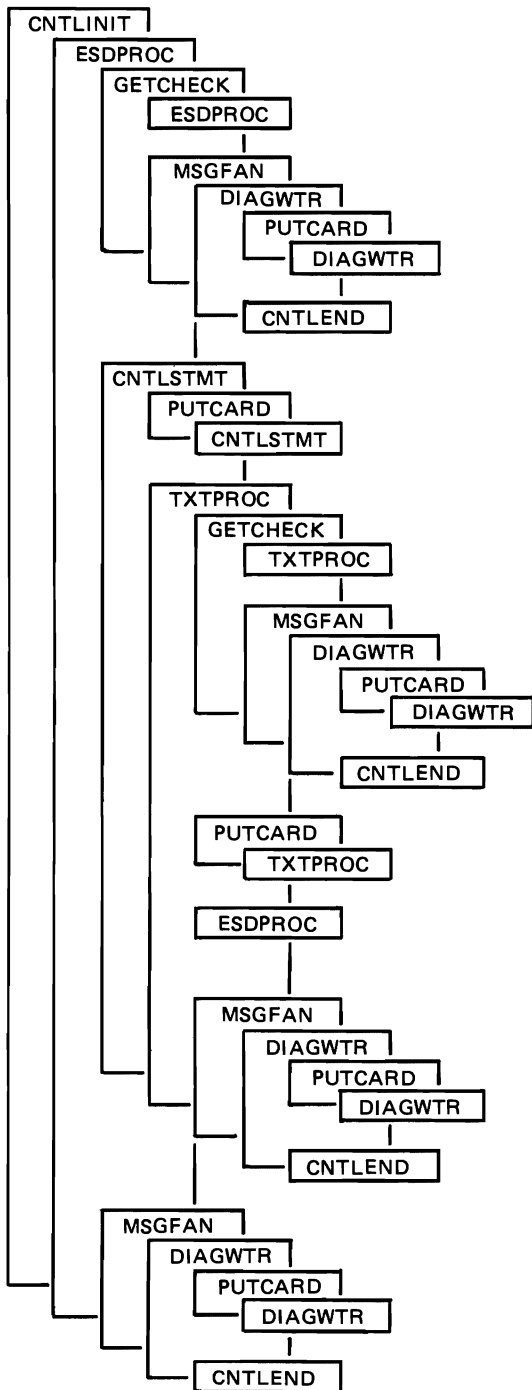


Figure 3-22. Module IJLFCT Hierarchy of Routines

By the time it receives control, the first TXT card has already been placed in an I/O buffer. As previously explained, this card is unaffected by the operation of CNTLSTMT.

TXTPROC invokes PUTCARD to put out that first card, then alternately invokes GETCHECK and PUTCARD to read in and put out the remaining TXT cards. TXTPROC performs this reading and writing function until it encoun-

ters an END card. At that point, it puts out the END card and reads the next record. If the next record read is the end-of-file indication, the routine passes control to the message fan-in routine for the construction of a message indicating successful completion. However, if the next record is an ESD card, then multiple input object decks are present. In this case, TXTPROC sets the multiple object deck indicator (NEWJOB) to one and the address counter (ADDRCTR) to zero, then passes control to the ESD processor. On the other hand, if the next record read is neither an end-of-file indication nor an ESD card, then an error condition exists. This error is detectable by GETCHECK. If it occurs, TXTPROC does not regain control from GETCHECK. Rather, GETCHECK passes control to the appropriate point in the message fan-in routine so that a descriptive error message may be written out to SYSLST.

**GETCHECK - Get and Check an Input Record:** The get and check an input record routine is invoked by ESDPROC and TXTPROC to read in the next sequential record from SYSIPT and to check three fields of the record. The three fields checked are deck ID, columns 73-76; card sequence number, columns 77-80; and card type, columns 2-4. GETCHECK validates the deck ID field to ensure that the four-character deck identifier is the same on every card of the input object deck. It validates the card sequence number field to ensure that no input records are out of place or missing. Finally, GETCHECK checks the card type field to ensure that the card types themselves are in the correct sequence. That is, an ESD card must be followed by another ESD card or a TXT card; a TXT card must be followed by another TXT card or an END card; an END card must be followed by an ESD card (indicating that another FD program module follows) or an end-of-file indication. If no errors are detected in the three fields, GETCHECK returns control to the routine that called it. However, if GETCHECK does detect an error, it ceases processing immediately and passes control to the appropriate point in the message fan-in routine so that a descriptive error message may be written out to SYSLST. If an end-of-file indication is encountered during the reading of SYSIPT, GETCHECK does not retain control; this condition is handled by the end-of-file subroutine of TXTPROC. GETCHECK uses the Device Independent (DI) access method for all data retrieval.

**Note:** GETCHECK can process both 80- and 81- character input records. When it discovers that an 81-character record has been read, it deletes the control character and processes the record as a standard 80-character record. Through this capability, the Control step of the FD utility can receive inputs from tape and disk, as well as from a card reader.

**PUTCARD - Put Out a Card:** The put out a card routine is invoked by CNTLSTMT and TXTPROC to write out a card

image to the SYSPCH file. It may also be invoked by the diagnostic writer routine (DIAGWTR) to write out a record to the SYSLST file. After the card image or record has been put out successfully, PUTCARD returns control to the routine that called it. PUTCARD uses the DI access method for all data writing.

**MSGFAN - Message Fan-In:** The message fan-in routine may be invoked by ESDPROC, GETCHECK, and CNTLEND. MSGFAN consists of a series of branch and link (BAL) instructions, each of which corresponds to a message contained in the message module, IJLFM01. The routine may be entered at any one of these BAL instructions. The purpose of MSGFAN is to establish a binary number and place it in register 3. This binary number equals the displacement between the start of MSGFAN and the point at which the entry to the routine is made. The number is used later by the diagnostic writer routine for scanning an index in IJLFM01 to retrieve the variable detail line(s) of the message. When the binary number has been established in register 3, the routine transfers control to the diagnostic writer routine.

**DIAGWTR - Diagnostic Writer:** The diagnostic writer routine is invoked by the message fan-in routine when all of the input record processing has been completed successfully or when an error has been detected in an input record. First, DIAGWTR issues a LOAD macro to bring the message module, IJLFM01, into main storage. Next, the routine moves the three variable characters of the message ID into the output area. DIAGWTR then uses the binary number located in register 3, which is the number placed there by MSGFAN, to recover the detail line (s) of the message. Finally, the routine composes the rest of the header line, including the remainder of the message ID, then writes the header and detail lines out to the SYSLST file. At this point, DIAGWTR passes control to the control end routine.

**CNTLEND - Control End:** The control end routine is invoked by the diagnostic writer routine. It may be called when all of the input record processing has been completed successfully or when an error has been discovered in an input record. The purpose of CNTLEND is to close the SYSIPT, SYSPCH, and SYSLST files. When these files have been closed, the routine gives control to the operating system.

**SYSIPT DTFDI Macro:** The DTFDI macro for the system input file, SYSIPT, specifies the following operands:

DEVADDR=SYSIPT	MODNAME=IJJFCBIC
EOFADDR=EOFRTN	RECSIZE=80
IOAREA1=INAREA1	WLRERR=WLREC
IOAREA2=INAREA2	SEPASMB=NO
IOREG=(9)	ERROPT=IGNORE

**SYSPCH DTFDI Macro:** The DTFDI macro for the system output file, SYSPCH, specifies the following operands:

```
DEVADDR=SYSPCH
IOAREA1=WKAREA
MODNAME=IJJFCBIC
ERROPT=IGNORE
RECSIZE=81
SEPASMB=NO
```

**SYSLST DTFDI Macro:** The DTFDI macro for the system print file, SYSLST, specifies the following operands:

```
DEVADDR=SYSLST
IOAREA1=PRNTAREA
MODNAME=IJJFCBIC
RECSIZE=81
SEPASMB=NO
ERROPT=IGNORE
```

**DIMOD:** The characteristics of the device independent module (DIMOD) are specified through the use of the DIMOD macro. The name of this module within IJLFCT is IJJFCBIC. Its characteristics are defined as follows:

```
IOAREA2=YES
SEPASMB=NO
TYPEFILE=OUTPUT
```

Additional information about the DTFDI and DIMOD macros is contained in the manual *IBM System/360 Disk Operating System Supervisor and Input/Output Macros*, Order Number GC24-5037.

### Linkage Editor Step Organization

The OS and DOS versions of the FD utility employ the OS and DOS Linkage Editor programs in their respective Linkage Editor steps. The inputs and outputs from the Linkage Editor are described in Part 3, Section 2. "Method of Operation." A description of the OS and DOS Linkage Editors' organization is beyond the scope of this manual. However, complete Linkage Editor information may be found in the following publications:

- OS Version: *IBM System/360 Operating System Linkage Editor and Loader*, Order Number GC28-6538; *IBM System/360 Operating System Linkage Editor [E] Program Logic Manual*, Order Number GY28-6610
- DOS Version: *IBM System/360 Disk Operating System: System Control and System Service Programs*, Order Number GC24-5036; *IBM System/360 Disk Operating System Linkage Editor Program Logic Manual*, Order Number GY24-5080

## Storage Step Organization

The overlay program that is executed in the Storage step is created by the Linkage Editor in the previous step. This program consists of the linkage edited object module IDFST (OS) or IJLFST, IJLFLOAD, and IJLFUPDT (DOS) and the overlay segments containing the FD programs. The organization of the modules is dissimilar enough to merit a separate discussion of each. These modules are described in the following paragraphs.

### *IDFST Organization*

The load module IDFST is logically organized in nine routines, two DCBs, and two Linkage-Editor-created tables. Furthermore, if the PARM facility of JCL is employed, IDFST may construct a replacement table. Since the generation and composition of the two Linkage-Editor-created tables were discussed in Part 3, Section 2, "Method of Operation," no more discussion of these tables is contained in this section. Figure 3-23 represents the physical organization of IDFST. The logical flow of the nine routines is graphically represented in Charts OB1 through OB4 at the rear of this section. Figure 3-24 illustrates the hierarchy of the routines within IDFST.

*STGINIT - Storage Initial:* The storage initial routine opens and tests the opening of the SYSLIB (user's library) and SYSPRINT files. Control passes automatically to the PARM processor routine unless one of the files fails to open properly. If the opening of SYSLIB fails, STGINIT gives control to the appropriate point in the message fan-in routine (MSGFAN) so that a descriptive error message may be written out to SYSPRINT. If the opening of SYSPRINT fails, STGINIT writes a message out to the operator console, then transfers control to the storage end routine.

*PARMPROC - PARM Processor:* The PARM processor routine receives control from STGINIT at the start of processing. The purpose of PARMPROC is to validate the parameters that were passed from the JCL and to set an indicator (PARMSW) to tell the new member routine (NEWMEM) whether a replacement table should be constructed. If any of the parameters are invalid, PARMPROC ceases processing immediately and gives control to the appropriate point in MSGFAN so that a descriptive error message may be written out to SYSPRINT. If the user has elected not to use the parameter feature, PARMPROC leaves PARMSW set to zero (the value at which it was initialized) and passes control to NEWMEM. If the user has specified REPLACE=ALL, PARMPROC sets PARMSW to one, then passes control to NEWMEM. If the user has specified a partial replacement, PARMPROC sets PARMSW to two, then passes control to NEWMEM. Only this value of PARMSW (two) causes NEWMEM to create a replacement table if it encounters any duplicate FD programs. A duplicate FD program is one that bears the name of an FD program that is already present in the user's library.

*NEWMEM - New Member:* The new member routine receives control from the PARM processor routine. The purpose of NEWMEM is to create the new library members and place them in the user's FD program library. NEWMEM first invokes the segment processor routine (SEGPROC) to load the first overlay segment, IDF1000, into main storage. It then examines the count subfield of the KUPBs within the segment and writes the 476-byte unpacked program block field out to the user's library. When all of the KUPBs in the segment have been examined and selectively written out, NEWMEM again invokes SEGPROC to load the next sequential overlay segment. The routine continues this loading, examining, and writing process until it encounters the end-of-form indication. At that point, NEWMEM stores the last unpacked program block of the FD program, then issues the appropriate form of the STOW macro to assign the new member's name.

NEWMEM's next action depends upon the return code from the STOW operation. If the STOW was successful, NEWMEM begins processing the next FD program. However, if the STOW was not successful because the FD program was a duplicate, NEWMEM may perform one of the following functions:

- If the user has specified REPLACE, NEWMEM replaces the old member with the new member.
- If the user has specified a partial replacement, NEWMEM creates a replacement table from the input parameter list, searches the table for the name of the new member, and replaces the old member with the new member if the name is found in the replacement table. If the name is not found in the replacement table, NEWMEM stores the new member under a temporary name, if one is available, or does not store it at all.
- If the user has not specified any replacement parameters, NEWMEM stores the new member under a temporary name, if one is available, or does not store it at all.

If the STOW was unsuccessful because of a stow error or a DASD space error, NEWMEM passes control to the appropriate point in the message fan-in routine so that a descriptive error message may be written out to SYSPRINT.

NEWMEM resumes processing with the next KUPB in the segment, if one is available, or it invokes SEGPROC to load the next segment. The next KUPB encountered is the first KUPB of the next FD program. The routine processes the new FD program and all other new FD programs in this manner until it encounters the end-of-assembly indicator. It then passes control to the diagnostic writer routine (DIAGWTR). If NEWMEM discovers an error in the count subfield of a KUPB, it ceases processing immediately and gives control to the appropriate point in MSGFAN so that a descriptive error message may be written out to SYSPRINT.

*SEGPROC - Segment Processor:* The segment processor routine is invoked by the new member routine to load an

Segment Table*	(SEGTAB)
Storage Initial	(STGINIT)
PARM Processor	(PARMPROC)
New Member	(NEWMEM)
Message Fan-in	(MSGFAN)
Segment Processor	(SEGPROC)
Diagnostic Writer	(DIAGWTR)
Storage End	(STGEND)
Control Synchronous I/O Error	(CTLSYNER)
Write Operator	(WTORTN)
SYSLIB DCB	(SYSLIB)
SYSPRINT DCB	(PRINT)
Entry Table*	(ENTAB)

Replacement Table**	(REPLTAB)
---------------------	-----------

\*Only present after the execution of the Linkage Editor step.

\*\*Built during the execution of the Storage Step.

Figure 3-23. Module IDFST Physical Organization

overlay segment into main storage. SEGPROC first sets the status indicator in the SEGTAB entry of the previously called segment to 11, increments the “to” segment number in ENTAB by one, and sets the “previous caller” field in ENTAB to zero. It then issues a SEGWT supervisor call (SVC 37) to read in the segment. When the input operation is complete, SEGPROC returns control to NEWMEM. If a request is made for a segment that exceeds the range of the available segments, SEGPROC passes control to the appropriate point in MSGFAN so that a descriptive error message may be written out to SYSPRINT.

*SYNAD1 and SYNAD2 - Control Synchronous I/O Error:* One of the control synchronous I/O error routines is invoked when an irrecoverable error is encountered during an input or output operation. These routines actually correspond to the two I/O files used by the Storage step. SYNAD1 corresponds to SYSPRINT; SYNAD2 corresponds to SYSLIB. The purpose of these routines is to determine the cause and type of error that occurred and to create an error message that explains the problem. The routines use the SYNADAF macro to define the error and construct the error message. This macro examines the following data

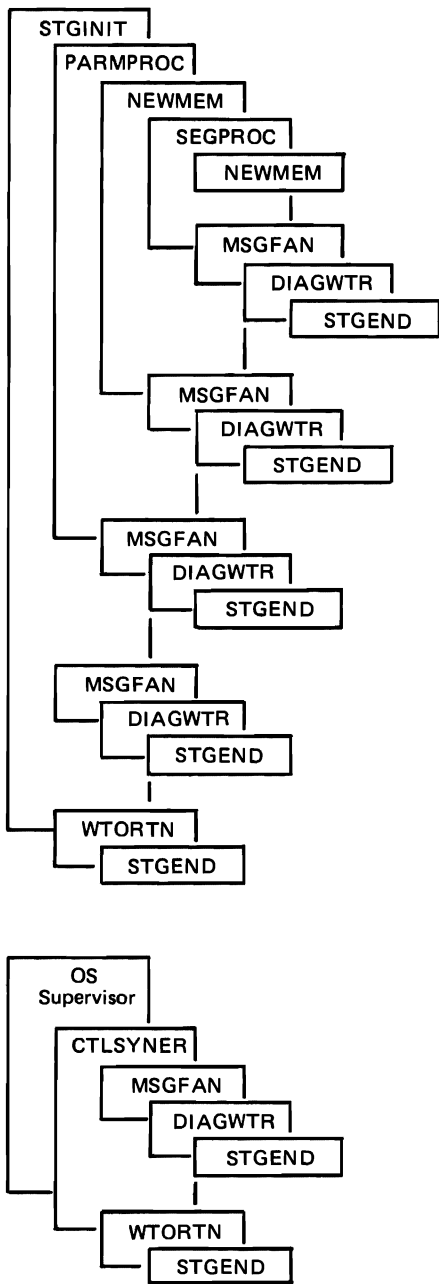


Figure 3-24. Module IDFSH Hierarchy of Routines

areas and records the pertinent information.

- The contents of the general registers
- The data event control block (DECB)
- The exceptional condition code
- The status and sense indications

After examining this data, the macro creates a descriptive error message that may be written out by either the diagnostic writer routine for a SYSLIB error or by SYNAD1 itself for a SYSPRINT error. After the message has been issued, control is passed to the storage end routine.

**MSGFAN - Message Fan-In:** The message fan-in routine may be invoked by any of the following routines: STGINIT, PARMPROC, NEWMEM, SEGPROC, SYNAD2, and STGEN. MSGFAN consists of a series of branch and link (BAL) instructions, each of which corresponds to a message contained in the message module, IDFM01. The routine may be entered at any one of these BAL instructions. The purpose of MSGFAN is to establish a binary number and place it in register 3. This binary number equals the displacement between the start of MSGFAN and the point at which the entry to the routine was made. The number is used later by DIAGWTR for scanning a second-level index in IDFM01 to retrieve the variable detail line(s) of the message. When the binary number has been established in register 3, the routine transfers control to DIAGWTR.

**DIAGWTR - Diagnostic Writer:** The diagnostic writer routine is invoked by the message fan-in routine when all of the overlay segment processing is complete or when an error condition is detected and the SYSPRINT file can be used. First, DIAGWTR issues a LOAD macro to bring the message module, IDFM01, into main storage. Next, the routine moves the three variable characters of the message ID into the output area, OUTPUT. DIAGWTR then uses the binary number located in register 3, which is the number placed there by MSGFAN, to recover the detail line(s) of the message. Finally, the routine composes the rest of the header line, including the remainder of the message ID, then writes the header and detail lines out to the SYSPRINT file. At this point, DIAGWTR checks to see if processing should continue. If so, it passes control to the proper routine for continued processing; if not, DIAGWTR passes control to the storage end routine.

**STGEN - Storage End:** The storage end routine is invoked by either DIAGWTR or WTORTN. It may be called when all of the overlay segment processing has been completed successfully or when any type of error condition has been encountered. The purpose of STGEN is to close the SYSLIB and SYSPRINT files. When these files have been closed, the routine gives control to the operating system.

**Output DCB:** The OUTPUT data control block describes the characteristics of the SYSLIB file as follows:

```

DDNAME=SYSLIB
LRECL=476
BLKSIZE=476
MACRF=(W)
DSORG=PO
SYNAD=SYNAD2
RECFM=F
  
```

**PRINT DCB:** The PRINT data control block describes the characteristics of the SYSPRINT file as follows:

```

DDNAME=SYSPRINT
MACRF=(PM)
  
```



DSORG=PS  
SYNAD=SYNAD1

**REPLTAB - Replacement Table:** The replacement table is constructed by the PARM processor routine if the user has employed the JCL PARM feature. The format of REPLTAB is illustrated in Figure 3-25.

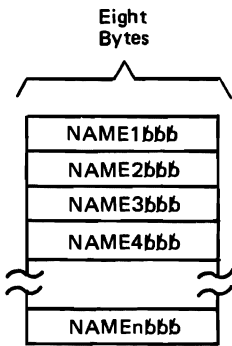


Figure 3-25. OS REPLTAB Format

#### IJLFST Organization

The phase IJLFST is logically organized in seven routines, two DTFDI macros, and one device independent module (DIMOD). Figure 3-26 represents the physical organization of IJLFST. The logical flow of the seven routines is graphically represented in Charts DB1 and DB2 at the rear of this section. Figure 3-27 illustrates the hierarchy of the routines within IJLFST. The routines, macros, and the DIMOD are described in the following paragraphs.

**OPEN - Open Files:** The open files routine opens the SYSIPT and SYSLST files. Control passes automatically to the get and process control cards routine unless one of the files fails to open. If one or both of the files fails to open, the DOS Supervisor terminates the Storage step and returns control to the DOS Job Control program.

**GET - Get and Process Control Cards:** The get and process control cards routine receives control from OPEN at the start of processing. First, it invokes GETCARD to read in and validate a control card. It then determines which operand the user has specified (DEVICE=, OPTION=, RPLACE=, or RPLACE). After determining the operand, GET validates the operand type to ensure that the control card has been coded correctly.

If the operand is DEVICE=, the operand type must be either 2311 or 2314; if neither of these is specified, GET passes control to the error exit routine, ERREXT. If either 2311 or 2314 is specified, GET sets the indicator in the first half-word of the replacement table, RPLTAB, to the appropriate value, then reads in another card for processing.

If the operand is OPTION=, the operand type must be UPDATE, LOAD, or LOADFST; if none of these is specified, GET passes control to ERREXT. If a valid operand

type is specified, GET sets the indicator in the second half-word of RPLTAB to the appropriate value, then reads in another card for processing.

If the operand is RPLACE=, GET increments the replace card counter by one and checks to see whether the total of replace cards has exceeded 20. If the total has exceeded 20, GET passes control to ERREXT; if not, it saves the specified name in the next available eight-byte location in RPLTAB, then reads in another card for processing.

If the operand is RPLACE, GET inserts a X'FF' in the location RPLTAB+4 to indicate that the REPLACE ALL function was specified. It then reads in another card for processing. GET continues reading and processing the control cards until it encounters an end-of-file indication, at which time control is passed to the end-of-file routine, EOFRTN.

**GETCARD - Get and Validate a Control Card:** The get and validate a control card routine is invoked by GET to read in a card from SYSIPT and validate it. The validation consists of checking the first three columns for “//” and ensuring that a valid operand is present. If the card is improperly coded, GETCARD passes control to ERREXT. Otherwise, it returns control to GET.

**ERREXT - Common Error Exit:** The common error exit routine may be invoked by either GET or GETCARD. It is invoked if any field of a control card is invalid. ERREXT first sets the error flag and prints out the message “INCORRECT - JOB TERMINATED,” then closes the SYSLST file and passes control to the flush cards routine, FLSHCARD.

**ERREXT1 - Special Error Exit:** The special error exit routine is invoked by GET if GET detects more than 20 control cards with the operand “RPLACE=”. ERREXT1 first sets the error flag and prints out the message “NUMBER OF REPLACE CARDS EXCEEDS TWENTY - JOB TERMINATED,” then closes the SYSLST file and passes control to FLSHCARD.

**FLSHCARD - Flush Cards:** The flush cards routine receives control from ERREXT or ERREXT1 when an error has been encountered. Its function is to read in cards from the SYSIPT file until the end-of-file indication is encountered. At that point, the DOS Supervisor passes control to EOFRTN.

**EOFRTN - End-of-File:** The end-of-file routine is invoked by the DOS Supervisor when the end-of-file indication is encountered on the SYSIPT file. When EOFRTN first gains control, it checks the error flag to see if an error has occurred. If so, it closes the SYSIPT file and exits to the operating system. If not, EOFRTN saves RPLTAB at the end of the user's partition and closes the SYSIPT and SYSLST files. It then checks to see which option the user has specified. If the user has specified LOADFST or UPDATE, EOFRTN fetches and passes control to

Open Files	(OPEN)
Get and Process Control Cards	(GET)
End-of-File	(EOFRTN)
Common Error Exit	(ERREXT)
Flush Cards	(FLSHCARD)
Special Error Exit	(ERREXT1)
Get and Validate a Control Card	(GETCARD)
SYSIPT DTFDI Macro	(CARDTF)
SYSLST DTFDI Macro	(PRNTDTF)
DIMOD	(INMOD)

Replacement Table*	(RPLTAB)
--------------------	----------

\*Built during the execution of IJLFST

Figure 3-26. Module IJLFST Physical Organization

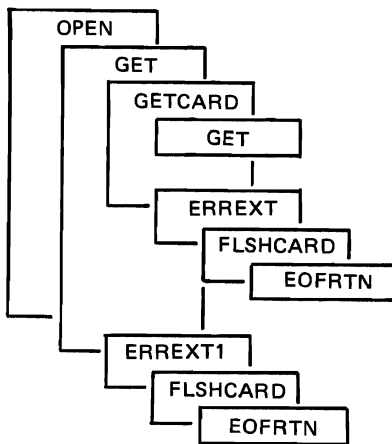


Figure 3-27. Module IJLFST Hierarchy of Routines

**IJLFUPDT.** If the user has specified **LOAD**, **EOFRTN** fetches and passes control to **IJLFLOAD**.

**SYSIPT DTFDI Macro:** The **DTFDI** macro for the system input file, **SYSIPT**, specifies the following operands:

```
DEVADDR=SYSIPT
EOFADDR=EOFRTN
IOAREA1=RDAREA
MODNAME=INMOD
RECSIZE=80
ERROPT=IGNORE
SEPASMB=NO
```

**SYSLST DTFDI Macro:** The **DTFDI** macro for the system print file, **SYSLST**, specifies the following operands:

```
DEVADDR=SYSLST
IOAREA1=ERRMSG
MODNAME=INMOD
RECSIZE=121
SEPASMB=NO
ERROPT=IGNORE
```

**DIMOD:** The characteristics of the device independent module (**DIMOD**) are specified through the use of the **DIMOD** macro. The name of this module within **IJLFST** is **INMOD**. Its characteristics are defined as follows:

```
SEPASMB=NO
TYPEFLE=OUTPUT
```

Additional information about the **DTFDI** and **DIMOD** macros is contained in the manual *IBM System/360 Disk Operating System Supervisor and Input/Output Macros*, Order Number GC24-5037.

### **IJLFLOAD Organization**

The phase **IJLFLOAD** is logically organized in seven routines, one **DTFDI** macro, one **DTFIS** macro, one device independent module (**DIMOD**), and one **ISAM** module (**ISMOD**). Figure 3-28 represents the physical organization of **IJLFLOAD**. The logical flow of the seven routines is graphically represented in charts **DC1** and **DC2** at the rear of this section. Figure 3-29 represents the hierarchy of the routines within **IJLFLOAD**. The routines, macros, and modules are described in the following paragraphs.

**CALLOAD - Load Initial:** The load initial routine opens the **IJFDLIB** (user's library) and **SYSLST** files. If one or both of the files fails to open properly, the **DOS Supervisor** terminates **IJLFLOAD** and returns control to the **DOS Job Control** program. However, if the files do open properly, **CALLOAD** invokes the load overlay phases routine, **LOADRTN**, to load the first overlay phase. It saves the name and count subfields of the first **KUPB** in the phase, then transfers control to the validate records routine, **RCDCHK**.

**RCDCHK - Validate Records:** The validate records routine receives control from **CALLOAD** after the first overlay phase has been loaded. **RCDCHK** validates the name and count subfields of the first **KUPB** in the overlay phase, then invokes the **ISAM** put routine, **ISMPUT**, to write out the 476-byte unpacked program block field to the user's library, together with the 10-byte key. It processes the second and third **KUPBs** in the same manner. When the three **KUPBs** in an overlay phase have been processed, **RCDCHK** invokes **LOADRTN** to load the next sequential phase. **RCDCHK** continues this reading, validating, and writing process until it encounters an end-of-assembly indication. At that point, it transfers control to the message fan-in routine, **MSGFAN**, to write out a message indicating the successful completion of **IJLFLOAD**. If **RCDCHK** discovers an error in the name or count subfield of any **KUPB**, it transfers control to the appropriate point in **MSGFAN** so that a descriptive error message may be written out.

**LOADRTN - Load Overlay Phases:** The load overlay phases routine is invoked by **CALLOAD** and **RCDCHK** to load an overlay phase into main storage from the core image library. **LOADRTN** uses the **LOAD** macro to read in the phase. When the input operation is complete, **LOADRTN** returns control to the calling routine.

**ISMPUT - ISAM Put:** The **ISAM** put routine is invoked by **RCDCHK** to put out a 476-byte unpacked program block and a 10-byte key to the user's library, **IJFDLIB**. Before **ISMPUT** writes out any records, it initializes the data set index areas by issuing a **SETFL** macro. It then attempts to write out the record. If the attempt is successful, **ISMPUT** returns control to **RCDCHK**. If the attempt is unsuccessful because of a **DASD** error, wrong length record error, or undetermined error, **ISMPUT** again attempts to write out the record. If this attempt is successful, the routine returns control to **RCDCHK**; if it is unsuccessful, **ISMPUT** passes control to the appropriate point in **MSGFAN** so that a descriptive error message may be written out. Similarly, if the original attempt to write out the record is unsuccessful for any reason other than those stated above **ISMPUT** passes control to **MSGFAN**.

**MSGFAN - Message Fan-In:** The message fan-in routine may be invoked by **RCDCHK** or **ISMPUT**. **MSGFAN** consists of a series of branch and link (**BAL**) instructions, each of which corresponds to a message contained in the message module, **IJLFM01**. The routine may be entered at any one of these **BAL** instructions. The purpose of **MSGFAN** is to establish a binary number and place it in register 3. This binary number equals the displacement between the start of **MSGFAN** and the point at which the entry to the routine is made. The number is used later by the diagnostic writer routine for scanning an index in **IJLFM01** to retrieve the required message. When the binary number has been established in register 3, the routine transfers control to the diagnostic writer routine.

Load Initial	(CALLOAD)
Validate Records	(RCDCHK)
End-of-Job	(EOJRTN)
Message Fan-in	(MSGFAN)
Diagnostic Writer	(DIAGWTR)
Load Overlay Phases	(LOADRTN)
ISAM Put	(ISMPUT)
IJFDLIB DTFIS Macro	(IJFDLIB)
SYSLST DTFDI Macro	(PRNTDTF)
ISMOD	(LOADMOD)
DIMOD	(OUTMOD)

Figure 3-28. Module IJLFLOAD Physical Organization

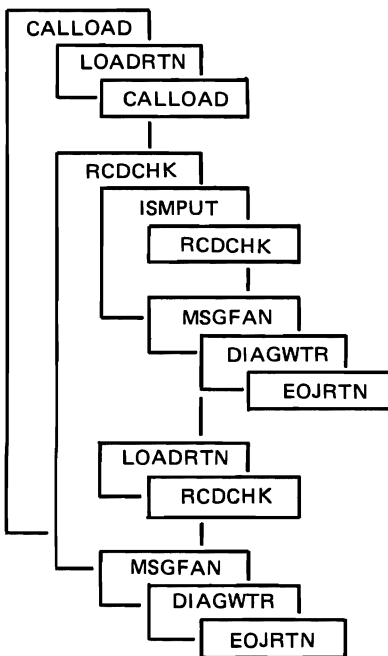


Figure 3-29. Module IJLFLOAD Hierarchy of Routines

*DIAGWTR - Diagnostic Writer:* The diagnostic writer routine is invoked by the message fan-in routine when all of the overlay phase processing has been completed successfully or when an error has been detected by RCDCHK or ISMPUT. First, DIAGWTR uses the value in register 3 to compute the proper offset in IJLFM01. Next, it issues a load macro to bring IJLFM01 into main storage, then moves the appropriate message into the output area. Finally, it writes the message out to the SYSLST file. When the output operation is complete, DIAGWTR passes control to the end-of-job routine, EOJRTN.

*EOJRTN - End-of-Job:* The end-of-job routine receives control from DIAGWTR. Its purpose is to close the IJFDLIB and SYSLST files. First, EOJRTN issues an ENDFL macro to write out an EOF indication to IJFDLIB. If this action causes the prime data area to overflow, EOJRTN transfers control to MSGFAN. If not, EOJRTN closes the two files and exits to the operating system.

*IJFDLIB DTFIS Macro:* The DTFIS macro for the user's library file, IJFDLIB, specifies the following operands:

DSKXTNT=3	DEVICE=2311
IOROUT=LOAD	ERREXT=YES
KEYLEN=10	HINDEX=2311
NRECD=1	IOAREAL=ISAMAREA
RECFORM=FIXUNB	MODNAME=LOADMOD
RECSIZE=476	WORKL=WKAREA
CYLOFL=8	

*SYSLST DTFDI Macro:* The DTFDI macro for the system print file, SYSLST, specifies the following operands:

DEVADDR=SYSLST  
IOAREA1=ERRMSG  
MODNAME=OUTMOD  
RECSIZE=121  
SEPASMB=NO  
ERROPT=IGNORE

*ISMOD:* The characteristics of the ISAM module (ISMOD) are specified through the use of the ISMOD macro. The name of this module within IJLFLOAD is LOADMOD. Its characteristics are defined as follows:

ERREXT=YES  
IOROUT=LOAD  
SEPASMB=NO

*DIMOD:* The characteristics of the device independent module (DIMOD) are specified through the use of the DIMOD macro. The name of this module within IJLFLOAD is OUTMOD. Its characteristics are defined as follows:

SEPASMB=NO  
TYPEFLE=OUTPUT

### *IJLFUPDT Organization*

The phase IJLFUPDT is logically organized in nine routines, one DTFIS macro, one DTFDI macro, one ISAM module (ISMOD), and one device independent module (DIMOD). Figure 3-30 represents the physical organization of IJLFUPDT. The logical flow of the nine routines is graphically represented in charts DD1 through DD4 at the rear of this section. Figure 3-31 represents the hierarchy of the routines within IJLFUPDT. The routines, macros, and modules are described in the following paragraphs. However, before the routines are entered, IJLFUPDT examines RPLTAB to determine which device type was specified. If the device type is 2311, control is passed immediately to the update initial routine, CALLOAD. However, if the device type is 2314, IJLFUPDT modifies the IJFDLIB DTFIS macro to indicate 2314. Control is then passed to CALLOAD.

*CALLOAD - Update Initial:* The update initial routine opens the IJFDLIB (user's library) and SYSLST files. If one or both of the files fails to open properly, the DOS Supervisor terminates IJLFUPDT and returns control to the DOS Job Control program. However, if the files do open properly, CALLOAD invokes the load overlay phases routine, LOADRTN, to load the first available overlay phase. It saves the name and count subfields of the first KUPB in the phase, then transfers control to the validate records routine, RCDCHK.

*RCDCHK - Validate Records:* The validate records routine receives control from CALLOAD after the first available overlay phase has been loaded. The purpose of RCDCHK is to create new library members and to place them in the user's FD program library. RCDCHK validates the name and count subfields of the first KUPB in the overlay phase, then invokes the ISAM put routine, ISMPUT, to write out the 476-byte unpacked program block and 10-byte key to the user's library. If the routine discovers an error in either of the subfields, it invokes MSGFAN to print out a descriptive error message, then passes control to the flush FD program routine, FLUSH. RCDCHK processes the second and third KUPBs in the same manner as the first. When the three KUPBs of an overlay phase have been processed, RCDCHK invokes LOADRTN to load the next available phase. It then processes the KUPBs within that phase. RCDCHK continues processing overlay phases in this manner until it reaches the end of the FD program. At that point, it prints out a message stating that the program has been stored, then resumes processing with the next FD program. When all of the FD programs have been processed, RCDCHK invokes MSGFAN to write out a message indicating the successful completion of the Storage step, then transfers control to the end-of-job routine, EOJRTN.

Update Initial	(CALLOAD)
Validate Records	(RCDCHK)
Flush FD Program	(FLUSH)
Load Overlay Phases	(LOADRTN)
ISAM Put	(ISMPUT)
End-of-Job	(EOJRTN)
Update Name	(UPDTNM)
Message Fan-in	(MSGFAN)
Diagnostic Writer	(DIAGWTR)
IJFDLIB DTFIS Macro	(IJFDLIB)
ISMOD	(UPDTMOD)
SYSLST DTFDI Macro	(PRNTDTF)
DIMOD	(OUTMOD)

Figure 3-30. Module IJLFUPDT Physical Organization

*LOADRTN - Load Overlay Phases:* The load overlay phases routine may be invoked by CALLOAD, RCDCHK, and FLUSH. The purpose of LOADRTN is to load an overlay phase into main storage. It issues a LOAD macro to bring in the phase, then increments the phase name operand that will be used to retrieve the next sequential overlay phase in the next pass through LOADRTN. Finally, it returns control to the calling program.

*ISMPUT - ISAM Put:* The ISAM put routine is invoked by RCDCHK to write out a keyed record on IJFDLIB. In addition to the initial output operation, ISMPUT contains four subroutines that are used to ensure that the record is written out properly. ISMPUT first moves the record to be stored into the output area. It writes out the record, waits for the output operation to end, then checks to see if the record was a duplicate. If the record was a duplicate, it transfers control to the DUPRTN subroutine; if not, it transfers control to the ERRRTN subroutine.

- ERRRTN - The purpose of ERRRTN is to ensure that the record was written out without error. First, it checks

to see if there was a DASD error or a wrong length record error. If there was an error, ERRRTN attempts the output operation again and checks the results. If the operation is successful this time, ERRRTN transfers control to the DATAOK subroutine; if not, it invokes MSGFAN to print out an error message, then transfers control to EOJRTN. If neither of the above mentioned errors has occurred, ERRRTN checks to see if an EOF was written out, if no output record was found, or if the overflow area was full. If one of these errors has occurred, ERRRTN invokes MSGFAN to write out an error message, then transfers control to EOJRTN. Otherwise, if ERRRTN does not detect any of these errors, it transfers control to the DATAOK subroutine.

- DATAOK - The DATAOK subroutine receives control from either ERRRTN or RPLCDUP. It checks to see if a temporary name was assigned to the FD program currently being processed and, if so, whether the user has been notified. If a temporary name has been assigned and the user has not been notified, DATAOK invokes

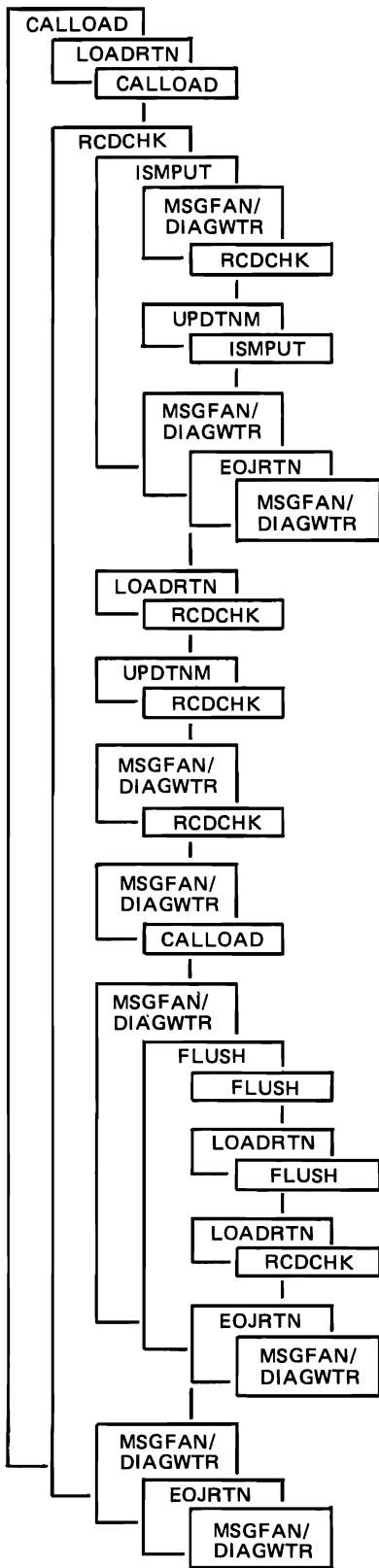


Figure 3-31. Module IJLFUPDT Hierarchy of Routines

MSGFAN to write out the information message, then returns control to RCDCHK. If no temporary name was assigned, DATAOK simply returns control to RCDCHK.

- **DUPRTN** - The DUPRTN subroutine receives control if the record that ISMPUT is trying to write out is part of a duplicate FD program. If the user has specified a LOADFST operation, or if he has specified RPLACE, or if the FD program's name is in the replacement table, DUPRTN transfers control to the RPLCDUP subroutine. If none of these conditions apply, DUPRTN prepares the record for storage under a temporary name, then transfers control to the start of ISMPUT so that the record may be stored. However, if no temporary names are available, DUPRTN prints out a message stating that fact and transfers control to FLUSH.
- **RPLCDUP** - The RPLCDUP subroutine receives control from DUPRTN when a duplicate record is to be replaced in IJFDLIB. It reads in the record to be replaced and writes out the new record in its place. It then checks to see if the record was written out properly. If it was not, RPLCDUP transfers control to ERRRTN. If the record was written out properly, RPLCDUP transfers control to DATAOK.

*UPDTNM - Update Name:* The update name routine may be invoked by RCDCHK and ISMPUT. The purpose of UPDTNM is to increment the temporary name number field by one. After it has accomplished this task, it returns control to the calling routine.

*FLUSH - Flush FD Program:* The flush FD program routine receives control from RCDCHK when that routine discovers an error in a KUPB. FLUSH simply reads in the erroneous FD program's KUPBs until it encounters the next FD program or the end-of-assembly indication. If it encounters the end-of-assembly indication, it transfers control to EOJRTN; otherwise, it transfers control to the label SAVENM1 in CALLOAD.

*MSGFAN - Message Fan-In:* The message fan-in routine may be invoked by RCDCHK, ISMPUT, and EOJRTN. MSGFAN consists of a series of branch and link (BAL) instructions, each of which corresponds to a message contained in the message module, IJLFM01. The routine may be entered at any one of these BAL instructions. The purpose of MSGFAN is to establish a binary number and to place it in register 3. This binary number equals the displacement between the start of MSGFAN and the point at which the entry to the routine is made. The number is used later by the diagnostic writer routine for scanning an index in IJLFM01 to retrieve the required message. When the binary number has been established in register 3, MSGFAN transfers control to the diagnostic writer routine.

**DIAGWTR - Diagnostic Writer:** The diagnostic writer routine is invoked by the message fan-in routine when all of the overlay phase processing has been completed successfully or when an error has been detected by RCDCHK or ISMPUT. First, DIAGWTR issues a LOAD macro to bring IJLFM01 into main storage. Next, it uses the value in register 3 to compute the proper offset in IJLFM01, then moves the message into the output area. If the message needs the FD program name or temporary name inserted into its text, DIAGWTR inserts it. It then writes out the message to the SYSLST file and returns control to the routine that invoked MSGFAN.

**EOJRTN - End-of-Job:** The end-of-job routine may be invoked by RCDCHK when all of the overlay phases have been processed correctly, or by ISMPUT if that routine encounters an error. EOJRTN prints out a message stating that the Storage step has been completed, then closes the IJFDLIB and SYSLST files. Finally, it exits to the operating system.

**IJFDLIB DTFIS Macro:** The DTFIS macro for the user's library file, IJFDLIB, specifies the following operands:

```

DSKXTNT=3           HINDEX=2311
IOROUT=ADDRTR      IOAREAL=RWKAREA
KEYLEN=10          IOAREAR=RWKAREA
NRECDs=1           IOSIZE=560
RECFORM=FIXUNB     KEYARG=KEYFLD
RECSIZE=476        MODNAME=UPDTMOD
CYLOFL=8           TYPEFLE=RANDOM
DEVICE=2311        WORKL=WKAREA
ERREXT=YES         WORKR=WKAREA

```

**SYSLST DTFDI Macro:** The DTFDI macro for the system print file, SYSLST, specifies the following operands:

```

DEVADDR=SYSLST
IOAREA1=ERRMSG
MODNAME=OUTMOD
RECSIZE=121
ERROPT=IGNORE

```

**ISMOD:** The characteristics of the ISAM module (ISMOD) are specified through the use of the ISMOD macro. The name of this module within IJLFUPDT is UPDTMOD. Its characteristics are defined as follows:

```

ERREXT=YES
IOROUT=ADDRTR
RECFORM=FIXUNB
TYPEFLE=RANDOM

```

**DIMOD:** The characteristics of the device independent module (DIMOD) are specified through the use of the DIMOD macro. The name of this module within IJLFUPDT is OUTMOD. Its characteristics are defined as follows:

```
TYPEFLE=OUTPUT
```

### Message Modules IDFM01 and IJLFM01 Organization

The message modules IDFM01 and IJLFM01 are organized alike. Figure 3-32 illustrates this organization. The message modules contain all of the text messages that are written out to a printer or print queue or to the operator console during the execution of the Control and Storage steps. (DOS messages are written out to the printer exclusively.) The messages are grouped this way to ease their translation and alteration. IDFM01 must reside in either the system link library or in a user-defined job/step library; IJLFM01 must reside in a core-image library.

When the need arises to write out a message during the execution of the Control or Storage step, the message module is loaded into main storage. The index is examined to determine the address of the appropriate detail message. The message is composed by the diagnostic writer routine of the step requiring the message. Certain fields within the message are added by the diagnostic writer, such as the number of CSECTs processed, the temporary name used to store an FD program or its real member (FD program) name, the deck ID, the card sequence number, etc. The message is then written out to the appropriate file.

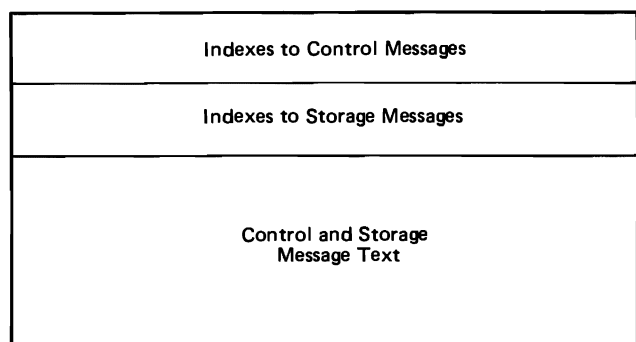


Figure 3-32. Modules IDFM01 and IJLFM01 Organization



# FD UTILITY FLOWCHARTS

## Chart OA1 OS FD UTILITY CONTROL STEP: IDFCT CHART 1 OF 5

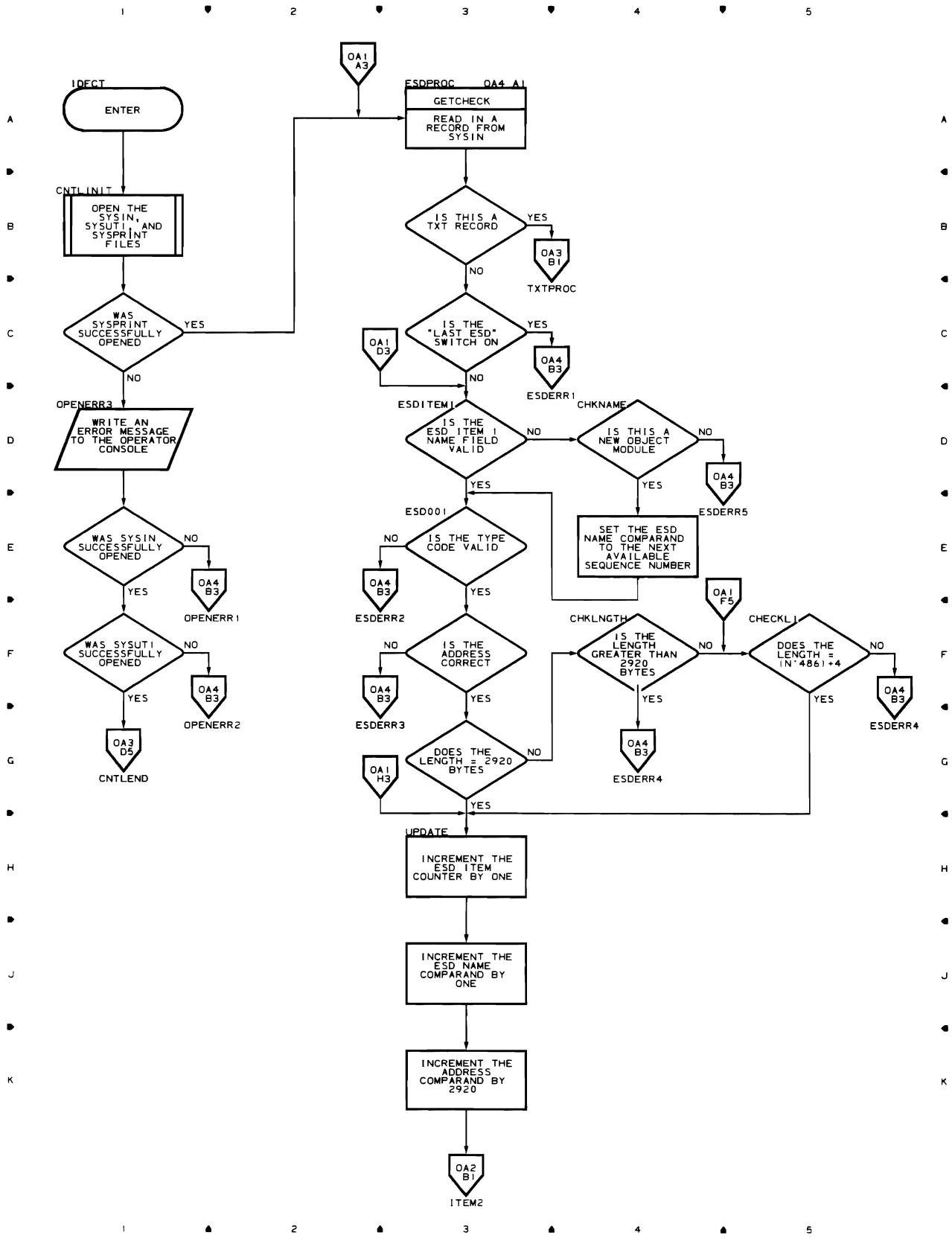


Chart OA2 OS FD UTILITY CONTROL STEP: IDFCT CHART 2 OF 5

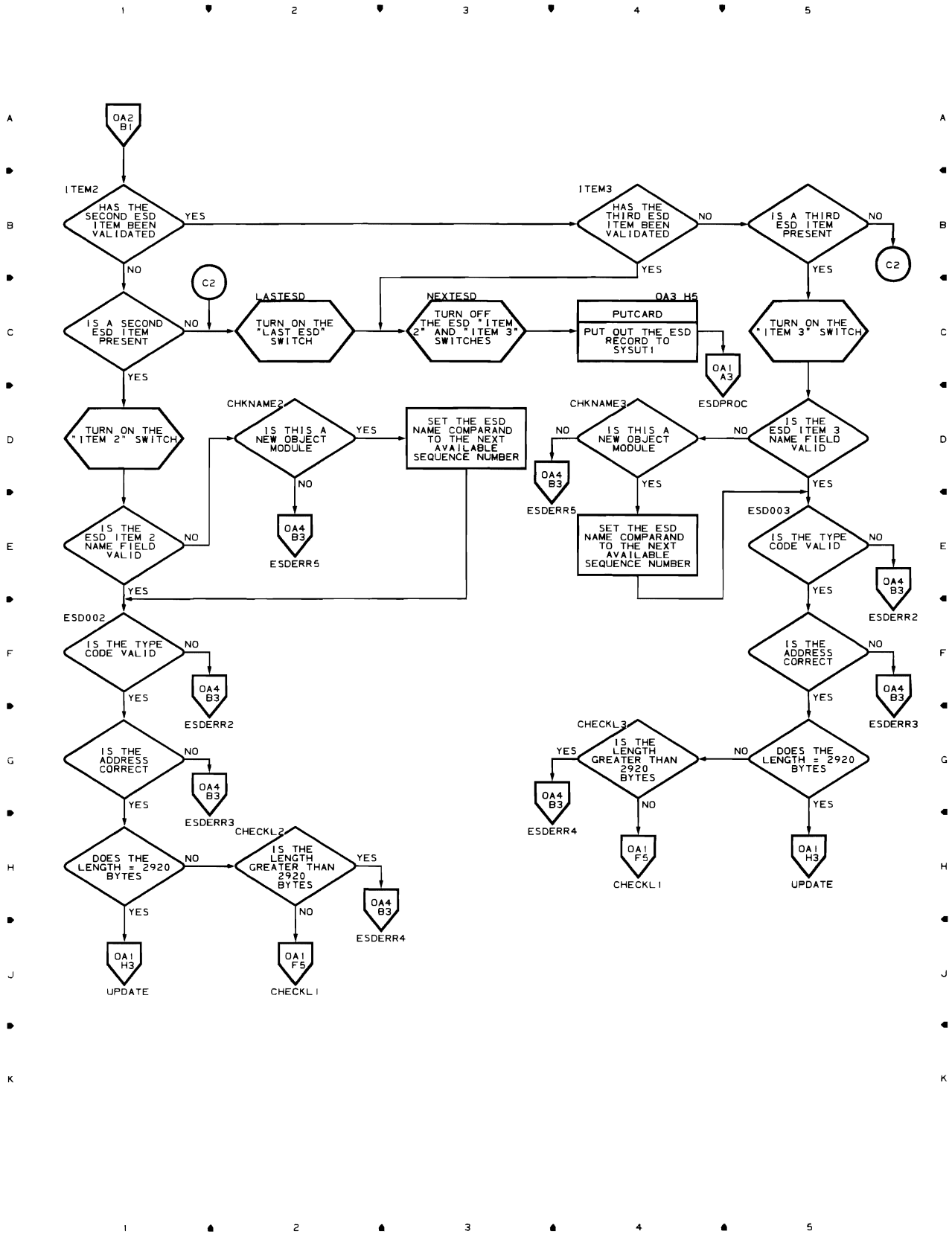


Chart OA3 OS FD UTILITY CONTROL STEP: IDFCT CHART 3 OF 5

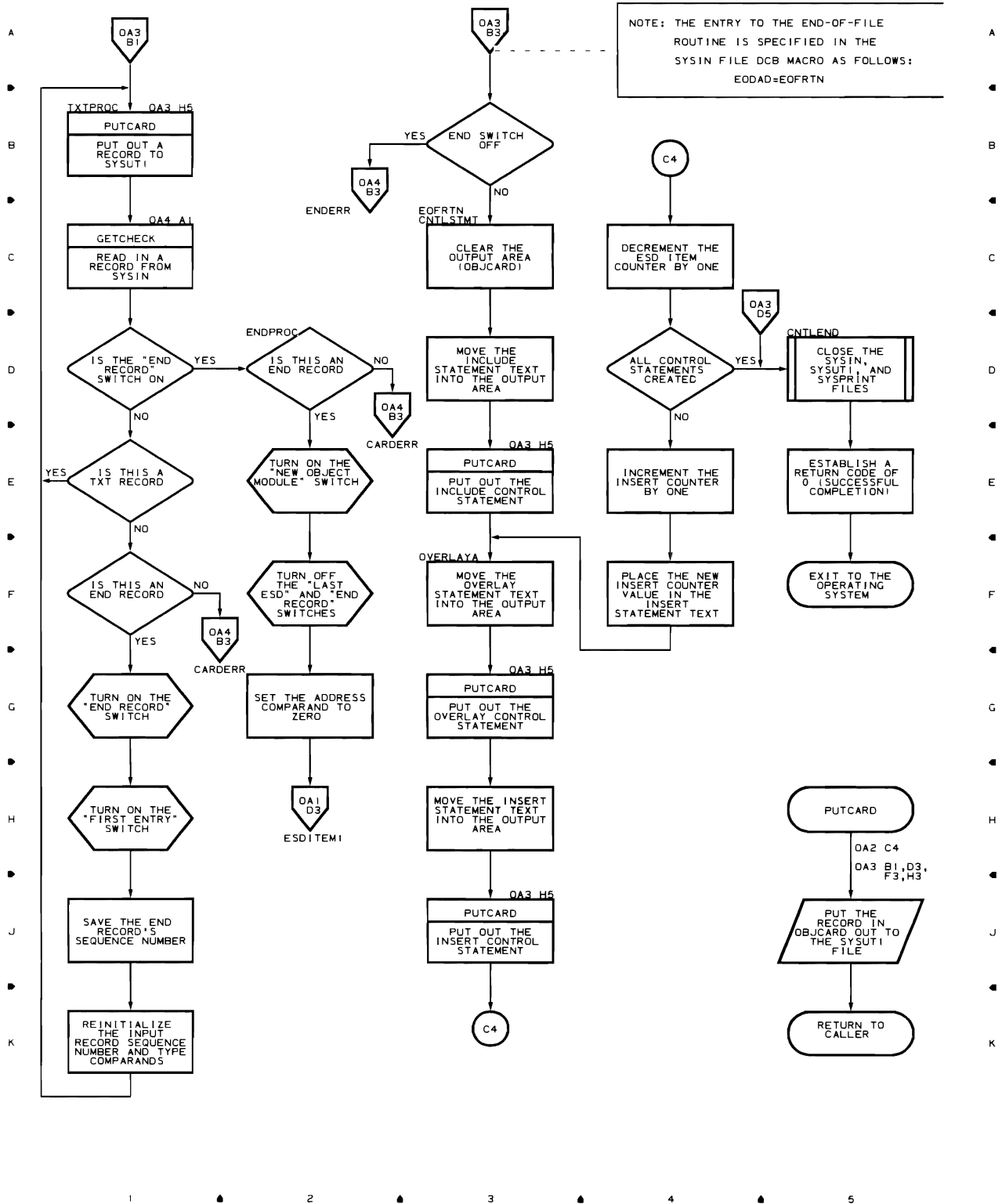
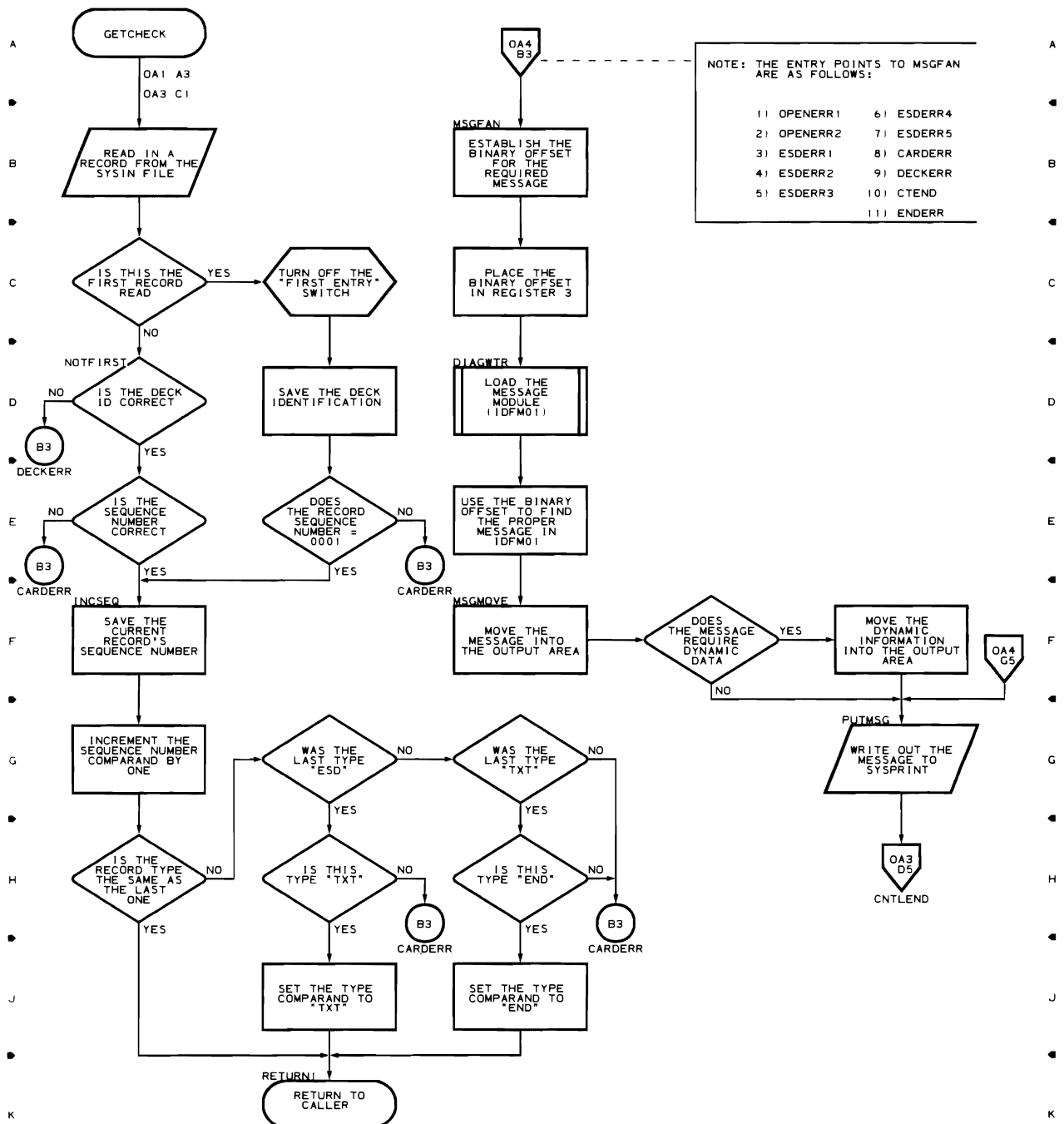


Chart OA4 OS FD UTILITY CONTROL STEP: IDFCT CHART 4 OF 5

1 2 3 4 5



1 2 3 4 5

Chart OA5 OS FD UTILITY CONTROL STEP: IDFCT CHART 5 OF 5

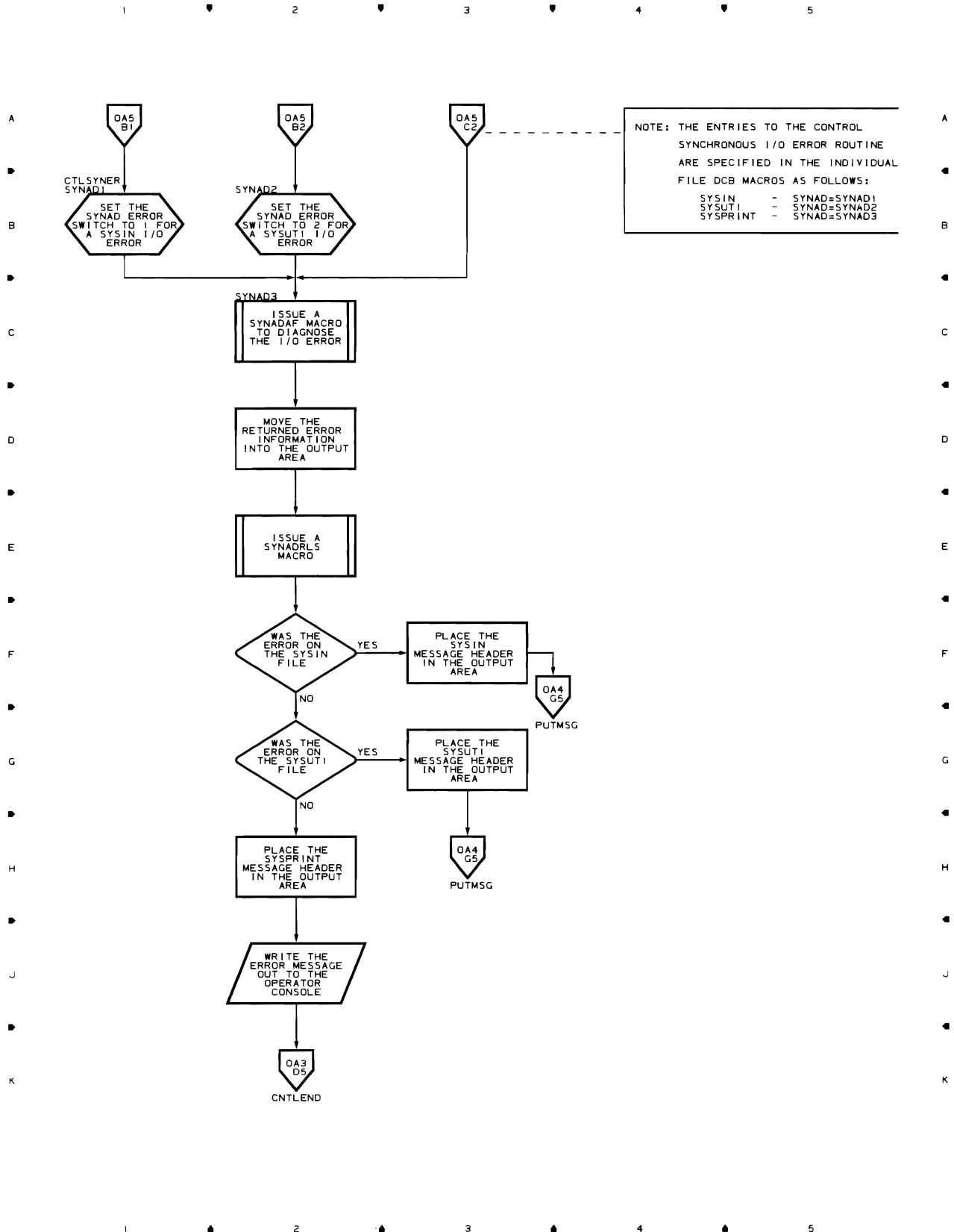


Chart OB1 OS FD UTILITY STORAGE STEP: IDFST CHART 1 OF 4

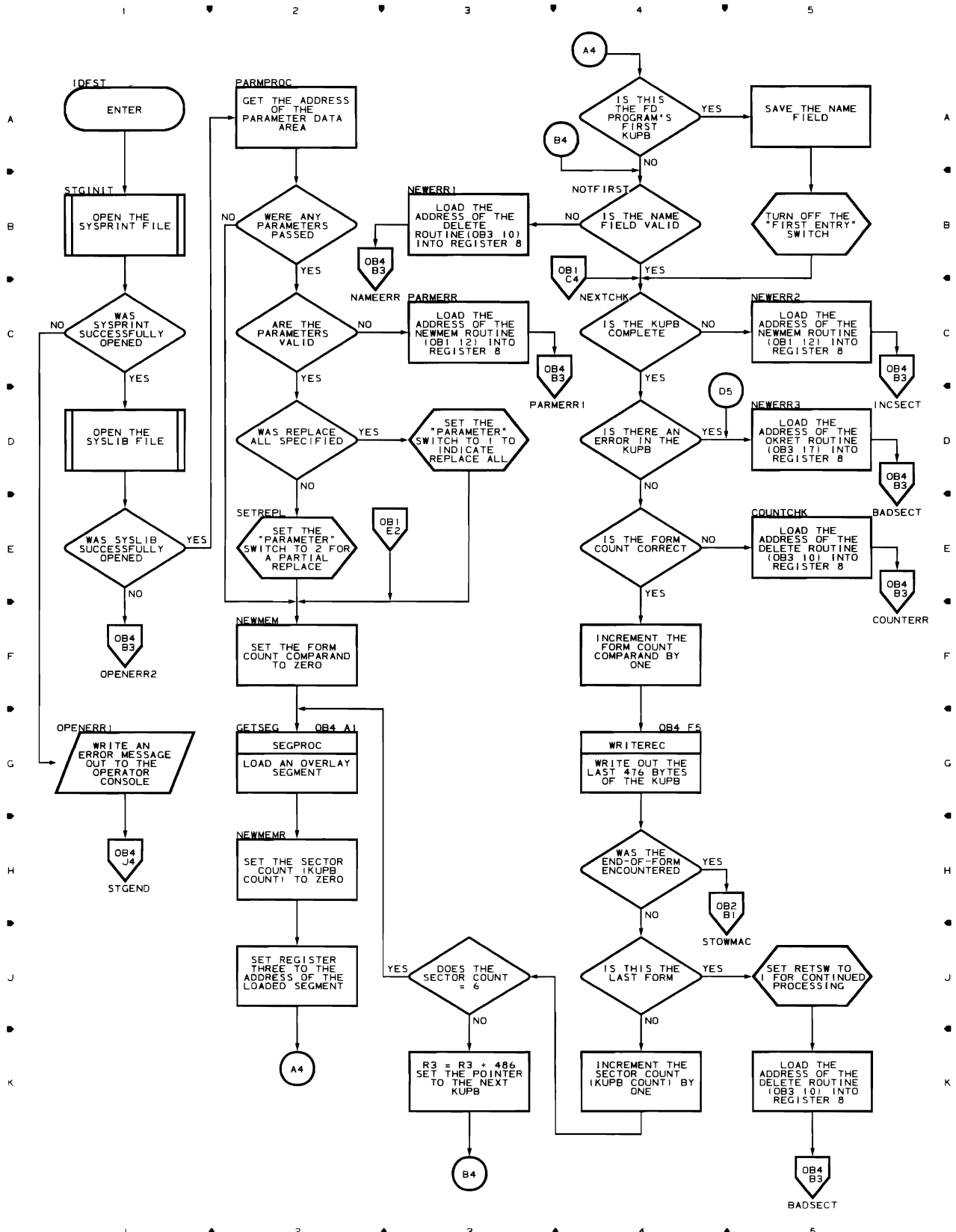


Chart OB2 OS FD UTILITY STORAGE STEP: IDFST CHART 2 OF 4

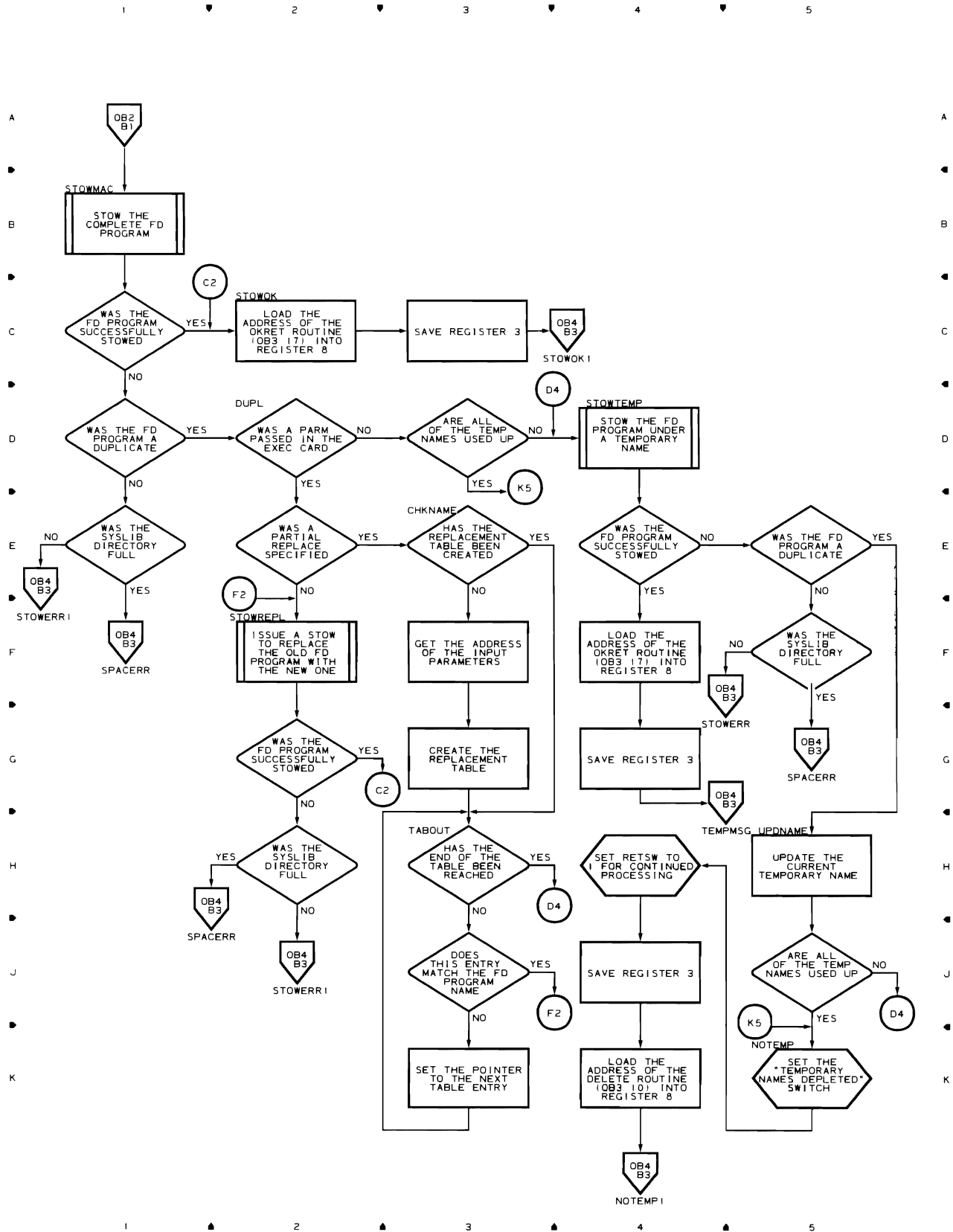


Chart OB3 OS FD UTILITY STORAGE STEP: IDFST CHART 3 OF 4

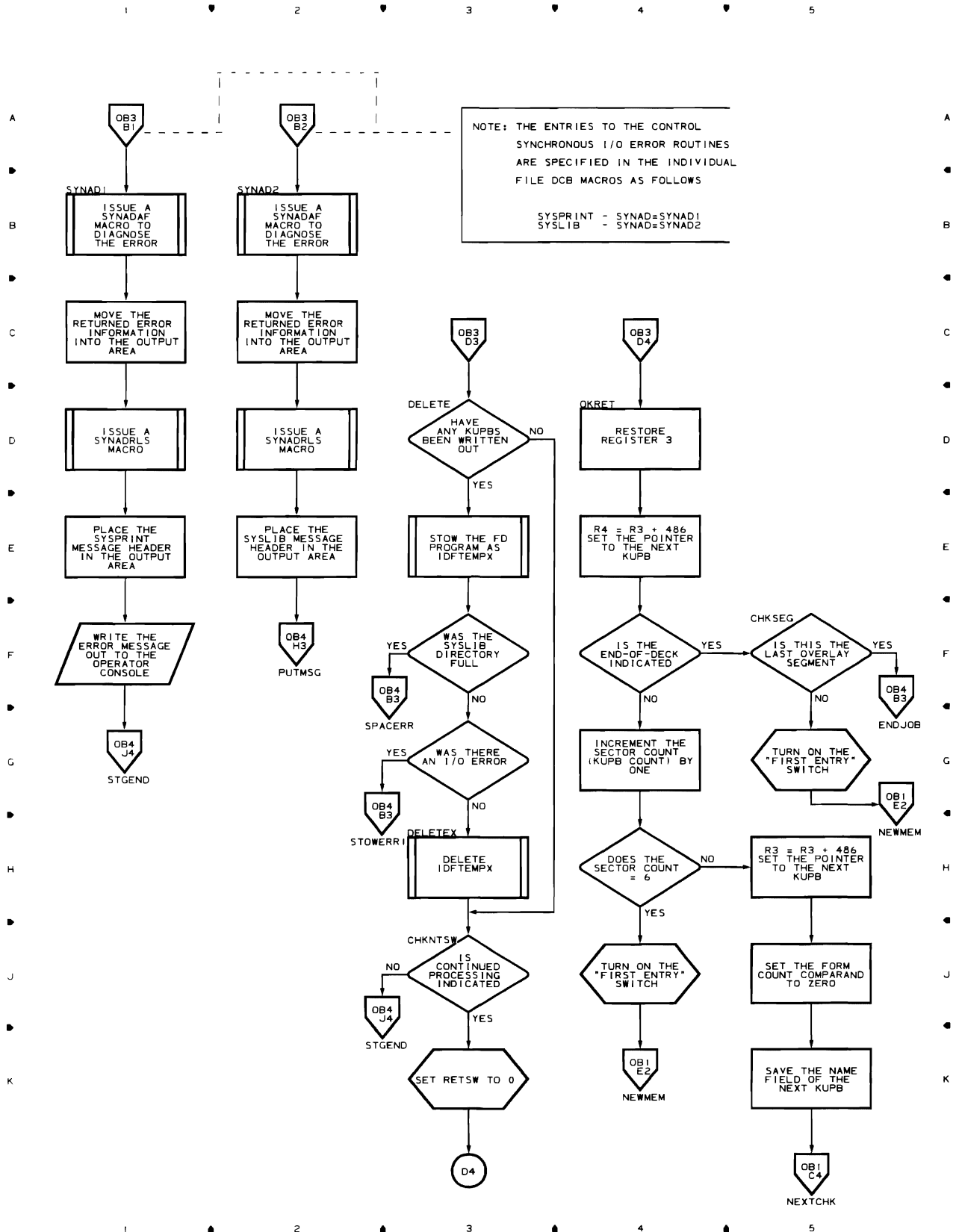




Chart OB4 OS FD UTILITY STORAGE STEP: IDFST CHART 4 OF 4

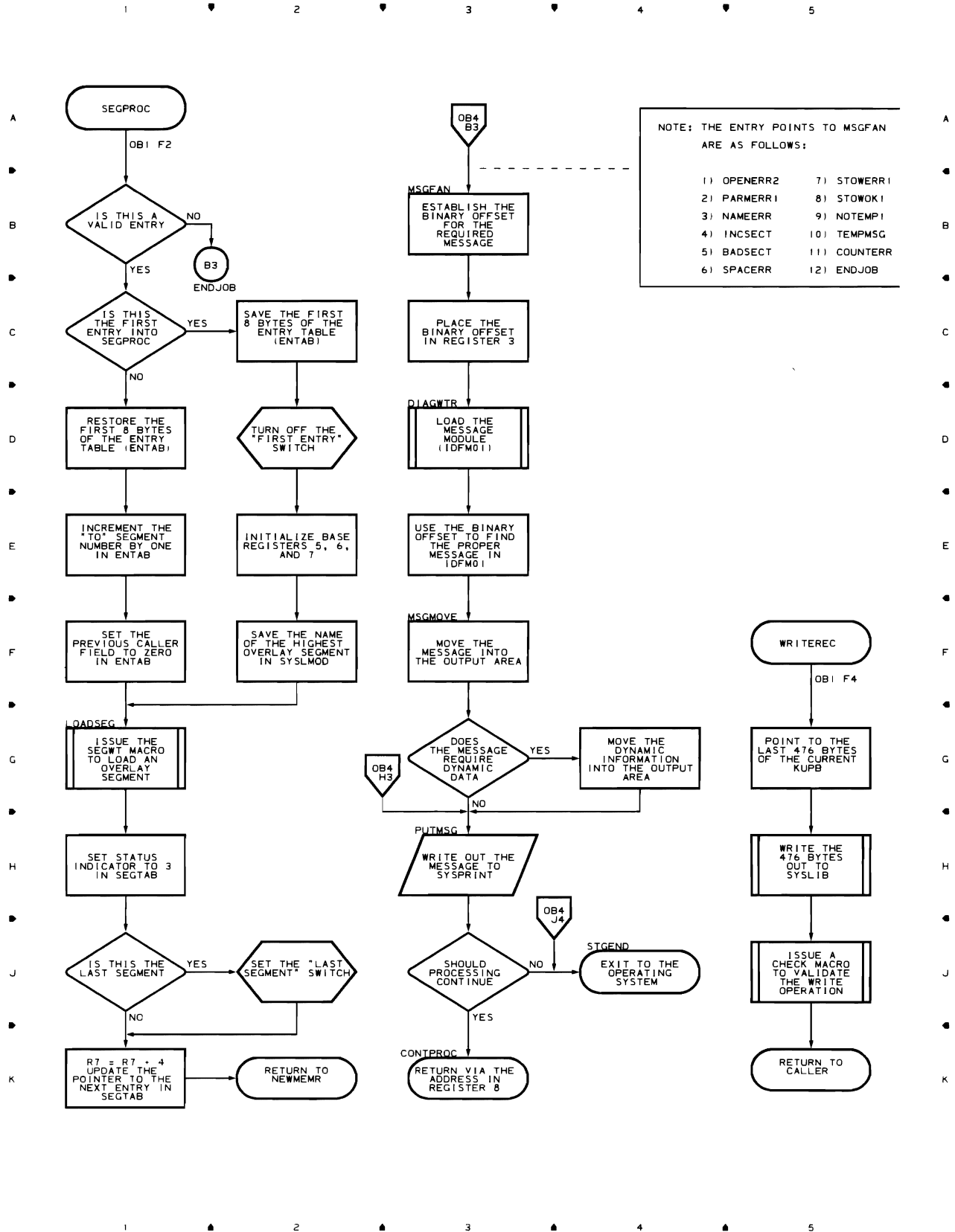


Chart DA1 DOS FD UTILITY CONTROL STEP: IJLFCF CHART 1 OF 4

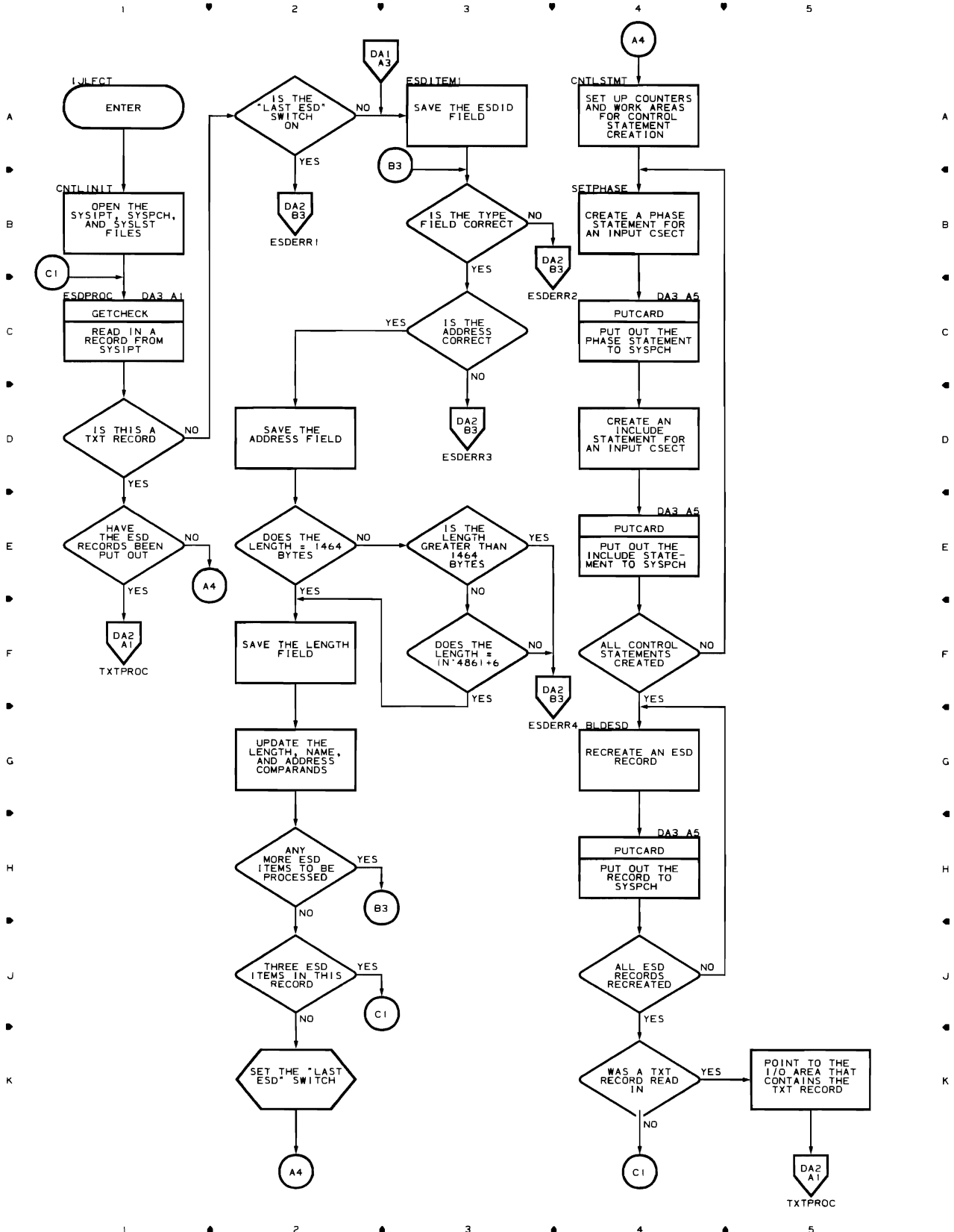


Chart DA2 DOS FD UTILITY CONTROL STEP: IJLFCT CHART 2 OF 4

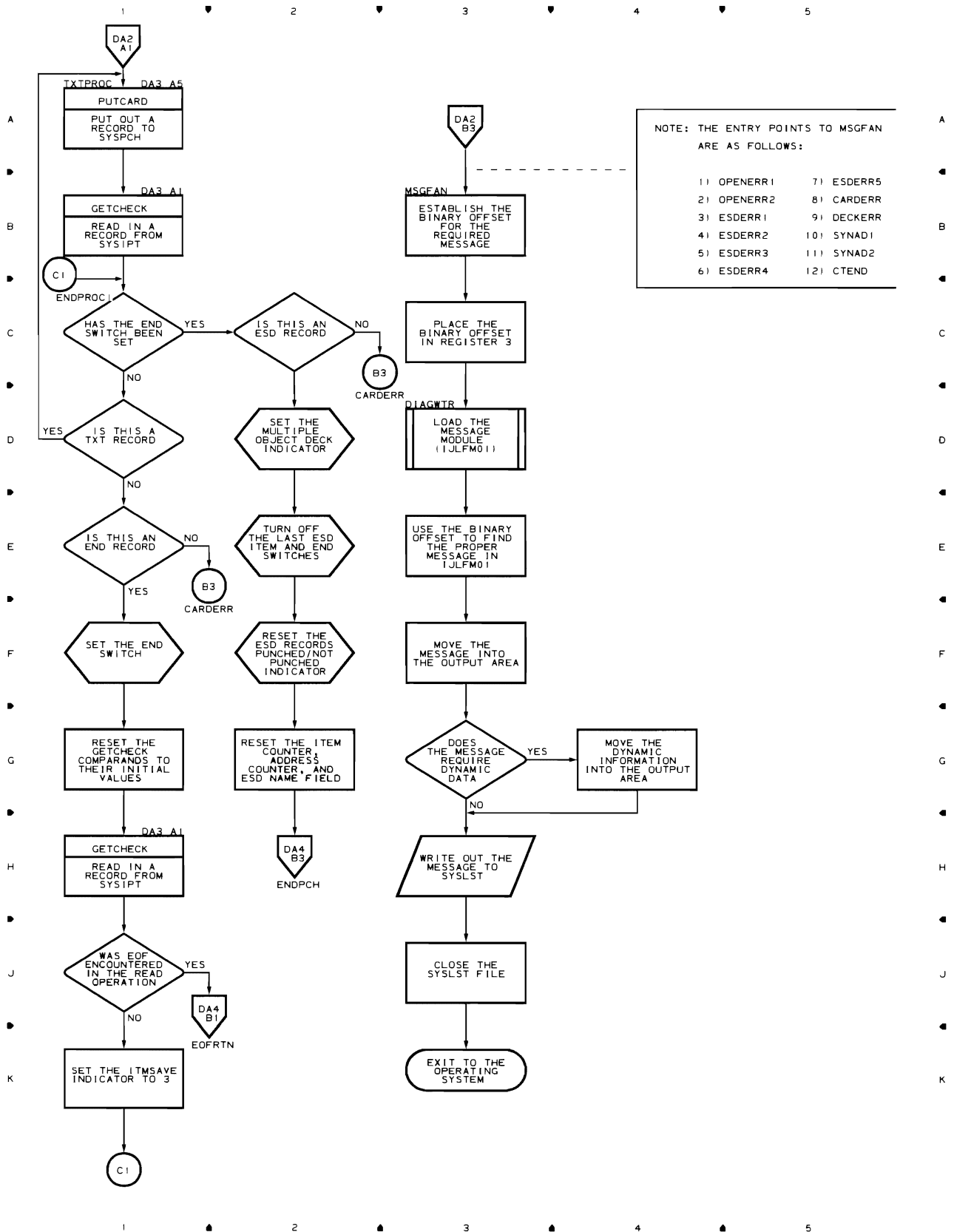


Chart DA3 DOS FD UTILITY CONTROL STEP: IJLFCT CHART 3 OF 4

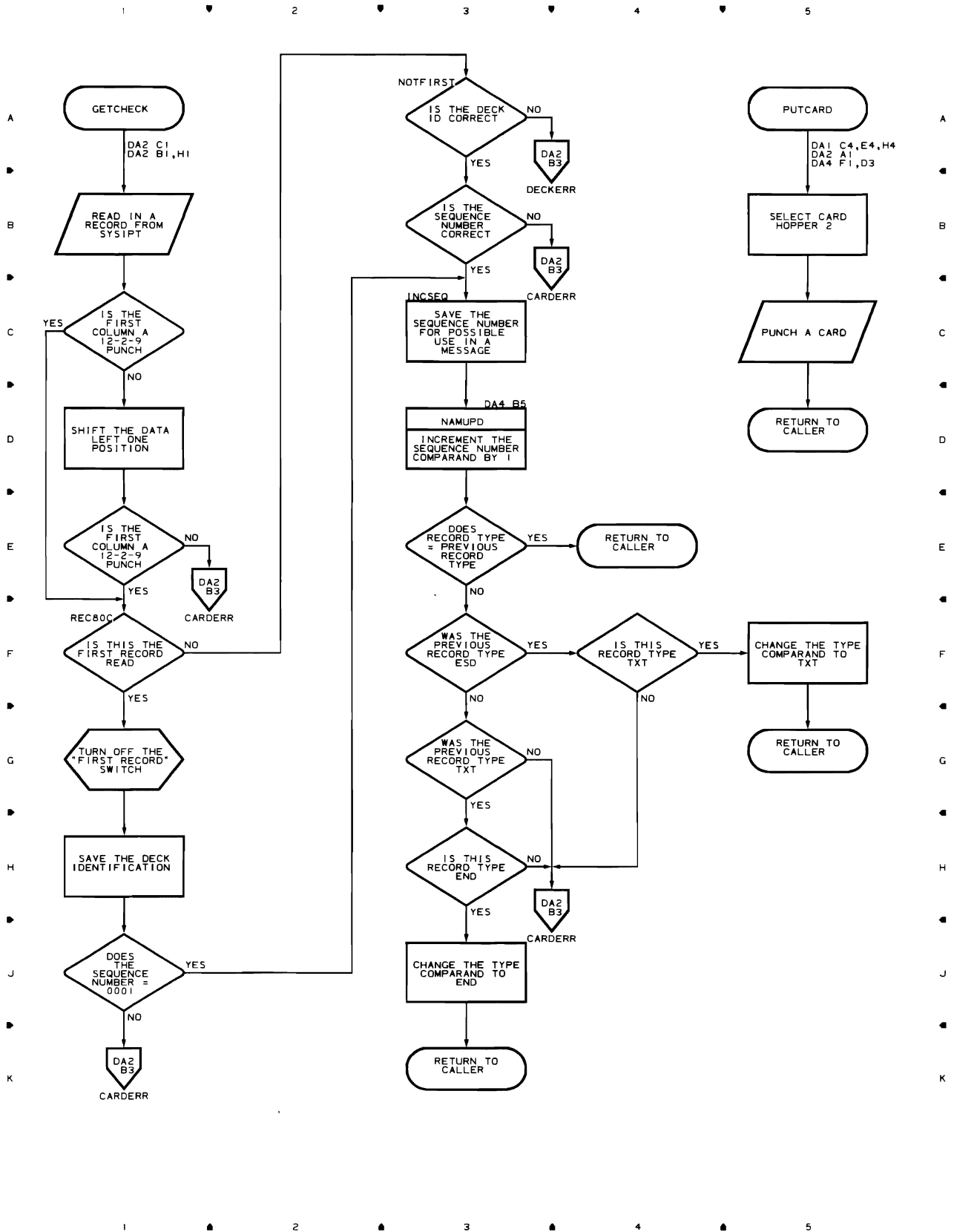


Chart DA4 DOS FD UTILITY CONTROL STEP: IJLFCT CHART 4 OF 4

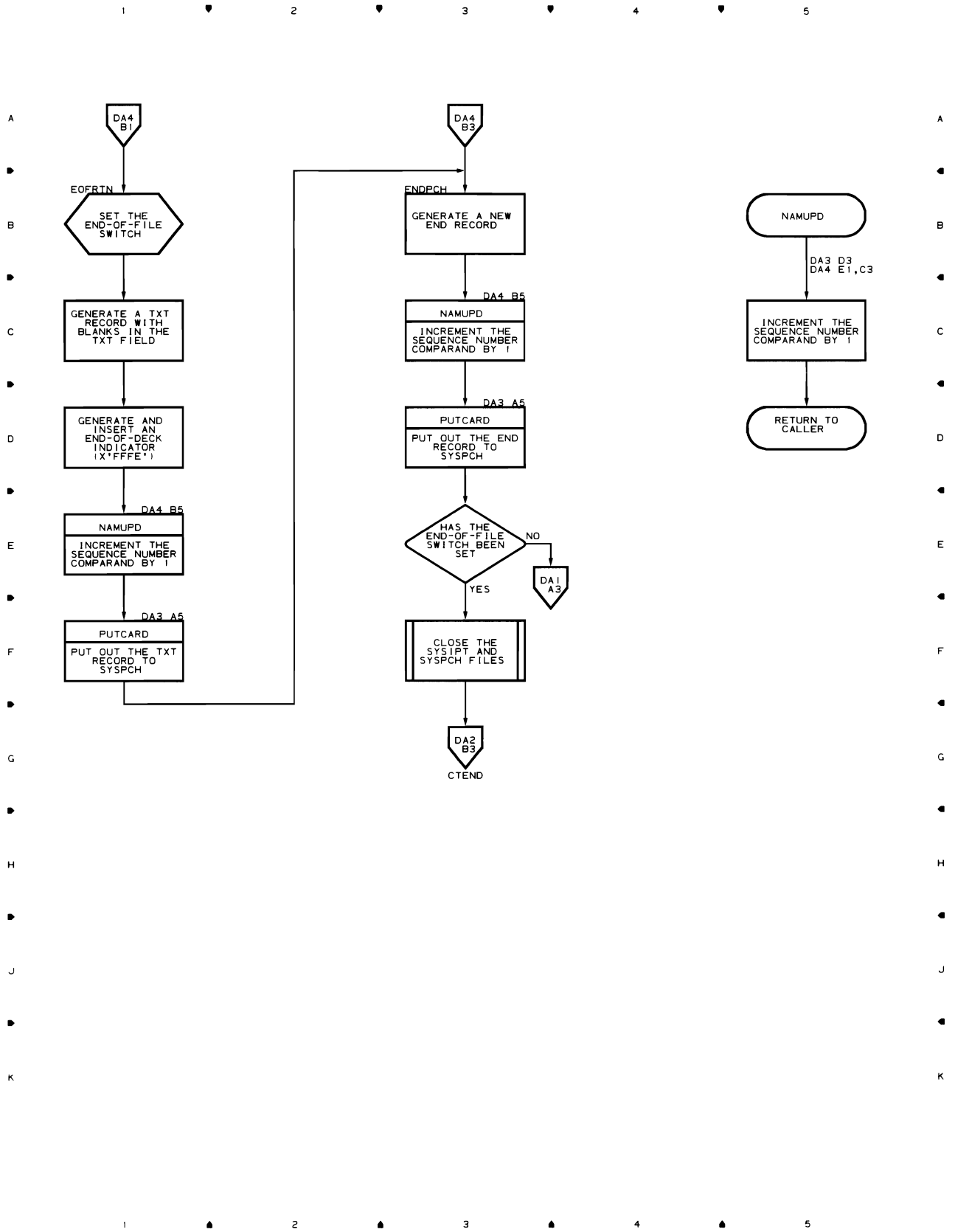


Chart DB1 DOS FD UTILITY STORAGE STEP: IJLFST CHART 1 OF 2

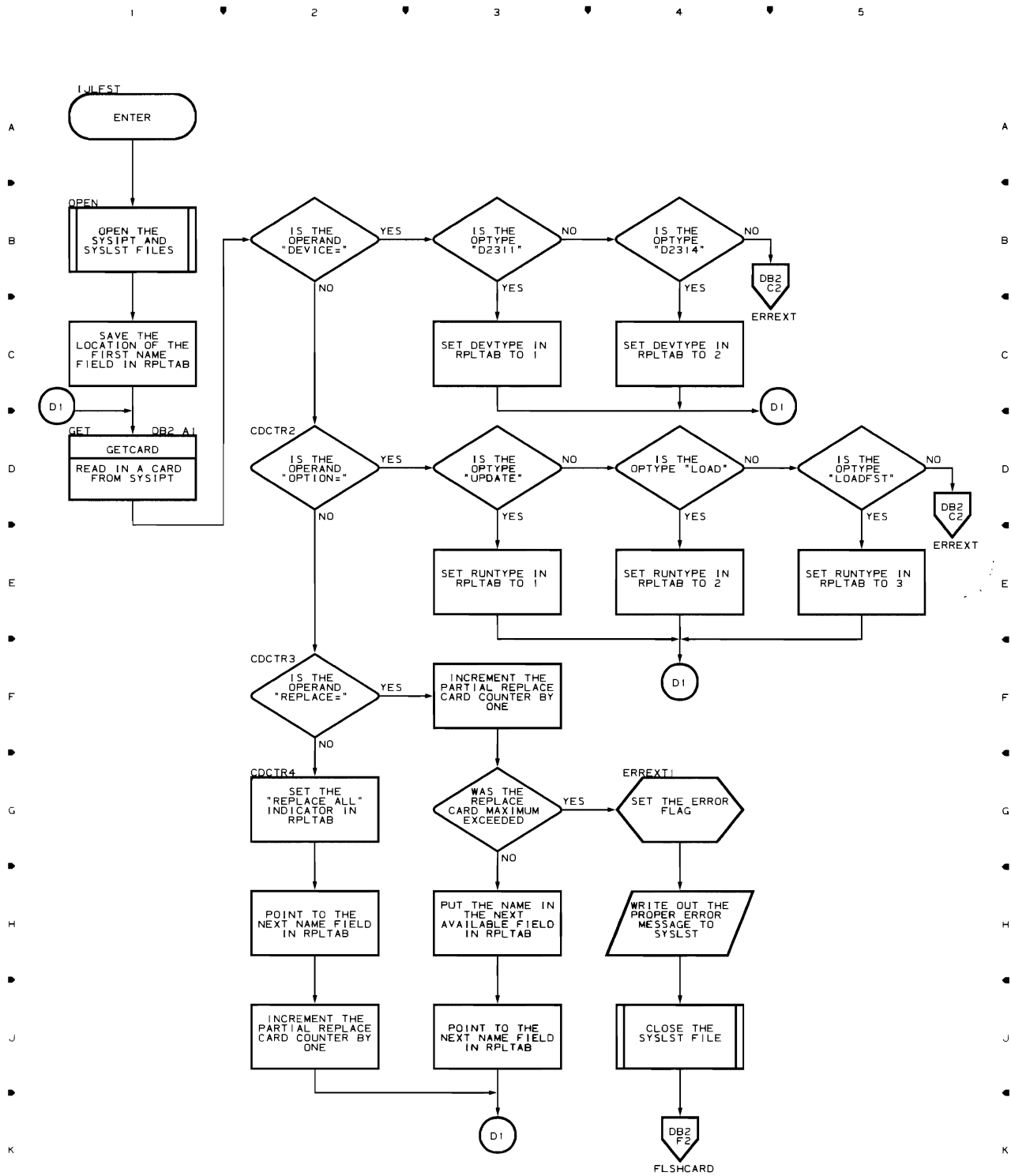


Chart DB2 DOS FD UTILITY STORAGE STEP: IJLFST CHART 2 OF 2

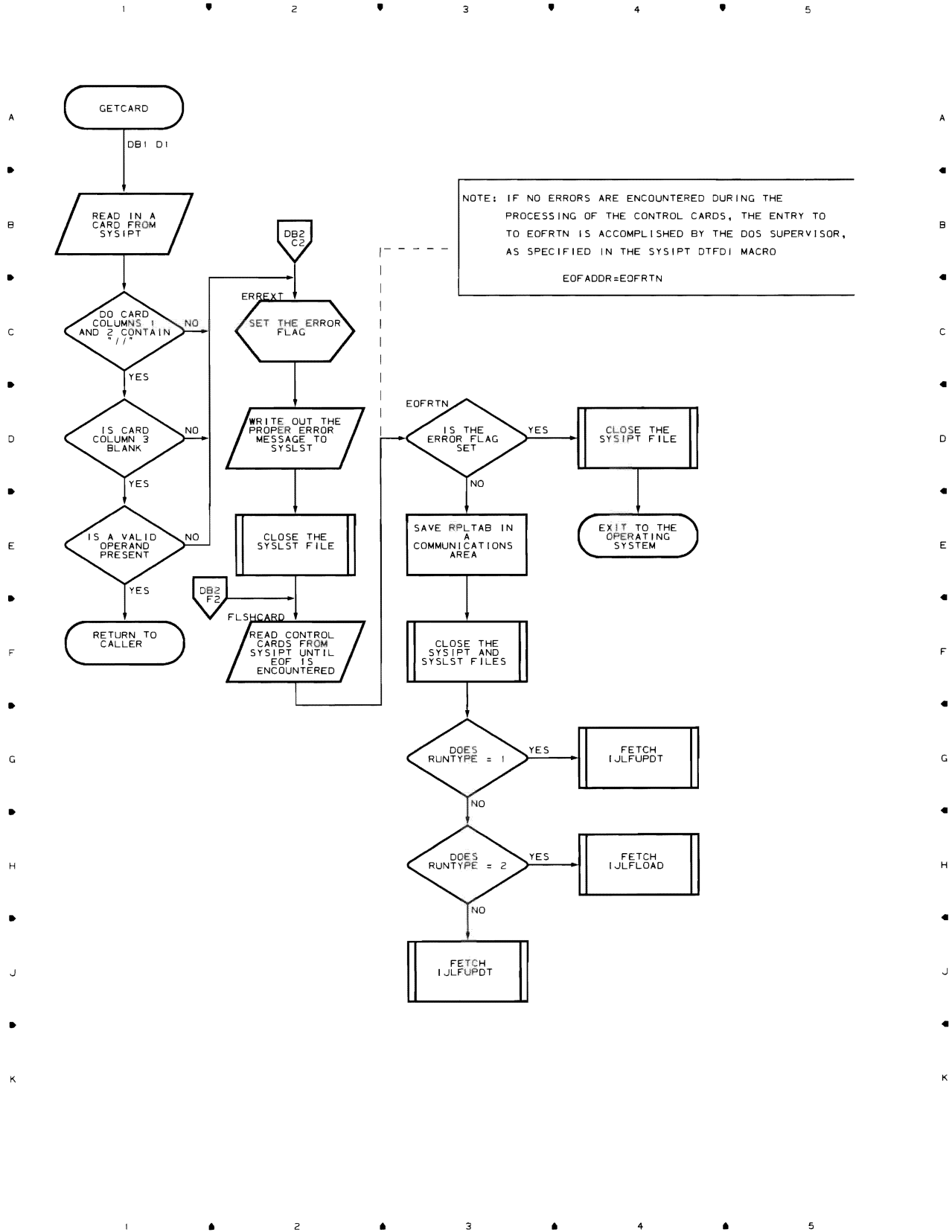


Chart DC1 DOS FD UTILITY STORAGE STEP: IJLFLOAD CHART 1 OF 2

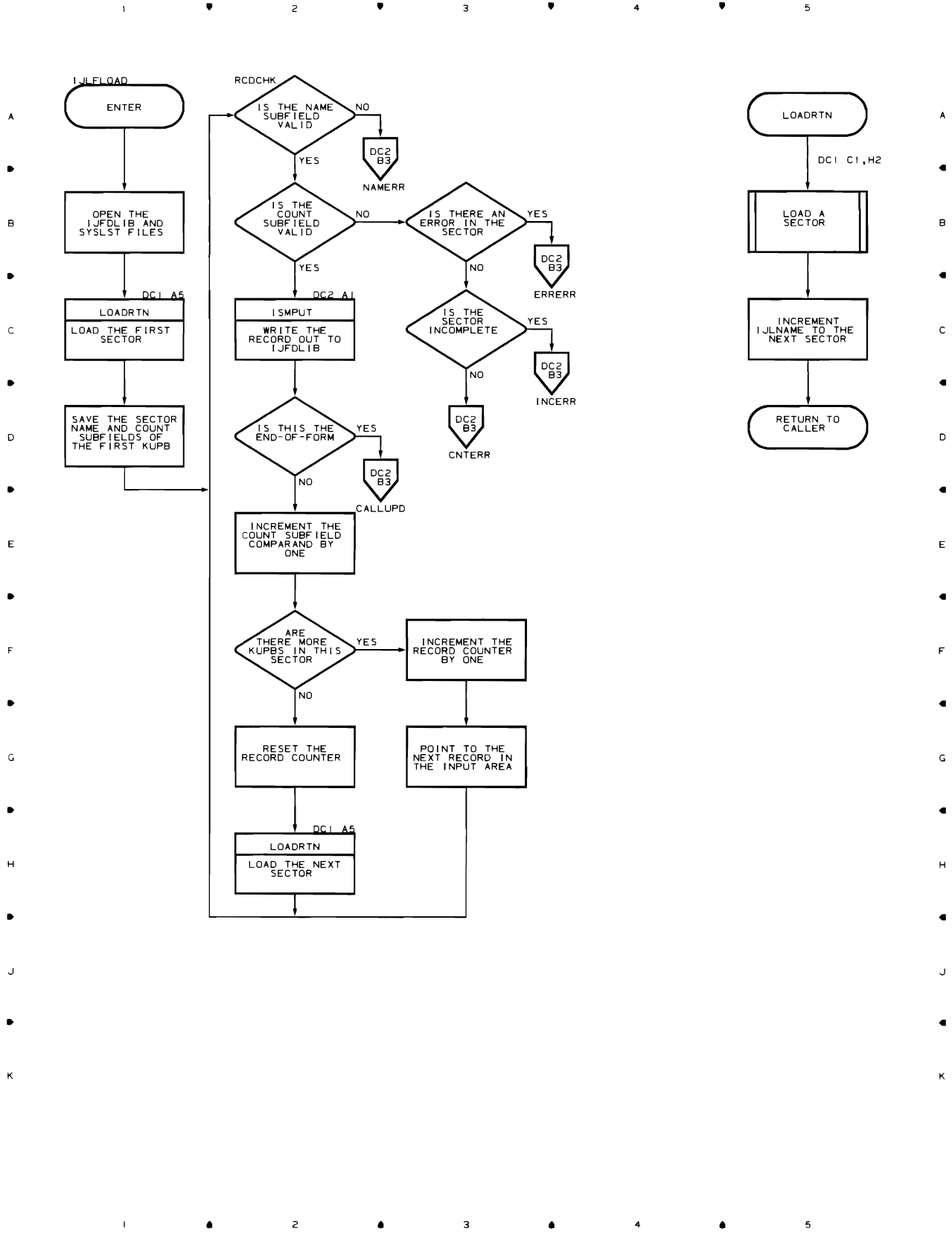




Chart DC2 DOS FD UTILITY STORAGE STEP: IJLFLOAD CHART 2 OF 2

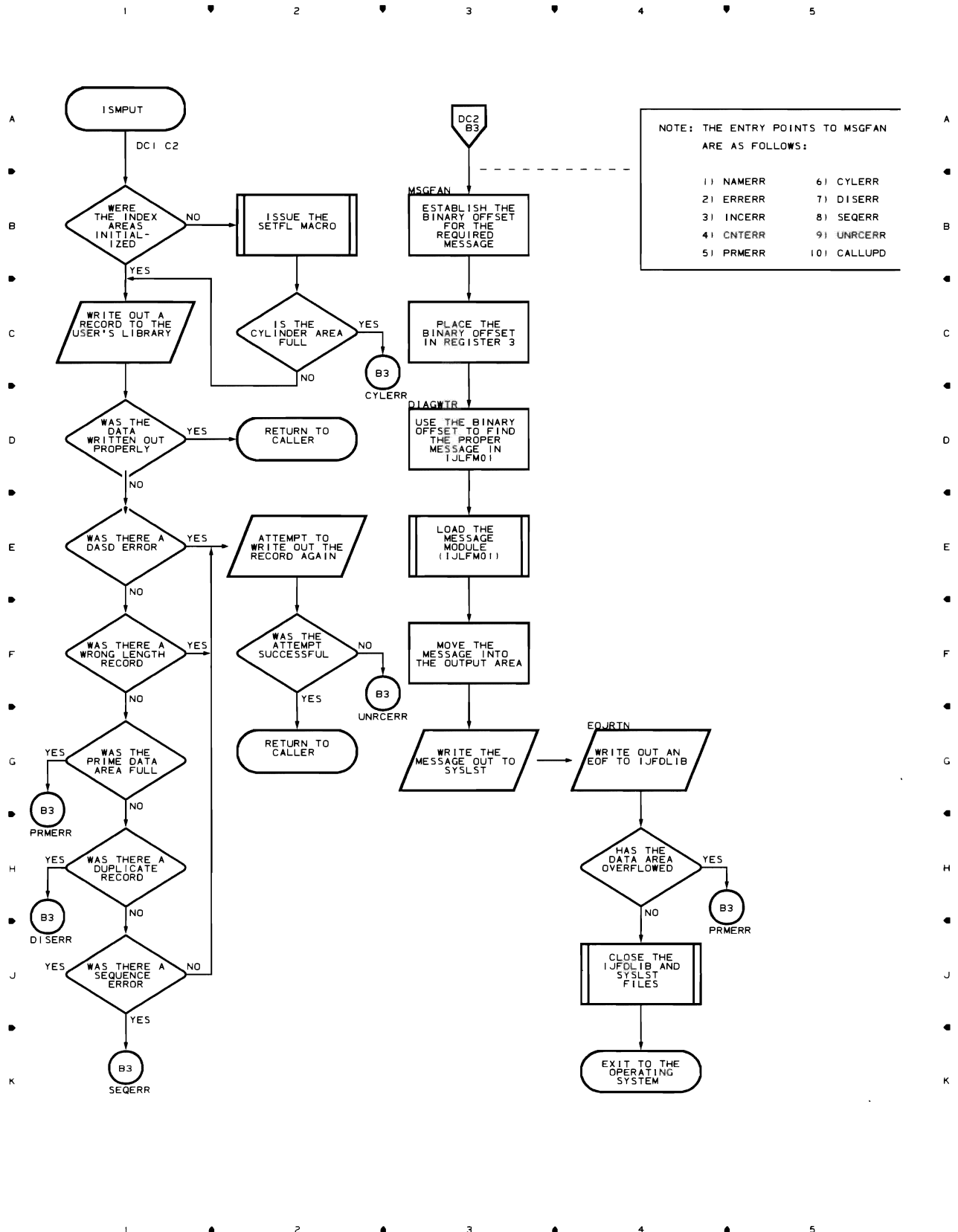


Chart DD1 DOS FD UTILITY STORAGE STEP: IJLFUPDT CHART 1 OF 4

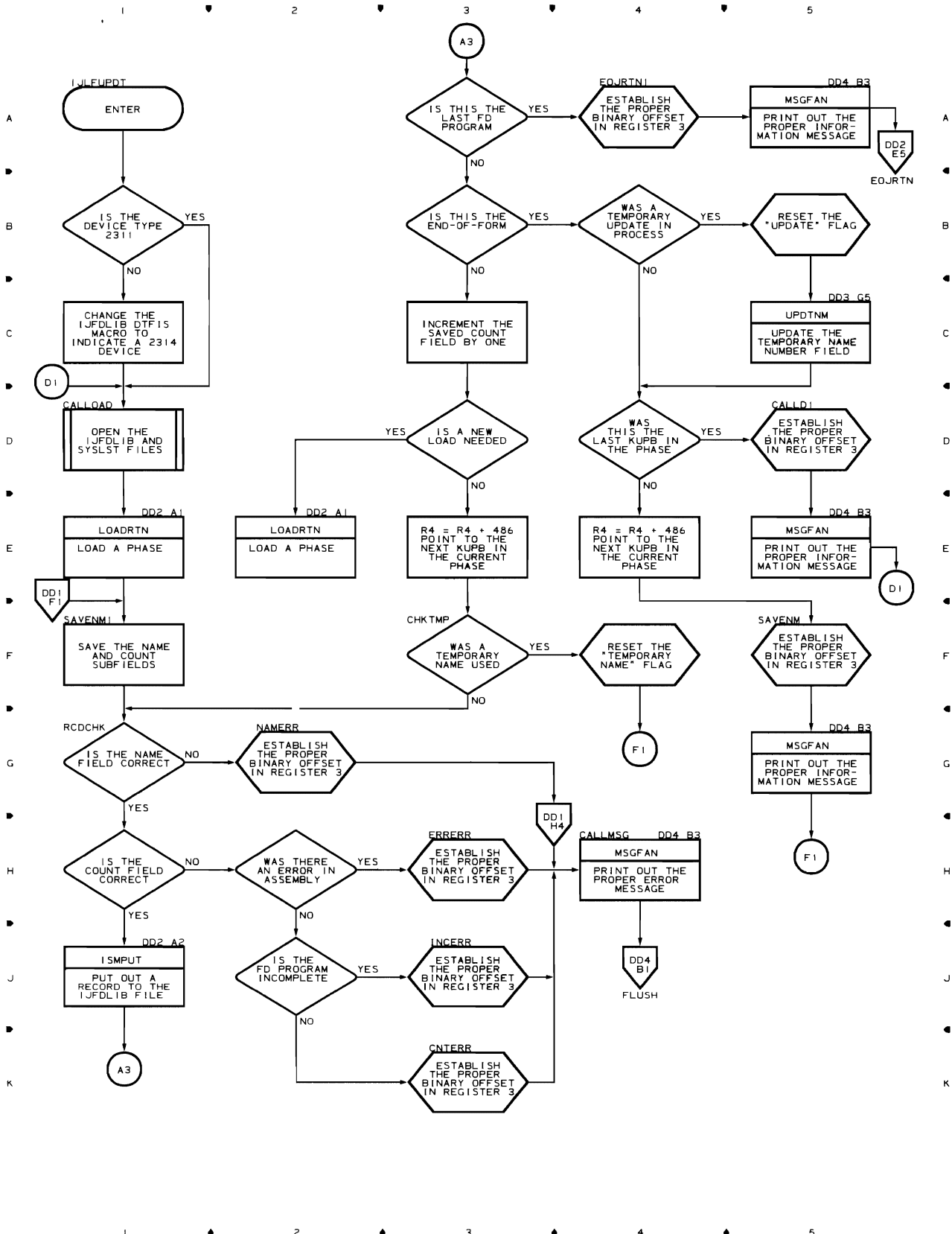


Chart DD2 DOS FD UTILITY STORAGE STEP: IJLFUPDT CHART 2 OF 4

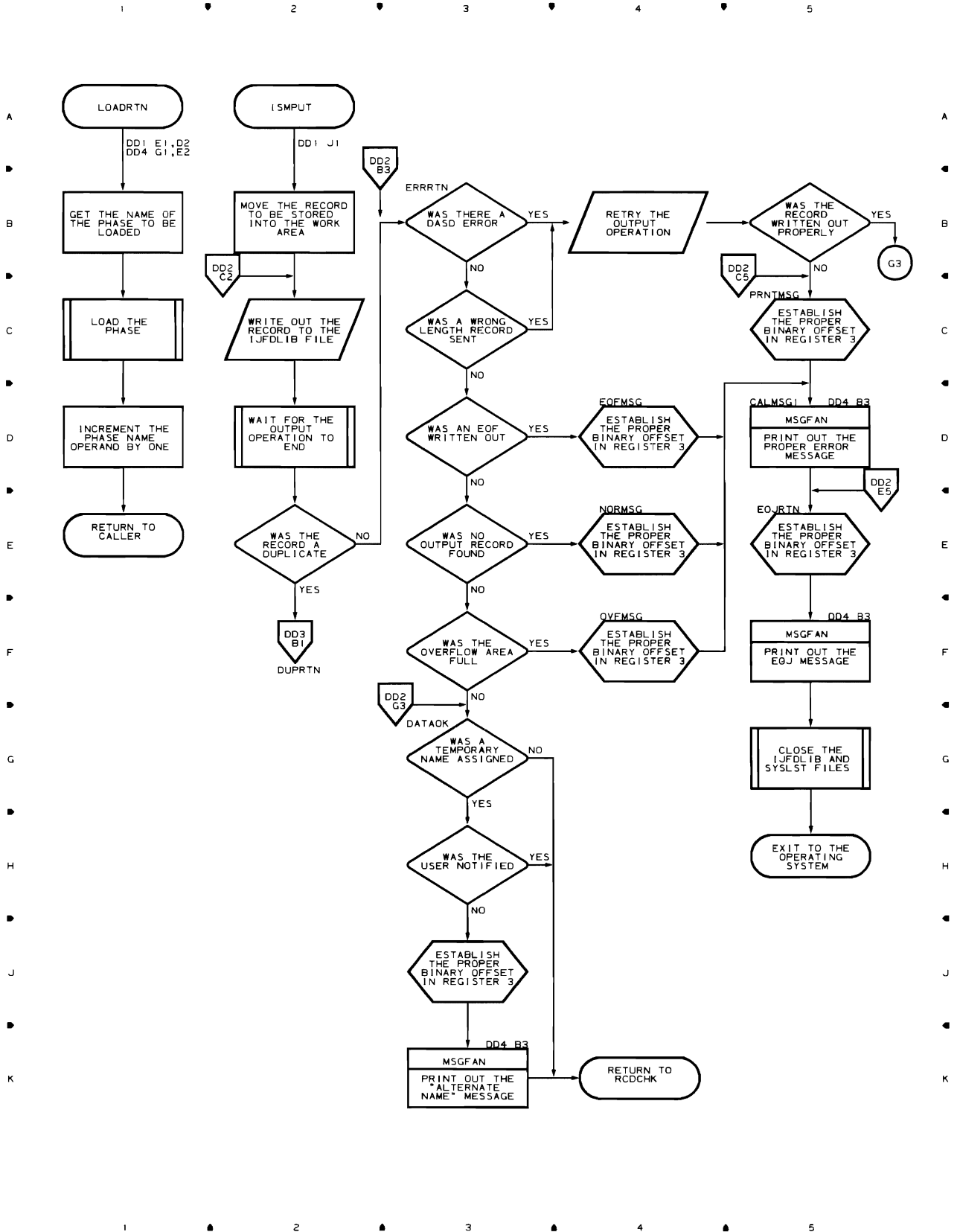


Chart DD3 DOS FD UTILITY STORAGE STEP: IJLFUPDT CHART 3 OF 4

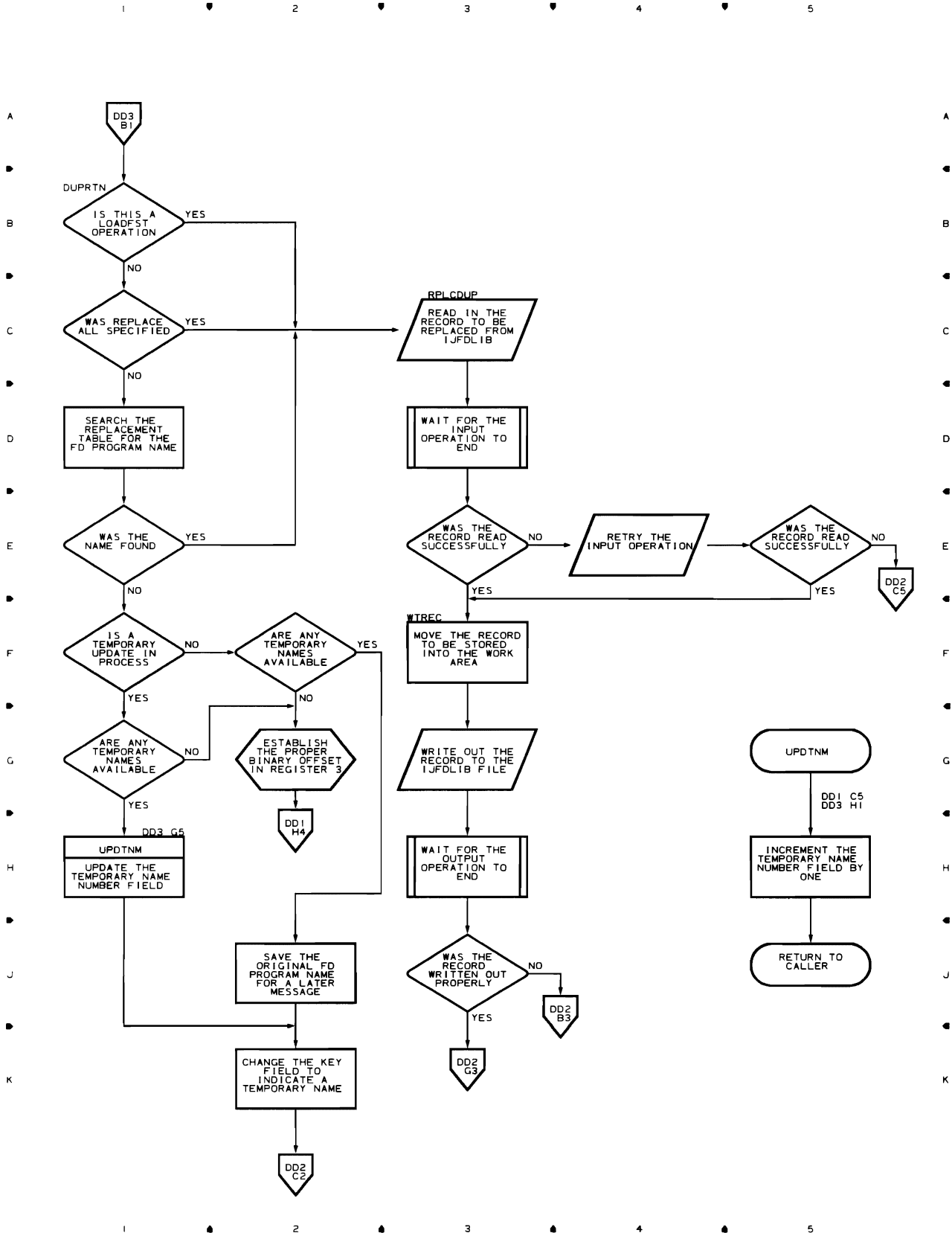
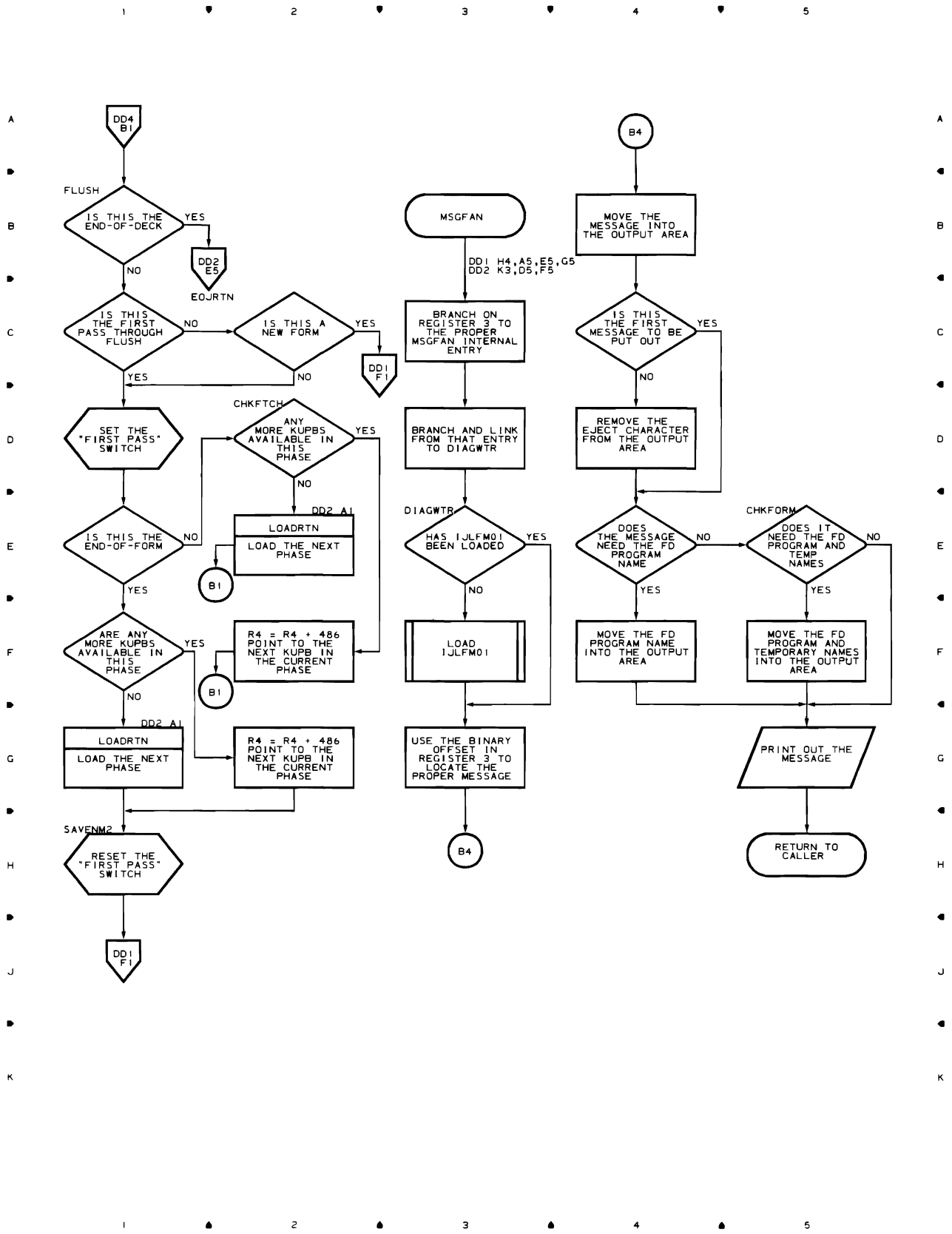


Chart DD4 DOS FD UTILITY STORAGE STEP: IJLFUPDT CHART 4 OF 4



## Section 4: Directory

Entry Point Name	Name of Routine or Table	Module Name	PLM References		Entry Point Name	Name of Routine or Table	Module Name	PLM References		
			Section	Chart ID				Section	Chart ID	
CALLOAD	Load Initial	IJLFLOAD	3	DC1	IJLFM01	DOS FD Utility				
CALLOAD	Update Initial	IJLFUPDT	3	DD1		Message Module	IJLFM01	3	—	
CNTLEND	Control End	IDFCT	3	OA3	INMOD	IJLFST DIMOD	IJLFST	3	—	
CNTLINIT	Control Initial	IDFCT	3	OA1	ISMPUT	ISAM Put	IJLFLOAD	3	DC2	
CNTLINIT	Control Initial	IJLFCT	3	DA1	ISMPUT	ISAM Put	IJLFUPDT	3	DD2	
CNTLSTMT	Create Control Statements	IDFCT	3	OA3	LOADMOD	IJLFLOAD ISMOD	IJLFLOAD	3	—	
CNTLSTMT	Create Control Statements	IJLFCT	3	DA1	LOADRTN	Load Overlay Phases	IJLFLOAD	3	DC1	
					LOADRTN	Load Overlay Phases	IJLFUPDT	3	DD2	
CTLSYNER	Control Synchronous I/O Error	IDFCT	3	OA5	MSGFAN	Message Fan-in	IDFCT	3	OA4	
					MSGFAN	Message Fan-in	IDFST	3	OB4	
					MSGFAN	Message Fan-in	IJLFCT	3	DA2	
DIAGWTR	Diagnostic Writer	IDFCT	3	OA4	MSGFAN	Message Fan-in	IJLFLOAD	3	DC2	
DIAGWTR	Diagnostic Writer	IDFST	3	OB4	MSGFAN	Message Fan-in	IJLFUPDT	3	DD4	
DIAGWTR	Diagnostic Writer	IJLFCT	3	DA2	NEWMEM	New Member	IDFST	3	OB1	
DIAGWTR	Diagnostic Writer	IJLFLOAD	3	DC2	OBJCARD	Card Image				
DIAGWTR	Diagnostic Writer	IJLFUPDT	3	DD4		Storage Area	IDFCT	3	—	
ENTAB	Entry Table	IDFST	2	—	OPEN	Open Files	IJLFST	3	DB1	
EOFRTN	End-of-File	IJLFST	3	DB2	OUTMOD	IJLFLOAD DIMOD	IJLFLOAD	3	—	
EOJRTN	End-of-Job	IJLFLOAD	3	DC2	OUTMOD	IJLFUPDT DIMOD	IJLFUPDT	3	—	
EOJRTN	End-of-Job	IJLFUPDT	3	DD2	PARMPROC	PARM Processor	IDFST	3	OB1	
ERREXT	Common Error Exit	IJLFST	3	DB2	PUTCARD	Put Out a Card	IDFCT	3	OA3	
ERREXT1	Special Error Exit	IJLFST	3	DB1	PUTCARD	Put Out a Card	IJLFCT	3	DA3	
					RCDCHK	Validate Records	IJLFLOAD	3	DC1	
ESDPROC	ESD Processor	IDFCT	3	OA1	RCDCHK	Validate Records	IJLFUPDT	3	DD1	
ESDPROC	ESD Processor	IJLFCT	3	DA1	REPLTAB	Replacement Table	IDFST	2,3	—	
FLSHCARD	Flush Cards	IJLFST	3	DB2	RPLTAB	Replacement Table	IJLFST	2,3	—	
FLUSH	Flush FD Program	IJLFUPDT	3	DD4	SEGPROC	Segment Processor	IDFST	3	OB4	
GET	Get and Process Control Cards	IJLFST	3	DB1	SEGTAB	Segment Table	IDFST	2	—	
GETCARD	Get and Validate a Control Card	IJLFST	3	DB2	STGEND	Storage End	IDFST	3	OB4	
GETCHECK	Get and Check an Input Record	IDFCT	3	OA4	STGINIT	Storage Initial	IDFST	3	OB1	
GETCHECK	Get and Check an Input Record	IJLFCT	3	DA3	SYNAD1	Control Synchronous I/O Error	IDFST	3	OB3	
IDFM01	OS FD Utility Message Module	IDFM01	3	—		Control Synchronous I/O Error				
IJFCBIC	IJLFCT DIMOD	IJLFCT	3	—	SYNAD2	Control Synchronous I/O Error	IDFST	3	OB3	
						TXTPROC	TXT Processor	IDFCT	3	OA3
						TXTPROC	TXT Processor	IJLFCT	3	DA2
					UPDTMOD	IJLFUPDT ISMOD	IJLFUPDT	3	—	
					UPDTNM	Update Name	IJLFUPDT	3	DD3	

## **Section 5: Data Area Layouts**

The information that would ordinarily be contained in this section has been placed in the text of Sections 2 and 3 to aid the discussion there. Refer to Section 2 for the layout of SEGTAB and ENTAB and to Section 3 for the layout of REPLTAB and RPLTAB.

## Section 6: Diagnostic Aids

No information is provided for this section because of the simplicity of the FD utility programs.



## **Part 4. Appendixes and Glossary**

# Contents

<b>Appendix A: Format of the Form Description Macro</b>	
Instructions . . . . .	4-3
<b>Appendix B: The Form Description Diagnostic Macros</b>	4-5
FDDSPY Macro . . . . .	4-5
FDTRACE Macro . . . . .	4-5
<b>Appendix C: Diagnostic Messages</b>	4-6
MNOTE Messages . . . . .	4-6
Informational MNOTE Messages . . . . .	4-6
Warning MNOTE Messages . . . . .	4-8
Termination MNOTE Messages . . . . .	4-10
FD Program Error Message . . . . .	4-14
Data Source Error Message . . . . .	4-14
<b>Appendix D: Sample FD Program</b>	4-15
<b>Glossary</b> . . . . .	4-27

## Appendix A. Format of the Form Description Macro Instructions

Name	Operation	Operands
symbol	FDFORM	FID='ddd' [,PACKING= { $\frac{NO}{YES}$ } ] DELIMIT } [,DEVICES=(3735,K[D] ) ] [,BUFFERS=( [RPB] [,LPB[, { $\frac{132}{126}$ } ] ) ) ] d } [,MRGSTOP= { $\frac{0}{d}$ } ] [,MESSAGE=( { cc[ ( { $\frac{1}{d}$ } ) ] } [, { cc[ ( { $\frac{1}{d}$ } ) ] } ] ... ) ] 'string' } { 'string' } [,HTAB=(d[,d] ... ) ]
[symbol]	FDPAGE	[pagenum] [,HEIGHT= { $\frac{66}{d}$ } ] [,VMRG=( [ { $\frac{1}{dt}$ } ] [, { $\frac{height}{db}$ } ] ) ] [,SAVELOC= { $\frac{NO}{YES}$ } ] d }
[symbol]	FDLINE	[ { linenum } ] [,WIDTH= { $\frac{85}{d}$ } ] SKIP(d) } [,HMRG=( [ { $\frac{mrgstop+1}{dl}$ } ] [, { $\frac{width}{dr}$ } ] ) ] [,CYCLE=( [d] [,limit] [,target] ) ] [,SAVELOC= { $\frac{NO}{YES}$ } ] d }
[symbol]	FDFIELD	[ { $\frac{hmrgdl}{prevdr+1}$ } ] [, { $\frac{comping}{LNG(d)}$ } ] d }                  dr } DUMMY } [,CTR=( (d,op[,FIELD]) [, (d,op[,FIELD]) ] ... ) ] [,IND=( (d,logexp) [, (d,logexp) ] ... ) ] [,CYCLE=( [d] [,limit] [,target] ) ] [,SAVELOC= { $\frac{NO}{YES}$ } ] d } [,SOURCE=(origin[,qualifier] ... ) ] [,KIND= { $\frac{U}{A}$ } ] N } AN } K } [,SELFCHK= { $\frac{NO}{\{ \frac{10}{11} \}}$ } ] [,GENONLY] } [,COUNT= ( [ (MIN,) { $\frac{1}{d1}$ } ] [, (MAX,) { $\frac{compmax}{d2}$ } ] ) ] d } [,COMPARE=( [FIELD,] comparopr,comparand [, { AND } [,FIELD,] comparopr,comparand ] ... ) ] OR } [,SINK=( [ (destination[,qualifier] ... ) ] [, [ (destination[,qualifier] ... ) ] ] ... ) ] [,JUSTIFY=( [justcode] [,justcode] ) ... ) ] [,FILL=( ['char'] [, ['char']] ... ) ] [,UL=( { $\frac{NO}{YES}$ } ] [, [ { $\frac{NO}{YES}$ } ] ] ... ) ] [,PICTURE=( ['picturespec'] [, ['picturespec']] ... ) ] [,BATCH=d]

<i>Name</i>	<i>Operation</i>	<i>Operands</i>
[symbol]	FDCTRL	<pre> [IF=(logterm[ { AND } ,logterm] ... ) ]                 { OR } [CTR=( (d[,CLR] [,op,opnd] ... )         [(d[,CLR] [,op,opnd] ... )] ... ) ] [IND=( (d, { ON } )[(d, { ON } )] ... ) ]                 { OFF }      { OFF }                 { INV }      { INV } [TOTAL=( (d,'fid',CTR(d) )           [(d,'fid',CTR(d) )] ... ) ] [COMMAND=( (cmndgrp)[, (cmndgrp) ] ... ) ] [GOTO=target] [CYCLE=( [d] [,limit] [,target] ) ] [SAVELOC= { NO } ]            { YES }            { d } </pre>
[symbol]	FDEND	,(This macro has no operands.)

## Appendix B: The Diagnostic Form Description Macros

### FDDSPLY MACRO

The FDDSPLY macro provides the ability to display the global variables (arrays) used in the FD macro processing. FDDSPLY should be specified simply:

FDDSPLY

The macro, the first time it is coded, turns on the &PIB (47) global bit which activates the IDFDSP inner macro. To turn off the global bit, code another FDDSPLY macro after you have displayed all the information you want. If you do not code another FDDSPLY macro, the &PIB (47) bit is turned off automatically at the end of the assembly.

IDFDSP inner macros are already scattered within the internal code waiting to be invoked by FDDSPLY. To add more IDFDSP inner macros, use either the IEBUPDTE utility program for OS or the MAINT library maintenance program for DOS. Refer to the *IBM System/360 OS Utility* publication, GC28-6586 for the way to specify the IEBUPDTE utility. To discover how to use the MAINT program, refer to the *IBM System/360 DOS System Control and System Service Program*, GC24-5036, publication.

The format for the IDFDSP inner macro is as follows:

```
IDFDSP { 'string' } , { 'string' } , ...
        { vname }   , { vname }
        { QUEUE }   , { QUEUE }
```

where

- 'string' specifies a character string that is emitted during the display. This string may be used to indicate the position where the IDFDSP inner macro is inserted.
- vname is the name without the ampersand of a global variable to be displayed. If the variable has only one character (i.e. A, B, etc.), all of the one-character variables are displayed. If the variable requested has a companion variable (i.e., &PIA and &PIB), the companion variable is also displayed.
- QUEUE specifies that all queue variables are to be displayed.

There is no limit to the number of suboperands that can be coded on the IDFDSP inner macro for the H assembler. However, the D assembler can handle only 100 suboperands and the F assembler can handle only 200 suboperands.

The IDFDSP inner macro generates MNOTE messages displaying the information requested for each macro.

*Examples:*

#### 1. IDFDSP 'AFTER GOTO', QUEUE, PRTA

This example specifies that the display is to begin where ever an IDFDSP inner macro is found, no matter what is

coded in the 'string' operand. All of the queue variables are to be displayed, along with the &PRTA global variable.

#### 2. IDFDSP 'AFTER PAGE CALCULATIONS', DFA, PRTA, A

This format requests a display of the &DFA, &PRTA, and &A global variables and needs to be inserted into the inner macro code after the page calculations. All of the one-character variables are displayed because a display of the one-character &A variable was specified.

### FDTRACE MACRO

The FDTRACE macro requests a trace of the entry to and exit from each outer and inner macro that is called during FD macro execution after an FDTRACE macro in the source deck. FDTRACE should be specified:

FDTRACE

The FDTRACE macro, the first time it is coded, turns on the &PIB (48) global bit, which activates the tracing functions. To turn off the bit, code another FDTRACE macro at the end of the area you want to trace. If you do not code another FDTRACE macro, the &PIB (48) bit is turned off automatically at the end of the assembly.

FDTRACE generates an MNOTE message to display the trace information:

```
IDF100 IN TRACE MODE {ENTERING} macro
                          {LEAVING }
```

*Example:* If you wish to display some variables and trace the macro processing of an FDCTRL outer macro, code the following statements.

```
FDDSPLY
FDTRACE
FDCTRL GOTO
FDDSPLY
FDTRACE
```

The first FDDSPLY turns on the &PIB (47) global bit and activates the IDFDSP inner macros in the FDCTRL macro. The following FDTRACE macro turns on the &PIB (48) bit to begin the tracing activity. The FDCTRL outer macro processing is traced and all appropriate global variables specified on IDFDSP inner macros are displayed. When the FDCTRL GOTO processing is finished, the second FDDSPLY and FDTRACE macros turn off the &PIB (47) and &PIB (48) bits, respectively, to stop the display and the trace. If the second pair of diagnostic macros are not specified, the display and trace continue until the end of the assembly.

## Appendix C: Diagnostic Messages

### MNOTE MESSAGES

The descriptive and diagnostic MNOTE messages that can be generated by the Form Description macro instructions are described in this section. The general format of the message presentation is as follows:

severity code (\*for descriptive messages), 'message text'

Refer to the publication *IBM 3735 Programmer's Guide*, GC30-3001, for a detailed explanation of each message.

A severity code of "\*" indicates an informational message. Other severity codes indicate more severe errors.

The internal error message IDF999 appears when the program detects an unusual or invalid internal error. Other

lines follow this error message to give a more detailed explanation of the error.

Refer to Figure 4-1 for a list of the relationship of the &M global variable with each of the message operands.

### Informational MNOTE Messages

The informational MNOTE messages have a severity code of asterisk and relate information about system parameters. These messages include also the FD program logic MNOTE messages that relate the starting and ending of paths and segments.

#### Message

* , IDF100 IN TRACE MODE	{ ENTERING } { LEAVING }	MACRO		
* , IDF101 FORM NAME IS name			MSG	001
* , IDF102 FORM ID IS ddd			MSG	002
* , IDF103 TABS SET AT COLUMNS t1 , t2 , t3 , t4 , t5			MSG	003
* , IDF104 PAGE pp INCLUDES LINE n1			MSG	004
* ,		THROUGH n2 WITHIN THE FORM		
* , IDF105 FDEND NOT NEEDED			MSG	007
* , IDF106 CTR(d) USED AS ACCUMULATOR			MSG	019
* , IDF107 CTR(d) USED AS GENERATOR			MSG	020
* , IDF108 STARTING PATH p			MSG	027
* , IDF109 STARTING SEGMENT s			MSG	028
* , IDF110 END OF SEGMENT s			MSG	029
* , IDF111 END OF PATH p			MSG	030
* , IDF112 INDICATORS USED IN PATH p			MSG	031
* , IDF113 IND(d)			MSG	032
* , IDF114 COUNTERS USED IN PATH p			MSG	033
* , IDF115 CTR(d)			MSG	034
* , IDF116 BUFFERS USED IN PATH p			MSG	035
* , IDF117 STANDARD DEFAULTS NOT OVERRIDDEN BY FDFORM			MSG	036
* , IDF118 THIS SEGMENT ENTERED FROM SEGMENT s			MSG	039
* , IDF119 THIS PATH ENTERED FROM			MSG	040
* ,		SEGMENT s OF PATH p		
* , IDF120	{ FORM } { PAGE } { LINE } { FIELD }	LEVEL ATTRIBUTES CHANGED FROM	{ STANDARD } { FORM } { PAGE } { LINE }	MSG1

*Note:* This message appears automatically as heading information for messages IDF122 through IDF128, inclusive.

<i>Message</i>	<i>Issued By</i>	<i>Call No.</i>
*, IDF121 NO ATTRIBUTES CHANGED AT { FORM } LEVEL { PAGE } { LINE } { FIELD }	MSG1	111
*, IDF122 { WIDTH } IS dd { HEIGHT } { MRGSTOP }	MSG1	120
*, IDF123 { LEFT } MARGIN IS dd { RIGHT } { TOP } { BOTTOM }	MSG1	121
*, IDF124 KIND IS { UNDEFINED } { ALPHABETIC } { NUMERIC } { KATAKANA }	MSG1	122
*, IDF125 SINK d IS { UNUSED } { TMT } { CTR (n) } { RPB, n } { PRT, n } { LPB, n } { PCH, n } { INQ, n }	MSG1	123
*, IDF126 { FILL } FOR SINK d IS { LEFT } { JUSTIFY }                  { CENTER } { UNDERLINE }              { RIGHT } { BLANK } { ZERO } { SPECIFIED } { NOT SPECIFIED }	MSG1	124
*, IDF127 SELF-CHECK OPTION IS { NOT USED } { CHECK } { MODULO 10 } { GENERATE } { MODULO 11 }	MSG1	125
*, IDF128 SOURCE IS { KEYBOARD } { REQUIRED } { AUTOEOF } { NUMPAD } { OPTIONAL } { NO AUTOEOF } { FID } { RSN } { EMITTED } { CTR (n) } { STG, n } { INQ, n } { RDR, n } { RPB, n } { LPB, n }	MSG1	126
*, IDF129 IND d { TESTED } { SET } { INVERTED }	MSG1	127
*, IDF130 { STG } { INQ } { RDR/PCH } { RPB } { LPB } { IDR } { CCR }	MSG1	130
*, IDF132 AT END OF CYCLE PRINT ELEMENT WAS *,        POSITIONED ON LINE dd OF FORM	MSG3	504
*, IDF133 NO TERMINATING ERRORS FOUND IN THIS FDP	MSG3	512
*, IDF134 SINK d OUTPUT COUNT IS digits	MSG3	573
*, IDF135 PICTURE WAS USED FOR FORMATTING *,        OUTPUT OF SINK d	MSG3	577

<i>Message</i>	<i>Issued By</i>	<i>Call No.</i>
* , IDF136 THIS SEGMENT BRANCHES TO SEGMENT ss OF PATH pp	MSG .	017
* , IDF137 type FEATURE INDICATOR TESTED	MSG1	142
* , IDF138 PACKING OPTION IS option	MSG1	129
* , IDF139 LINE NUMBER IS dd	MSG	026
* , IDF140 ind SPECIAL INDICATOR SET OR TESTED	MSG1	143
* , IDF141 POSITION LIMITS FOR LPB ARE 1 AND n	MSG	048
* , IDF142 SOURCE/SINK OPTION FOR 5496 IS RPB	MSG	049
* , IDF143 SELECTRIC II PRINT REGION BEGINS AT * , COLUMN a, ENDS AT COLUMN b	MSG	050
* , IDF144 TMT DATA FORMAT IS [ZERO OR] [a TO] b CHARACTERS [* , DELIMITED BY SEPARATOR]	MSG1	144
* , IDF145 SOURCE CHARACTER COUNT IS n	MSG3	578
* , IDF146 FORM DESCRIPTION PROGRAM SPECIFIED * , SELECTRIC II FORM HAVING nnn LINES [* , AND 3286 FORM HAVING nnn LINES]	MSG3	579
* , IDF147 SUMMARY OF FDP-GENERATED DATA * , ----- * , UNPACKED FDP OUTPUT = nnn BLOCKS * , 3735 DISK STORAGE = s1 . s2 SECTORS	MSG3	580

#### Warning MNOTE Messages

The warning MNOTE messages have a severity code of zero and relate a warning that some unusual condition or parameter has been found. Check each MNOTE message to be sure that the condition or parameter is actually what was desired. These messages, used along with the path and segment messages, can assist in finding oversights and possible logic errors in the FD program.

<i>Message</i>	<i>Issued By</i>	<i>Call No.</i>
0, IDF400 FDFORM MUST START FORM	MSG	008
0, IDF401 ELEMENT n OF HTAB OPERAND INVALID	MSG	011
0, IDF402 CTR(d) MAY NOT HAVE BEEN USED AS 0, OUTPUT SINCE PRIOR INPUT	MSG	021
0, IDF403 CTR(d) MAY NOT HAVE BEEN PROPERLY LOADED 0, BEFORE CURRENT OUTPUT	MSG	022
0, IDF404 IND d MAY NOT HAVE BEEN TESTED SINCE SET	MSG	023
0, IDF405 IND d MAY NOT HAVE BEEN SET BEFORE TEST	MSG	024
0, IDF406 CTR(d) MAY NOT HAVE BEEN CLEARED 0, BEFORE FIRST INPUT	MSG	025
0, IDF407 MESSAGE USED VERTICAL SPACING	MSG	037
0, IDF408 MESSAGE USED HORIZONTAL TABS	MSG	038
0, IDF409 CHAINING IN EFFECT, keyword OPERAND IGNORED keyword - From Figure 4-1.	MSG1	100
0, IDF410 keyword IGNORED FOR DUMMY FIELD keyword - From Figure 4-1.	MSG1	101
0, IDF411 SUBOPERANDS AFTER SUBOPERAND n OF 0, keyword OPERAND IGNORED keyword - From Figure 4-1.	MSG1	104
0, IDF412 EXCESS CHARACTERS OF keyword 0, SUBOPERAND n IGNORED keyword - From Figure 4-1.	MSG1	105





<i>Message</i>	<i>Issued By</i>	<i>Call No.</i>
0, IDF445 KIND SET TO NUMERIC BY IND OPERAND	MSG3	563
0, IDF446 PICTURE OPERAND IGNORED, SINK d NULL OR CTR	MSG3	567
0, IDF447 POSSIBLE OVERLAY OF SINK d	MSG3	574
0, IDF448 FIRST OPERATION ON ( STG ) BUFFER MAY NOT HAVE ( INQ ) ( RDR/PCH ) ( RPB ) ( LPB ) ( IDR ) ( CCR )	MSG1	132
0,           BEEN UNCONDITIONAL CLEAR OR INPUT		
0, IDF449 ( STG ) BUFFER MAY HAVE BEEN CLEARED ( INQ ) ( RDR/PCH ) ( RPB ) ( LPB ) ( IDR ) ( CCR )	MSG1	133
0,           OR INPUT WITHOUT PRIOR OUTPUT		
0, IDF450 ( STG ) BUFFER MAY HAVE BEEN OUTPUT ( INQ ) ( RDR/PCH ) ( RPB ) ( LPB ) ( IDR ) ( CCR )	MSG1	134
0,           WITHOUT PRIOR INPUT		
0, IDF451 FIRST OPERATION AFFECTING ind INDICATOR	MSG1	140
0,           WAS NOT UNCONDITIONAL CLEAR OR SEND		
0, IDF452 ind INDICATOR MAY HAVE BEEN	MSG1	141
0,           CLEARED WITHOUT PRIOR TEST		
0, IDF453 SAVELOC IN SUMMARY BLOCK IGNORED	MSG	005
0, IDF454 NAME OMITTED, SAVELOC IGNORED	MSG	006
0, IDF455 SAVELOC COUNT NOT BETWEEN d1 AND d2	MSG	009
0,           ASSUME SAVELOC=YES		
0, IDF456 { READ } COMMAND MAY NOT HAVE BEEN ISSUED { SEND }	MSG1	140
0,           WITHOUT PRIOR TEST OF { EOF(RDR) } INDICATOR { TIMEOUT }		
0, IDF457 { EOF(RDR) } INDICATOR MAY HAVE BEEN TESTED { TIMEOUT }	MSG1	141
0,           WITHOUT PRIOR { READ } COMMAND { SEND }		
0, IDF458 KIND SET TO NUMERIC BY PICTURE OPERAND	MSG3	525
0, IDF459 SAVELOC'D MACRO macro name	MSG3	581
0,           nn UNUSED REFERENCES		

#### Termination MNOTE Messages

The termination MNOTE messages have a severity code of eight and indicate that a severe error that suppresses the generation of the FD program has been found. The actual assembly process terminates. Although the assembly ends, the checking of operands continues as though no error had been found. An MNOTE message with a severity code of eight, if issued during the assembly of an FD program, flags the FD program as invalid. If this invalid FD program is used as input to the FD utility, the utility automatically

rejects the program. All errors associated with termination MNOTE messages must be corrected and the FD program assembled again before a valid FD program is generated.

<i>Message</i>	<i>Issued By</i>	<i>Call No.</i>
8, IDF700 MANDATORY FID OPERAND OMITTED	MSG	010
8, IDF701 FORM NAME INVALID; subname SUBSTITUTED	MSG	012
8, IDF702 INVALID BRANCH	MSG	013
8, IDF703 PREVIOUS FORM NOT PROPERLY TERMINATED	MSG	014
8, IDF704 PAGE WITHIN CYCLE IGNORED	MSG	015
8, IDF705 FORM ENDED BEFORE CYCLE LIMIT ENCOUNTERED	MSG	016
8, IDF706 EXPECTED CHAINING OF PRECEDING MACRO 0, NOT FOUND, CHAINING TERMINATED	MSG	018
8, IDF707 COMMAND GROUP n, ILLEGAL USE OF CLEAR	MSG	041
8, IDF708 SKIPTO COMMAND ILLEGAL IN CYCLE OR SUMMARY	MSG	042
8, IDF709 COMMAND GROUP n, SKIP OR SKIPTO NONDECIMAL	MSG	043
8, IDF711 COMMAND GROUP n, PRINT ILLEGAL AFTER CLEAR	MSG	045
8, IDF712 COMMAND GROUP n, ILLEGAL CLEAR OR READ	MSG	046
8, IDF713 COMMAND GROUP n, ILLEGAL DUE TO 0, SPECIFICATION OF m DEVICE TYPES	MSG	047
8, IDF714 EXPECTED CHAINING OF keyword OPERAND 0, NOT FOUND, CHAINING TERMINATED keyword - From Figure 4-1.	MSG1	102
8, IDF715 CHARACTER NEAR POSITION p OF keyword 0, OPERAND IS ILLEGAL keyword - From Figure 4-1.	MSG1	103
8, IDF716 $\left. \begin{array}{l} \text{FDFORM} \\ \text{FDPAGE} \\ \text{FDLINE} \\ \text{FDFIELD} \end{array} \right\}$ MUST FOLLOW $\left. \begin{array}{l} \text{FDFORM} \\ \text{FDPAGE} \\ \text{FDLINE} \\ \text{FDFIELD} \end{array} \right\}$	MSG1	112
8, IDF717 keyword OPERAND INVALID keyword - From Figure 4-1.	MSG2	200
8, IDF718 keyword OPERAND OMITTED keyword - From Figure 4-1.	MSG2	202
8, IDF719 keyword OPERAND 0, NOT BETWEEN a AND b keyword - From Figure 4-1.	MSG2	201
8, IDF720 BATCH FOR keyword SUBOPERAND n INVALID keyword - From Figure 4-1.	MSG2	212
8, IDF721 FID FOR keyword SUBOPERAND n INVALID keyword - From Figure 4-1.	MSG2	213
8, IDF722 CTR FOR keyword SUBOPERAND n INVALID keyword - From Figure 4-1.	MSG2	214
8, IDF723 IND FOR keyword SUBOPERAND n INVALID keyword - From Figure 4-1.	MSG2	215
8, IDF724 EOF FOR keyword SUBOPERAND n INVALID keyword - From Figure 4-1.	MSG2	216
8, IDF725 OPERATOR FOR keyword SUBOPERAND n INVALID keyword - From Figure 4-1.	MSG2	217
8, IDF726 EMIT FOR keyword SUBOPERAND n INVALID keyword - From Figure 4-1.	MSG2	218
8, IDF727 BATCH FOR keyword SUBOPERAND n NOT 0, BETWEEN a AND b keyword - From Figure 4-1.	MSG2	252

<i>Message</i>	<i>Issued By</i>	<i>Call No.</i>
8, IDF728 FID FOR keyword SUBOPERAND n NOT 0, BETWEEN a AND b keyword - From Figure 4-1.	MSG2	253
8, IDF729 CTR FOR keyword SUBOPERAND n NOT 0, BETWEEN a AND b keyword - From Figure 4-1.	MSG2	254
8, IDF730 IND FOR keyword SUBOPERAND n NOT 0, BETWEEN a AND b keyword - From Figure 4-1.	MSG2	255
8, IDF732 SKIP FOR keyword SUBOPERAND n NOT 0, BETWEEN a AND b keyword - From Figure 4-1.	MSG2	257
8, IDF733 INVALID CHARACTER IN MESSAGE SUBOPERAND n	MSG3	500
8, IDF734 ATTEMPTED MOVEMENT TO A PREVIOUSLY 0, DEFINED LINE INVALID	MSG3	505
8, IDF735 CYCLE COUNT INVALID, COUNT OF 1 ASSUMED	MSG3	508
8, IDF736 CYCLE COUNT NOT BETWEEN a AND b 0, COUNT OF 1 ASSUMED	MSG3	509
8, IDF737 TOO MANY UNRESOLVED BRANCHES	MSG3	510
8, IDF738 INVALID FORM DESCRIPTION PROGRAM	MSG3	513
8, IDF739 DOCUMENT FIELD LNG(d) IS NONDECIMAL	MSG3	514
8, IDF740 DEAD CODE, MACRO IGNORED	MSG3	516
8, IDF741 SOURCE KEYBOARD OPTIONS INVALID	MSG3	519
8, IDF742 SOURCE START OR END POSITION INVALID	MSG3	520
8, IDF743 SOURCE START OR END POSITION NOT 0, WITHIN THE RANGE a TO b	MSG3	521
8, IDF744 SOURCE LENGTH SPECIFICATION INVALID	MSG3	522
8, IDF745 SOURCE LENGTH NOT BETWEEN a AND b	MSG3	523
8, IDF746 MAX/EXACT COUNT NOT BETWEEN a AND b	MSG3	526
8, IDF747 MINIMUM COUNT NOT BETWEEN a AND b	MSG3	527
8, IDF748 START POSITION FOR SINK d INVALID	MSG3	529
8, IDF749 START POSITION FOR SINK d 0, NOT BETWEEN a AND b	MSG3	530
8, IDF750 LNG(d) FOR SINK d INVALID	MSG3	531
8, IDF751 LNG(d) FOR SINK d NOT BETWEEN a AND b	MSG3	532
8, IDF752 END POSITION FOR SINK d INVALID	MSG3	533
8, IDF753 END POSITION FOR SINK d 0, NOT BETWEEN a AND b	MSG3	534
8, IDF754 NUMBER OF EMITTED "STRING" CHARACTERS 0, NOT BETWEEN a AND b	MSG3	536
8, IDF755 NUMBER OF CHARACTERS IN COMPARAND OF COMPARE 0, SUBOPERAND n NOT BETWEEN a AND b	MSG3	538
8, IDF756 INVALID ARITHMETIC OPERATOR IN CTR 0, SUBOPERAND n	MSG3	542
8, IDF757 INVALID COMPARAND LENGTH IN IND OPERAND	MSG3	544
8, IDF758 UNRESOLVED BRANCH TO name 0, FROM PATH p SEGMENT s	MSG3	546
8, IDF759 LOGICAL OPERATOR NEAR POSITION p OF 0, IND SUBOPERAND n INVALID	MSG3	560
8, IDF760 COMPARISON OPERATOR NEAR POSITION p OF 0, IND SUBOPERAND n INVALID	MSG3	561

<i>Message</i>	<i>Issued By</i>	<i>Call No.</i>
8, IDF761 COMPARAND CHARACTER NEAR POSITION p OF 0, IND SUBOPERAND n INVALID	MSG3	562
8, IDF762 IND COMPARAND LENGTH NOT BETWEEN 1 AND 127	MSG3	564
8, IDF763 PICTURE ILLEGAL WITH EMITTED SOURCE	MSG3	565
8, IDF764 PICTURE ILLEGAL WITH NON-NUMERIC COMPARISONS	MSG3	566
8, IDF765 LENGTH SPECIFICATION FOR SINK d IS INADEQUATE	MSG3	568
8, IDF766 PICTURE SUBOPERAND n IMPROPERLY FRAMED	MSG3	569
8, IDF767 CHARACTER c OF PICTURE SUBOPERAND n IS 0, INVALID, MUST BE ONE OF THE FOLLOWING 0, characters	MSG3	570
8, IDF768 PICTURE SUBOPERAND n NOT PROPERLY TERMINATED	MSG3	571
8, IDF769 SINK COUNT NOT BETWEEN a AND b	MSG3	572
8, IDF770 PRINT ELEMENT POSITION CANNOT BE DETERMINED 0, COUNT MUST BE CODED	MSG3	575
8, IDF771 PRINTING SINK EXCEEDS FIELD MARGINS	MSG3	576
8, IDF772 keyword OPERAND, SUBOPERAND n, FORMAT INVALID keyword - From Figure 4-1.	MSG3	517
8, IDF773 keyword OPERAND, SUBOPERAND n, NOT AN ALLOWED EXACT VALUE 0, OR NOT BETWEEN a AND b keyword - From Figure 4-1.	MSG3	518
8, IDF774 COMMAND GROUP n, INVALID FORMAT	MSG1	150
8, IDF775 COMMAND GROUP n, SKIP OR SKIPTO VALUE 0, NOT BETWEEN a AND b	MSG1	151
8, IDF999 FDM SYSTEM ERROR	MSG	000

*Note:* The four message inner macros MSG, MSG1, MSG2, and MSG3 also issue this message in-line.

<i>&amp;M</i>	<i>Structural Operand Name</i>	<i>Control<sup>1</sup> Operand Name</i>	<i>&amp;M</i>	<i>Structural Operand Name</i>	<i>Control<sup>1</sup> Operand Name</i>
1	SOURCE	IF	17	COMPARE	
2	KIND	IND	18	CTR	
3	SELFCHK	CTR	19	IND	
4	SINK	COMMAND	20	PICTURE	
5	FILL	TOTAL	21	BATCH	
6	JUSTIFY	GOTO	22	Field right margin <sup>3</sup>	
7	UL (underline)		23	MESSAGE	
8	WIDTH		24	HTAB	
9	HMRG		25	FID	
10	SAVELOC	SAVELOC	26	PACKING	
11	NAME (macro name)	NAME	27	MRGSTOP	
12	Page or line number <sup>2</sup> Field left margin <sup>3</sup>		28-59	Reserved	
13	CYCLE	CYCLE			
14	HEIGHT				
15	VMRG				
16	COUNT				

Figure 4-1. &M Operand Relationship for Keywords

## FD PROGRAM ERROR MESSAGE

When the terminal control program in the 3735 detects an invalid FCD byte, it issues an FD program error message.

The message format is as follows:

FDP ERROR TT BB AA AA

where

TT is the type of byte in error:

- 00 - immediate byte;
- 01 - data source byte;
- 02 - data type byte;
- 03 - function byte;
- 04 - data sink byte;
- 05 - end control byte;

BB is the actual byte in error;

AA AA are the two bytes that contain the address of the error byte relative to the beginning of the FD program sector.

To correct this error, the following steps must be performed:

1. Determine if the byte represented by BB is in error. If the byte is incorrect, check whether missing or extra bytes in the FD program are causing the terminal control program to interpret BB incorrectly. (For example, a missing end control byte would cause the first byte of the next FCD to be interpreted as an end control byte and not as the first byte in the FCD.)
2. Note the point in the form description where the error occurs (taking into account any possible branches to that point) by examining a PRINT GEN assembly listing of the FD program. Then modify the source level encoding to bypass the problem. Refer to the discussion of FD program maintenance in Part 2, Section 3 for further correction suggestions.

## DATA SOURCE ERROR MESSAGE

When the terminal control program in the 3735 detects an error in a data field that has a source specified other than

the keyboard, it issues the data source error message. The message format is as follows:

DATA SOURCE ERROR SS EE DT BB

where

SS specifies the data source type that is in error:

- 50 - FD program ID (FID);
- 51 - Record number;
- 52 - Emitted data;
- 53 - Counters;
- 54 - Storage (STG) buffer;
- 55 - Inquiry (INQ) buffer;
- 56 - 5496 buffer;
- 57 - 3286-3 buffer;
- 5F - Operator Identification Card Reader (IDR or CCR) buffer;
- 7F - CPU data;

EE specifies the type of error:

- 00 - Character type;
- 05 - Value (range) check;
- 10 - Self check;
- 20 - Length error.

When EE is 00 specifying a character byte, the DT and BB bytes are displayed:

DT indicates the character type for the field:

- 00 - data is undefined;
- 10 - data is numeric only;
- 20 - data is alphabetic only;
- 30 - data is alphameric;
- 40 - data is Katakana;

BB indicates the character that caused the error.

To correct this error, the user must check the coding of his outer FD macro statements to determine whether he has an invalid encoding. This error occurs as a result of specifying SOURCE=storage buffer when the data in the buffer is not character data or as a result of the source counter not meeting the COMPARE test specified.

## Appendix D: Sample FD Program

This appendix contains three sections: excerpts from three code sources to illustrate the correlation between (1) a PRINT GEN macro assembly listing, (2) a hexadecimal dump of the data created by the assembly, and (3) the internal 3735 code created. This information may aid in finding and solving problems in the assembly of FD programs.

The first section contains portions of a PRINT GEN macro assembly listing composed of excerpts from the sample FD macro program listed in Appendix C in the *IBM 3735 Programmer's Guide*, GC30-3001. This sample program describes the sample form presented in Appendix A in the *IBM 3735 Programmable Buffered Terminal Concept and Application* publication, GA27-3043. The assembly listing contains alphabetic identification characters to relate specific parts of the listing to corresponding parts of the hexadecimal dump and the 3735 code that follow. Each alphabetic character and the portion of the FD program that it identifies is as follows:

- A** the form ID specified in the FID operand on the FDFORM macro.
- B** the data condensation specified in the PACKING operand on the FDFORM macro.
- C** the form length specified in the HEIGHT operand.
- D** the begin tab definition delimiter.
- E** the begin the FCD indicator.
- F** the specifications resulting from the coding of the FDFIELD macro named ADDRESS1.
- G** the specifications resulting from the coding of an FDFIELD macro for control-character input.
- H** the specifications resulting from the coding of the FDFIELD macro named DATEFLD. This FDFIELD macro illustrates a PICTURE specification. The macro also shows how the assembly uses ORG statements to go back and overlay parts of the object code with later form specifications. In this case, the data-type byte and the function

byte are initially assembled as X'4175' and X'4070' and then replaced by X'4171' and X'4170' via two ORG statements.

- I** the specifications resulting from an FDCTRL macro with a GOTO branch that is dependent on the setting of an indicator in IND(2). If IND=OFF, this FDCTRL macro branches to the next FDCTRL macro. If IND=ON, the branch is to the FDFIELD macro named DELIVER.
- J** the specifications resulting from the FDLINE macro named BODY illustrating an example of a cycle.
- K** the repeat cycle indicator.
- L** the end cycle indicator specified in the FDLINE macro named GROSS.
- M** the end form indicator in the CSECT named IJLF1001.

The hexadecimal dump in the second section of this appendix results from using one of the BTAM (OS or DOS) sample programs described in Appendix G or Appendix H in the *IBM 3735 Programmer's Guide*, GC30-3001. Each of these programs reads data from the 3735, dumps it on the system printer if requested to do so, and then sends FD programs (if any) to the 3735. When through, the program sends the terminate communicate mode message to the 3735, issues a Write Disconnect macro, and concludes processing. This dump contains alphabetic identification characters that match those in the PRINT GEN assembly listing to point to the fields generated by that macro assembly.

The internal 3735 code in the third section of this appendix also contains alphabetic identification characters corresponding to those in the assembly listing and the hexadecimal dump. These characters further illustrate the individual specifications defined in the excerpts from the sample FD program in Appendix C in the *IBM 3735 Programmer's Guide*, GC30-3001.

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
				3	PDFORM PDFORM FID='026', FDP NO IS 026 X
					MRGSTOP=11, MECHANICAL LEFT MARGIN STOP X
					MESSAGE='USE FORM 786425. FDP 011 MUST BE PERFORMED BEFX
					ORE THIS FDP.', OPERATOR INSTRUCTION X
					HTAB=(33,64), HORIZONTAL TAB STOP POSITIONS X
					PACKING=DELIMIT DELETE TRAILING BLANKS, ADD X
					FS CHARACTER TO EACH FIELD
				4**	***** FD MACROS CHANGE LEVEL 02/04/72, 1100 *****
				5	*
				6	*,IJLFP101 FORM NAME IS PDFORM
000000				7+	IJLFP1000 START 0
000000	C6C4C6D6D9D44040			8+	DC CL8'PDFORM' SECTOR HEADER - FORMNAME
000008	0000			9+	DC H'0' SECTOR NUMBER
00000A	4070			10+	DC X'4070' RESERVED SECTOR CHAIN
0001EC				11+	ORG **480 SET VALID
00000C				12+	ORG *-480 LENGTH ATTRIBUTE
00000C	4070			13+	DC X'4070' GENERATED CONSTANT
00000C				14+	ORG IJLFP1000+12 ORG TO DESIRED LOCATION TO ASH NEXT BYTE
00000C	4370			15+	DC H'17264' GENERATED ARITHMETIC
00000E	4372			16+	DC H'17266' GENERATED ARITHMETIC
000010	4376			17+	DC H'17270' GENERATED ARITHMETIC
				18	*,IJLFP102 FORM ID IS 026
000008				19+	ORG IJLFP1000+8 ORG TO DESIRED LOCATION TO ASH NEXT BYTE
000008	FFFE			20+	DC X'FFFE' GENERATED CONSTANT
00000A				21+	PDFORM EQU * **TEST FOR DUPLICATE FORM NAMES IN THIS ASSEMBLY ***
000012				22+	ORG IJLFP1000+18 ORG TO ASH NEXT BYTE AFTER HIGHEST
000012	4470			23+	DC H'17520' GENERATED ARITHMETIC
				24	*,IJLFP138 PACKING OPTION IS 'DELIMIT'
000014	4070			25+	DC X'4070' GENERATED CONSTANT
000016	4070			26+	DC X'4070' GENERATED CONSTANT
000018	4070			27+	DC X'4070' GENERATED CONSTANT
00001A	4070			28+	DC X'4070' GENERATED CONSTANT
				29	*,IJLFP122 MRGSTOP IS 11
00001C	4575			30+	DC X'4575' TRANSLATED CHARACTER
00001E	4573			31+	DC X'4573' TRANSLATED CHARACTER
000020	4475			32+	DC X'4475' TRANSLATED CHARACTER
000022	4270			33+	DC X'4270' TRANSLATED CHARACTER
000024	4476			34+	DC X'4476' TRANSLATED CHARACTER
000026	447F			35+	DC X'447F' TRANSLATED CHARACTER
000028	4572			36+	DC X'4572' TRANSLATED CHARACTER
00002A	447D			37+	DC X'447D' TRANSLATED CHARACTER
00002C	4270			38+	DC X'4270' TRANSLATED CHARACTER
00002E	4377			39+	DC X'4377' TRANSLATED CHARACTER
000030	4378			40+	DC X'4378' TRANSLATED CHARACTER
000032	4376			41+	DC X'4376' TRANSLATED CHARACTER
000034	4374			42+	DC X'4374' TRANSLATED CHARACTER
000036	4372			43+	DC X'4372' TRANSLATED CHARACTER
000038	4375			44+	DC X'4375' TRANSLATED CHARACTER
00003A	427E			45+	DC X'427E' TRANSLATED CHARACTER
00003C	4270			46+	DC X'4270' TRANSLATED CHARACTER
00003E	4270			47+	DC X'4270' TRANSLATED CHARACTER
000040	4476			48+	DC X'4476' TRANSLATED CHARACTER
000042	4474			49+	DC X'4474' TRANSLATED CHARACTER
000044	4570			50+	DC X'4570' TRANSLATED CHARACTER
000046	4270			51+	DC X'4270' TRANSLATED CHARACTER
000048	4370			52+	DC X'4370' TRANSLATED CHARACTER
00004A	4371			53+	DC X'4371' TRANSLATED CHARACTER
00004C	4371			54+	DC X'4371' TRANSLATED CHARACTER
00004E	4270			55+	DC X'4270' TRANSLATED CHARACTER
000050	447D			56+	DC X'447D' TRANSLATED CHARACTER
000052	4575			57+	DC X'4575' TRANSLATED CHARACTER
000054	4573			58+	DC X'4573' TRANSLATED CHARACTER
000056	4574			59+	DC X'4574' TRANSLATED CHARACTER
000058	4270			60+	DC X'4270' TRANSLATED CHARACTER
00005A	4472			61+	DC X'4472' TRANSLATED CHARACTER
00005C	4475			62+	DC X'4475' TRANSLATED CHARACTER
00005E	4270			63+	DC X'4270' TRANSLATED CHARACTER
000060	4570			64+	DC X'4570' TRANSLATED CHARACTER
000062	4475			65+	DC X'4475' TRANSLATED CHARACTER
000064	4572			66+	DC X'4572' TRANSLATED CHARACTER
000066	4476			67+	DC X'4476' TRANSLATED CHARACTER
000068	447F			68+	DC X'447F' TRANSLATED CHARACTER
00006A	4572			69+	DC X'4572' TRANSLATED CHARACTER
00006C	447D			70+	DC X'447D' TRANSLATED CHARACTER
00006E	4475			71+	DC X'4475' TRANSLATED CHARACTER
000070	4474			72+	DC X'4474' TRANSLATED CHARACTER
000072	4270			73+	DC X'4270' TRANSLATED CHARACTER
000074	4472			74+	DC X'4472' TRANSLATED CHARACTER
000076	4475			75+	DC X'4475' TRANSLATED CHARACTER
000078	4476			76+	DC X'4476' TRANSLATED CHARACTER
00007A	447F			77+	DC X'447F' TRANSLATED CHARACTER
00007C	4572			78+	DC X'4572' TRANSLATED CHARACTER
00007E	4475			79+	DC X'4475' TRANSLATED CHARACTER
000080	4270			80+	DC X'4270' TRANSLATED CHARACTER
000082	4574			81+	DC X'4574' TRANSLATED CHARACTER

**A**

**B**

**C**



LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
000084	4478			82+	DC X'4478' TRANSLATED CHARACTER
000086	4479			83+	DC X'4479' TRANSLATED CHARACTER
000088	4573			84+	DC X'4573' TRANSLATED CHARACTER
00008A	4270			85+	DC X'4270' TRANSLATED CHARACTER
00008C	4476			86+	DC X'4476' TRANSLATED CHARACTER
00008E	4474			87+	DC X'4474' TRANSLATED CHARACTER
000090	4570			88+	DC X'4570' TRANSLATED CHARACTER
000092	427E			89+	DC X'427E' TRANSLATED CHARACTER
000094	477F			90+	DC X'477F' START TAB DEFINITION BYTES] <b>D</b>
000096	4175			91+	DC H'16757' GENERATED ARITHMETIC
000098	417F			92+	DC H'16767' GENERATED ARITHMETIC
				93	*,IJLP103 TABS SET AT COLUMNS 33, 64
00009A	4070			94+	DC X'4070' GENERATED CONSTANT] <b>E</b>
				95	*
				96	*,IJLP108 STARTING PATH 1
				97	*
				98	*,IJLP109 STARTING SEGMENT 1

144 \*,IJLP122 WIDTH IS 85  
 145 \*,IJLP123 LEFT MARGIN IS COLUMN 12  
 146 \*,IJLP123 RIGHT MARGIN IS COLUMN 80

148	NAME1	FD	FIELD 12,31,	NAME POSITION	X
			SOURCE=(RDR,1,20),	GET FROM CARD IMAGE	X
			SINK=PRT	PUT TO SELECTRIC II	
149	**FD	FIELD	-360N-CQ-490 - CHANGE LEVEL 3-A		
150	NAME1	EQU	* *** TEST FOR DUPLICATE NAMES ***		
0000B6					
0000B6	4576				
0000B8	4070				
0000BA	4074				
0000BC	4074				
0000BE	4174				
151+		DC	H'17782' GENERATED ARITHMETIC		
152+		DC	H'16496' GENERATED ARITHMETIC		
153+		DC	H'16500' GENERATED ARITHMETIC		
154+		DC	H'16500' GENERATED ARITHMETIC		
155+		DC	H'16756' GENERATED ARITHMETIC		
156			*		
157			*,IJLP125 SINK 1 IS PRT		
158			*,IJLP143 SELECTRIC II PRINT REGION BEGINS AT		
159			*, COLUMN 12, ENDS AT COLUMN 31		
160			*		
161			*,IJLP128 SOURCE IS RDR, 1		
162			*,IJLP145 SOURCE CHARACTER COUNT IS 20		
163			*,IJLP124 KIND IS UNRESTRICTED		

165 FDLINE 13 SOLD-TO STREET ADDRESS LINE  
 166\*\*FDLINE -360N-CQ-490 - CHANGE LEVEL 3-A  
 167 \*  
 168 \*,IJLP139 LINE NUMBER IS 13  
 169 \*,IJLP122 WIDTH IS 85  
 170 \*,IJLP123 LEFT MARGIN IS COLUMN 12  
 171 \*,IJLP123 RIGHT MARGIN IS COLUMN 80

173	ADDRESS1	FD	FIELD 12,31,	STREET ADDRESS POSITION	X
			SOURCE=(RDR,21,40),	GET FROM CARD IMAGE	X
			SINK=PRT	PUT TO SELECTRIC II	
174	**FD	FIELD	-360N-CQ-490 - CHANGE LEVEL 3-A		
175	ADDRESS1	EQU	* *** TEST FOR DUPLICATE NAMES ***		
0000C0					
0000C0	4471				
0000C2	4576				
0000C4	4070				
0000C6	4178				
0000C8	4074				
0000CA	4174				
176+		DC	H'17521' GENERATED ARITHMETIC		
177+		DC	H'17782' GENERATED ARITHMETIC		
178+		DC	H'16496' GENERATED ARITHMETIC		
179+		DC	H'16760' GENERATED ARITHMETIC		
180+		DC	H'16500' GENERATED ARITHMETIC		
181+		DC	H'16756' GENERATED ARITHMETIC		
182			*		
183			*,IJLP125 SINK 1 IS PRT		
184			*,IJLP143 SELECTRIC II PRINT REGION BEGINS AT		
185			*, COLUMN 12, ENDS AT COLUMN 31		
186			*		
187			*,IJLP128 SOURCE IS RDR, 21		
188			*,IJLP145 SOURCE CHARACTER COUNT IS 20		
189			*,IJLP124 KIND IS UNRESTRICTED		
191		FD	LINE 14	SOLD-TO CITY/STATE/ZIP LINE	
192	**FD	LINE	-360N-CQ-490 - CHANGE LEVEL 3-A		
193			*		
194			*,IJLP139 LINE NUMBER IS 14		
195			*,IJLP122 WIDTH IS 85		
196			*,IJLP123 LEFT MARGIN IS COLUMN 12		
197			*,IJLP123 RIGHT MARGIN IS COLUMN 80		

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT		
				237	PDFIELD , SOURCE=(KBD,OPTIONAL) , COUNT=1, COMPARE=(FIELD,EQ,'K') , IND=(1,FIELD,EQ,'K') , SINK=NULL	CONTROL-CHARACTER INPUT GET FROM KEYBOARD SINGLE CHARACTER IF ENTRY MADE, MUST BE 'K' FLAG ENTRY OF 'K' NO OUTPUT	X X X X X
				238**PDFIELD	-360N-CQ-490 - CHANGE LEVEL 3-A		
0000E8	4070			239+	DC X'4070' GENERATED CONSTANT		
0000EA	4073			240+	DC H'16499' GENERATED ARITHMETIC		
0000EC	4071			241+	DC H'16497' GENERATED ARITHMETIC		
0000EE	427A			242+	DC H'17018' GENERATED ARITHMETIC		
0000F0	4070			243+	DC H'16496' GENERATED ARITHMETIC		
0000F2	4270			244+	DC H'17008' GENERATED ARITHMETIC		
0000F4	4071			245+	DC H'16497' GENERATED ARITHMETIC		
0000F6	447B			246+	DC X'447B' TRANSLATED CHARACTER		
0000F8	4270			247+	DC H'17008' GENERATED ARITHMETIC		
0000FA	4071			248+	DC H'16497' GENERATED ARITHMETIC		
				249**		TRANSLATE	
0000FC	447B			250+	DC X'447B' TRANSLATED CHARACTER		
				251**		END TRANSLATE	
				252		*,IJL129 IND 1 SET	
0000F0				253+	ORG IJL1000+240 ORG TO DESIRED LOCATION TO ASM NEXT BYTE		
0000F0	4270			254+	DC H'17008' GENERATED ARITHMETIC		
0000FE				255+	ORG IJL1000+254 ORG TO ASM NEXT BYTE AFTER HIGHEST		
0000FE	4070			256+	DC H'16496' GENERATED ARITHMETIC		
				257		*	
				258		*,IJL128 SOURCE IS KEYBOARD, OPTIONAL, NO AUTOEOF	
				259		*,IJL145 SOURCE CHARACTER COUNT IS ZERO OR 1	
				260		*,IJL124 KIND IS UNRESTRICTED	
				262	FDCTRL IF=IND(1),GOTO=KSHIPTO ON MEANS OPR. WILL KEY SHIP-TO		
				263**FDCTRL	-360N-CQ-490 - CHANGE LEVEL 3-A		

G

				521	HEADLINE	FDLINE 23	HEADING LINE	
				522**FDLINE	-360N-CQ-490 - CHANGE LEVEL 3-A			
00019A				523+HEADLINE	EQU * *** TEST FOR DUPLICATE NAMES ***			
				524		*		
				525		*,IJL110 END OF SEGMENT 3		
				526		*,IJL109 STARTING SEGMENT 4		
				527		*		
				528		*		
				529		*,IJL139 LINE NUMBER IS 23		
				530		*,IJL122 WIDTH IS 85		
				531		*,IJL123 LEFT MARGIN IS COLUMN 12		
				532		*,IJL123 RIGHT MARGIN IS COLUMN 80		
				534	DATEFLD	PDFIELD 12,19,	DATE POSITION	
						SOURCE=(STG,1,6) ,	GET FROM STORAGE (PRESET)	
						SINK=PRT,	PUT TO SELECTRIC II	
						PICTURE='Y9/99/99',	MM/DD/YY FORMAT	
						KIND=N		
				535**PDFIELD	-360N-CQ-490 - CHANGE LEVEL 3-A			
00019A				536+DATEFLD	EQU * *** TEST FOR DUPLICATE NAMES ***			
00015E				537+	ORG IJL1000+350 ORG TO DESIRED LOCATION TO ASM NEXT BYTE			
00015E	4770			538+	DC X'4770' GENERATED CONSTANT			
000160	4475			539+	DC H'17525' GENERATED ARITHMETIC			
00019A				540+	ORG IJL1000+410 ORG TO ASM NEXT BYTE AFTER HIGHEST			
00019A	4475			541+	DC H'17525' GENERATED ARITHMETIC			
				542		*		
				543		*,IJL118 THIS SEGMENT ENTERED FROM SEGMENT 3		
000162				544+	ORG IJL1000+354 ORG TO DESIRED LOCATION TO ASM NEXT BYTE			
000162	4671			545+	DC X'4671' GENERATED CONSTANT			
000164	4070			546+	DC H'16496' GENERATED ARITHMETIC			
000166	417A			547+	DC H'16762' GENERATED ARITHMETIC			
000168	4670			548+	DC X'4670' GENERATED CONSTANT			
00016A	4670			549+	DC X'4670' GENERATED CONSTANT			
00016C	4670			550+	DC X'4670' GENERATED CONSTANT			
00016E	4670			551+	DC X'4670' GENERATED CONSTANT			
000170	4670			552+	DC X'4670' GENERATED CONSTANT			
000172	4670			553+	DC X'4670' GENERATED CONSTANT			
000174	4670			554+	DC X'4670' GENERATED CONSTANT			
000176	4670			555+	DC X'4670' GENERATED CONSTANT			
				556		*		
				557		*,IJL118 THIS SEGMENT ENTERED FROM SEGMENT 2		

H

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
00019C				558+	ORG IJLF1000+412 ORG TO ASM NEXT BYTE AFTER HIGHEST
00019C	4574			559+	DC H'17780' GENERATED ARITHMETIC
00019E	4070			560+	DC H'16496' GENERATED ARITHMETIC
0001A0	4074			561+	DC H'16500' GENERATED ARITHMETIC
				562	0,IJLF448 FIRST OPERATION ON STG BUFFER MAY NOT HAVE
				563	0, BEEN UNCONDITIONAL CLEAR OR INPUT
0001A2	4175			564+	DC H'16757' GENERATED ARITHMETIC
0001A4	4076			565+	DC H'16502' GENERATED ARITHMETIC
0001A6	4070			566+	DC H'16496' GENERATED ARITHMETIC
0001A8	4070			567+	DC H'16496' GENERATED ARITHMETIC
0001AA	407A			568+	DC H'16506' GENERATED ARITHMETIC
0001AC	4075			569+	DC H'16501' GENERATED ARITHMETIC
0001AE	4579			570+	DC X'4579' GENERATED CHARACTER
0001B0	4379			571+	DC X'4379' GENERATED CHARACTER
0001B2	427F			572+	DC X'427F' GENERATED CHARACTER
0001B4	4379			573+	DC X'4379' GENERATED CHARACTER
0001B6	4379			574+	DC X'4379' GENERATED CHARACTER
0001B8	427F			575+	DC X'427F' GENERATED CHARACTER
0001BA	4379			576+	DC X'4379' GENERATED CHARACTER
0001BC	4379			577+	DC X'4379' GENERATED CHARACTER
0001BE	477F			578+	DC X'477F' GENERATED CONSTANT
0001AC				579+	ORG IJLF1000+428 ORG TO DESIRED LOCATION TO ASM NEXT BYTE
0001AC	4076			580+	DC H'16502' GENERATED ARITHMETIC
				581	*
				582	*,IJLF125 SINK 1 IS PRT
				583	*,IJLF143 SELECTRIC II PRINT REGION BEGINS AT
				584	*, COLUMN 12, ENDS AT COLUMN 19
0001A2				585+	ORG IJLF1000+418 ORG TO DESIRED LOCATION TO ASM NEXT BYTE
0001A2	4171			586+	DC H'16753' GENERATED ARITHMETIC
0001A6				587+	ORG IJLF1000+422 ORG TO DESIRED LOCATION TO ASM NEXT BYTE
0001A6	4170			588+	DC H'16752' GENERATED ARITHMETIC
				589	*,IJLF135 PICTURE 1 WAS USED FOR FORMATTING
				590	*, OUTPUT OF SINK 1
				591	*
				592	*,IJLF128 SOURCE IS STG, 1
				593	*,IJLF145 SOURCE CHARACTER COUNT IS 6
				594	*,IJLF124 KIND IS NUMERIC

H

596 INVNO FDFIELD 21,28, INVOICE NUMBER POSITION X  
 SOURCE=(STG,7,13), GET FROM STORAGE (PRESET) X  
 CTR=(1,ADD,FIELD), PREVIOUSLY CLEARED X  
 SINK=(PRT,TMT), PUT TO SEL. II AND TMT X  
 PICTURE=('99B99999','99B99999') PICTURES SHOULD AGREE

844 \* BRANCH TABLE TO CONTROL PRINTING OF SHIP-VIA FIELD

0002AC	4070			846	FDCTRL IF=(IND(2)), ON IF 'D' KEYED X
0002AE	4672				GOTO=DELIVER
				847+*FDCTRL	-360N-CQ-490 - CHANGE LEVEL 3-A
				848+	DC X'4070' GENERATED CONSTANT
				849+	DC X'4672' GENERATED CONSTANT
				850	*,IJLF129 IND 2 TESTED
0002B0	4070			851+	DC H'16496' GENERATED ARITHMETIC
0002B2	4170			852+	DC H'16752' GENERATED ARITHMETIC
0002B4	4070			853+	DC X'4070' GENERATED CONSTANT
0002B6	407D			854+	DC X'407D' GENERATED CONSTANT
0002B8	4670			855+	DC X'4670' GENERATED CONSTANT
0002BA	4670			856+	DC X'4670' GENERATED CONSTANT
0002BC	4670			857+	DC X'4670' GENERATED CONSTANT
0002BE	4670			858+	DC X'4670' GENERATED CONSTANT
0002C0	4670			859+	DC X'4670' GENERATED CONSTANT
0002C2	4670			860+	DC X'4670' GENERATED CONSTANT
0002C4	4670			861+	DC X'4670' GENERATED CONSTANT
0002C6	4670			862+	DC X'4670' GENERATED CONSTANT
0002C8	4670			863+	DC X'4670' GENERATED CONSTANT
0002CA	4670			864+	DC X'4670' GENERATED CONSTANT
0002CC	4670			865+	DC X'4670' GENERATED CONSTANT
0002CE	4670			866+	DC X'4670' GENERATED CONSTANT
0002D0	4670			867+	DC X'4670' GENERATED CONSTANT

If IND = OFF,  
go to next FDCTRL

I

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT	
				869	FDCTRL IF=(IND(3)),	ON IF 'P' KEYED X
					GOTO=PICKUP	
				870**FDCTRL	-360N-CQ-490 - CHANGE LEVEL 3-A	
				871	*	
				872	*,IJL110 END OF SEGMENT 4	
				873	*,IJL109 STARTING SEGMENT 5	
				874	*	
				875	*	
				876	*,IJL118 THIS SEGMENT ENTERED FROM SEGMENT 4	
0002D2	4672			877+	DC X'4672' GENERATED CONSTANT	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <b>I</b>                       Use this if IND = OFF                 </div>
				878	*,IJL129 IND 3 TESTED	
0002D4	4070			879+	DC H'16496' GENERATED ARITHMETIC	
0002D6	4270			880+	DC H'17008' GENERATED ARITHMETIC	
0002D8	4070			881+	DC X'4070' GENERATED CONSTANT	
0002DA	407D			882+	DC X'407D' GENERATED CONSTANT	
0002DC	4670			883+	DC X'4670' GENERATED CONSTANT	
0002DE	4670			884+	DC X'4670' GENERATED CONSTANT	
0002E0	4670			885+	DC X'4670' GENERATED CONSTANT	
0002E2	4670			886+	DC X'4670' GENERATED CONSTANT	
0002E4	4670			887+	DC X'4670' GENERATED CONSTANT	
0002E6	4670			888+	DC X'4670' GENERATED CONSTANT	
0002E8	4670			889+	DC X'4670' GENERATED CONSTANT	
0002EA	4670			890+	DC X'4670' GENERATED CONSTANT	
0002EC	4670			891+	DC X'4670' GENERATED CONSTANT	
0002EE	4670			892+	DC X'4670' GENERATED CONSTANT	
0002F0	4670			893+	DC X'4670' GENERATED CONSTANT	
0002F2	4670			894+	DC X'4670' GENERATED CONSTANT	
0002F4	4670			895+	DC X'4670' GENERATED CONSTANT	
				897	FDCTRL IF=(IND(4)),	ON IF 'R' KEYED X
					GOTO=RREXP	
-----						
		1005 DELIVER	PDFIELD 61,80,		SOURCE='DELIVER',	SHIP-VIA POSITION X
					SINK=PRT,	GET LITERAL STRING X
					JUSTIFY=C	PUT TO SELECTRIC II X
						CENTERED X
						Branch to here from I X
-----						
				1290	*,IJL143 SELECTRIC II PRINT REGION BEGINS AT	
				1291	*, COLUMN 68, ENDS AT COLUMN 72	
				1292	*,IJL126 JUSTIFY OPTION FOR SINK 1 IS CENTER	
00045A				1293+	ORG IJL1000+1114 ORG TO DESIRED LOCATION TO ASH NEXT BYTE	
00045A	4070			1294+	DC H'16496' GENERATED ARITHMETIC	
				1295	*	
				1296	*,IJL128 SOURCE IS OMITTED	
				1297	*,IJL145 SOURCE CHARACTER COUNT IS 5	
				1298	*,IJL124 KIND IS UNRESTRICTED	
				1300 BODY	FDLINE 28,CYCLE=(28,AMOUNT,GROSS) PROVIDES FOR LINES 28-55	
				1301**FDLINE	-360N-CQ-490 - CHANGE LEVEL 3-A	
00045C				1302+BODY	EQU * *** TEST FOR DUPLICATE NAMES ***	
				1303	*	
				1304	*,IJL110 END OF SEGMENT 12	
				1305	*,IJL109 STARTING SEGMENT 13	
				1306	*	
				1307	*	
				1308	*,IJL139 LINE NUMBER IS 28	
				1309	*,IJL122 WIDTH IS 85	
				1310	*,IJL123 LEFT MARGIN IS COLUMN 12	
				1311	*,IJL123 RIGHT MARGIN IS COLUMN 80	
00042E				1312+	ORG IJL1000+1070 ORG TO DESIRED LOCATION TO ASH NEXT BYTE	
00042E	4770			1313+	DC X'4770' GENERATED CONSTANT	
000430	4475			1314+	DC H'17525' GENERATED ARITHMETIC	
0003E8				1315+	ORG IJL1000+1000 ORG TO DESIRED LOCATION TO ASH NEXT BYTE	
0003E8	4770			1316+	DC X'4770' GENERATED CONSTANT	
0003EA	4475			1317+	DC H'17525' GENERATED ARITHMETIC	
0003A0				1318+	ORG IJL1000+928 ORG TO DESIRED LOCATION TO ASH NEXT BYTE	
0003A0	4770			1319+	DC X'4770' GENERATED CONSTANT	
0003A2	4475			1320+	DC H'17525' GENERATED ARITHMETIC	
00036A				1321+	ORG IJL1000+874 ORG TO DESIRED LOCATION TO ASH NEXT BYTE	
00036A	4770			1322+	DC X'4770' GENERATED CONSTANT	
00036C	4475			1323+	DC H'17525' GENERATED ARITHMETIC	
00045E				1324+	ORG IJL1000+1118 ORG TO ASH NEXT BYTE AFTER HIGHEST	
00045E	4475			1325+	DC H'17525' GENERATED ARITHMETIC	
				1326	*	
				1327	*,IJL118 THIS SEGMENT ENTERED FROM SEGMENT 12	
00036E				1328+	ORG IJL1000+878 ORG TO DESIRED LOCATION TO ASH NEXT BYTE	
00036E	4671			1329+	DC X'4671' GENERATED CONSTANT	

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
000370	4070			1330+	DC H'16496' GENERATED ARITHMETIC
000372	467D			1331+	DC H'18045' GENERATED ARITHMETIC
000374	4670			1332+	DC X'4670' GENERATED CONSTANT
000376	4670			1333+	DC X'4670' GENERATED CONSTANT
000378	4670			1334+	DC X'4670' GENERATED CONSTANT
00037A	4670			1335+	DC X'4670' GENERATED CONSTANT
00037C	4670			1336+	DC X'4670' GENERATED CONSTANT
00037E	4670			1337+	DC X'4670' GENERATED CONSTANT
000380	4670			1338+	DC X'4670' GENERATED CONSTANT
000382	4670			1339+	DC X'4670' GENERATED CONSTANT
				1340	*
				1341	*,IJLF118 THIS SEGMENT ENTERED FROM SEGMENT 8
0003A4				1342+	ORG IJLF1000+932 ORG TO DESIRED LOCATION TO ASM NEXT BYTE
0003A4	4671			1343+	DC X'4671' GENERATED CONSTANT
0003A6	4070			1344+	DC H'16496' GENERATED ARITHMETIC
0003A8	4572			1345+	DC H'17778' GENERATED ARITHMETIC
0003AA	4670			1346+	DC X'4670' GENERATED CONSTANT
0003AC	4670			1347+	DC X'4670' GENERATED CONSTANT
0003AE	4670			1348+	DC X'4670' GENERATED CONSTANT
0003B0	4670			1349+	DC X'4670' GENERATED CONSTANT
0003B2	4670			1350+	DC X'4670' GENERATED CONSTANT
0003B4	4670			1351+	DC X'4670' GENERATED CONSTANT
0003B6	4670			1352+	DC X'4670' GENERATED CONSTANT
0003B8	4670			1353+	DC X'4670' GENERATED CONSTANT
				1354	*
				1355	*,IJLF118 THIS SEGMENT ENTERED FROM SEGMENT 9
0003EC				1356+	ORG IJLF1000+1004 ORG TO DESIRED LOCATION TO ASM NEXT BYTE
0003EC	4671			1357+	DC X'4671' GENERATED CONSTANT
0003EE	4070			1358+	DC H'16496' GENERATED ARITHMETIC
0003F0	4377			1359+	DC H'17271' GENERATED ARITHMETIC
0003F2	4670			1360+	DC X'4670' GENERATED CONSTANT
0003F4	4670			1361+	DC X'4670' GENERATED CONSTANT
0003F6	4670			1362+	DC X'4670' GENERATED CONSTANT
0003F8	4670			1363+	DC X'4670' GENERATED CONSTANT
0003FA	4670			1364+	DC X'4670' GENERATED CONSTANT
0003FC	4670			1365+	DC X'4670' GENERATED CONSTANT
0003FE	4670			1366+	DC X'4670' GENERATED CONSTANT
000400	4670			1367+	DC X'4670' GENERATED CONSTANT
				1368	*
				1369	*,IJLF118 THIS SEGMENT ENTERED FROM SEGMENT 10
000432				1370+	ORG IJLF1000+1074 ORG TO DESIRED LOCATION TO ASM NEXT BYTE
000432	4671			1371+	DC X'4671' GENERATED CONSTANT
000434	4070			1372+	DC H'16496' GENERATED ARITHMETIC
000436	4174			1373+	DC H'16756' GENERATED ARITHMETIC
000438	4670			1374+	DC X'4670' GENERATED CONSTANT
00043A	4670			1375+	DC X'4670' GENERATED CONSTANT
00043C	4670			1376+	DC X'4670' GENERATED CONSTANT
00043E	4670			1377+	DC X'4670' GENERATED CONSTANT
000440	4670			1378+	DC X'4670' GENERATED CONSTANT
000442	4670			1379+	DC X'4670' GENERATED CONSTANT
000444	4670			1380+	DC X'4670' GENERATED CONSTANT
000446	4670			1381+	DC X'4670' GENERATED CONSTANT
				1382	*
				1383	*,IJLF118 THIS SEGMENT ENTERED FROM SEGMENT 11
000460				1384+	ORG IJLF1000+1120 ORG TO ASM NEXT BYTE AFTER HIGHEST
000460	4673			1385+	DC X'4673' GENERATED CONSTANT
000462	4070			1386+	DC H'16496' GENERATED ARITHMETIC
000464	417C			1387+	DC H'16764' GENERATED ARITHMETIC
000466	4070			1388+	DC H'16496' GENERATED ARITHMETIC
000468	417C			1389+	DC H'16764' GENERATED ARITHMETIC
00046A	4670			1390+	DC X'4670' GENERATED CONSTANT
00046C	4670			1391+	DC X'4670' GENERATED CONSTANT

J

J  
Cycle

000540				1662+	ORG IJLF1000+1344 ORG TO ASM NEXT BYTE AFTER HIGHEST
000540	4471			1663+	DC H'17521' GENERATED ARITHMETIC
000542	4674			1664+	DC X'4674' GENERATED CONSTANT
00046A				1665+	ORG IJLF1000+1130 ORG TO DESIRED LOCATION TO ASM NEXT BYTE
00046A	4070			1666+	DC H'16496' GENERATED ARITHMETIC
00046C	467B			1667+	DC H'18043' GENERATED ARITHMETIC

K

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
				1669	GROSS FDLINE 58 FOOTING LINE
				1670	*FDLINE -360N-CQ-490 - CHANGE LEVEL 3-A
00046E				1671	+GROSS EQU * *** TEST FOR DUPLICATE NAMES ***
000544				1672	ORG IJLF1000+1348 ORG TO ASM NEXT BYTE AFTER HIGHEST
000544	4675			1673	DC X'4675' GENERATED CONSTANT
				1674	*, IJLF132 AT END OF CYCLE PRINT ELEMENT WAS
				1675	*, POSITIONED ON LINE 56 OF FORM
				1676	*,
				1677	*, IJLF110 END OF SEGMENT 13
				1678	*, IJLF109 STARTING SEGMENT 14
				1679	*,
				1680	*,
				1681	*, IJLF139 LINE NUMBER IS 58
				1682	*, IJLF122 WIDTH IS 85
				1683	*, IJLF123 LEFT MARGIN IS COLUMN 12
				1684	*, IJLF123 RIGHT MARGIN IS COLUMN 80
				2146	*,
				2147	*, IJLF116 BUFFERS USED IN PATH 1
				2148	*,
				2149	*, IJLF130 STG
				2150	0, IJLF421 STG BUFFER MAY NOT HAVE BEEN USED AS
				2151	0, OUTPUT SINCE PRIOR INPUT
				2152	*, IJLF130 RDR/PCH
000686				2153	ORG IJLF1001+206 ORG TO ASM NEXT BYTE AFTER HIGHEST
000686	4070			2154	DC X'4070' GENERATED CONSTANT
				2155	*,
				2156	*, IJLF118 THIS SEGMENT ENTERED FROM SEGMENT 14
000000				2157	+IJLF1000 CSECT
000008				2158	ORG IJLF1000+8 ORG TO DESIRED LOCATION TO ASM NEXT BYTE
000008	0000			2159	DC X'0000' GENERATED CONSTANT
000014				2160	ORG IJLF1000+20 ORG TO DESIRED LOCATION TO ASM NEXT BYTE
000014	4070			2161	DC H'16496' GENERATED ARITHMETIC
000016	4070			2162	DC H'16496' GENERATED ARITHMETIC
000018	4070			2163	DC H'16496' GENERATED ARITHMETIC
00001A	4472			2164	DC H'17522' GENERATED ARITHMETIC
				2165	*,
				2166	*, IJLF146 FORM DESCRIPTION PROGRAM SPECIFIED
				2167	*, SELECTRIC II FORM HAVING 66 LINES
000588				2168	+IJLF1001 CSECT
000688				2169	ORG IJLF1001+208 ORG TO ASM NEXT BYTE AFTER HIGHEST
000688	4679			2170	DC X'4679' GENERATED CONSTANT
00068A	401E			2171	DC X'401E' GENERATED CONSTANT
00068C	4070407040704070			2172	DC (134)X'4070'
000798	FFFFFFFFFFFF			2173	DC 3H'-1' LAST SECTOR FLAG
00079E	FFFFFFFFFFFF			2174	DC (2+1)H'-1' LAST CSECT FLAG
00079E				2175	ORG *-4-2*1
				2176	*,
				2177	*, IJLF147 SUMMARY OF FDP-GENERATED DATA
				2178	*,
				2179	*, UNPACKED FDP OUTPUT = 4 BLOCKS
				2180	*, 3735 DISK STORAGE = 3.4 SECTORS
				2181	*,
				2182	*, IJLF133 NO TERMINATING ERRORS FOUND IN THIS FDP
				2183	END

Branch for J

M

GR 0-7	00006EF0	00006ED0	00006D0E	00006692	0000667E	00006D78	9000612C	00005F98
GR 8-F	00007B0A	0A0407F1	8000617A	40006092	00006FD8	66648	64A4	0000
FP REG	00000000	00000000	00000000	00000000	00A80600	<b>A</b> 0500	<b>B</b> 0100	0000 <b>C</b>
006880	<b>C</b>	0000C6C4	C6D6D9D4	40400002	40704370	43724376	44704070	40704070
0068A0	4472	4575	45734475	42704476	447F4572	447D4270	43774378	43764375
0068C0	4270	42704476	44744570	42704370	43714371	4270447D	4575	<b>D</b> 45744270
0068E0	<b>E</b> 4475	42704570	44754572	44754572	4572447D	44754474	4270	44754476
006900	4572	44754270	45744478	44754478	42704476	44744572	27F477F	4175417F
006920	4070	4677	072477F	467A4070	477	41704672	407041	<b>G</b> 0704071
006940	407440	<b>G</b> 1744471	45764070	41784074	41744471	457640	27C4074	41744271
006960	477040	5764070	44704174	40754070	40734071	427A4270	42704071	447B4270
006980	4071447B	40704070	46724070	40704070	407D4770	44724671	40704371	46704670
0069A0	46704670	46704670	46704670	47704472	45764070	40744074	41744471	45764070
0069C0	41784074	41744471	45764070	427C4074	41744074	47704071	45764070	44704074
0069E0	4075	47704475	46714070	417A4670	4670	<b>H</b> 46704670	46704670	46704076
006A00	4174	<b>H</b> 44714076	41744370	44714076	4174	42714770	40714176	40754078
006A20	4475	40704074	41714076	41704070	407A4076	45794379	427F4379	4379427F
006A40	43794379	477F4071	45744070	407A4171	40774570	42704070	4070407B	40774379
006A60	43794270	43790000	00000000	03107C00				

GR 0-7	00006EF0	00006ED0	00006DCE	00006692	0000667E	00006D78	9000612C	00005F98
GR 8-F	00007B0A	0A0407F1	8000617A	40006092	00006FD8	60006648	800064A4	00000000
FP REG	00000000	00000000	00000000	00000000	00A80600	00A80500	00A90100	00000004
006880		0000C6C4	C6D6D9D4	40400002	40704379	43794379	4379477F	40704572
0068A0	41714071	43714470	42704070	40704573	40704171	407A4170	40714070	407A4077
0068C0	40714576	40704475	417C4075	40714576	4070447A	41744073	40714576	4070447D
0068E0	4274407A	40754576	40704577	41714074	41704070	40704074	43794379	427F4379
006900	4379477F	40704073	4071417A	42704271	40714471	42	<b>I</b> 71	44744271
006920	42714071	40714071	40714574	42744071	44744170	42	<b>I</b> 71	45704270
006940	45724370	40714071	45744470	40704672	40704170	4070407D	47704271	47704073
006960	46714070	457F4670	46704670	46704670	46704672	40704270	4070407D	47704271
006980	47704073	46714070	46784670	46704670	46704670	46704672	40704370	4070407D
0069A0	47704271	47704171	46714070	47714670	46704670	46704670	46704672	40704470
0069C0	4070407D	47704271	47704074	46714071	40724670	46704670	46704670	46704770
0069E0	42714770	40714572	4071407B	44714479	45724270	44764572	44754479	44774478
006A00	45744170	407C4672	40704770	44754671	4070467D	46704670	46704670	46704670
006A20	46704670	45724071	40774474	4475447C	44794576	44754572	41704070	46724070
006A40	47704475	46714070	45724670	46704670	46704670	46704670	46704572	40714077
006A60	45704479	44730000	00000000	03107000				

GR 0-7	00006EF0	00006ED0	00006D0E	00006692	0000667E	C0006D78	9000612C	00005F98
GR 8-F	00007B0A	0A0407F1	8000617A	40006092	00006FD8	60006648	800064A4	00000000
FP REG	00000000	00000000	00000000	00000000	00A80600	00A80500	00A90100	00000004
006880		0000C6C4	C6D6D9D4	40400002	4070447B	42704575	45704170	40704672
0068A0	40704770	44754671	40704377	46704670	46704670	46704670	46704670	4071
0068C0	407F457	714479	447C4577	44714579	42704475	45784570	45724475	J 4573
0068E0	4170407	J 724070	47704475	46714070	41744670	46704670	46704670	4670
006900	4670457	714075	45744572	45754473	447B4170	40704672	44754673	4070417C
006920	4070417C	4070467B	417B4073	43704570	46714071	46704072	40704272	4071417B
006940	40734270	457C4170	40724070	42724071	45734072	4179407A	41704070	407A4073
006960	457A457A	457A477F	40714077	4072417A	40704271	40724475	44714271	40724474
006980	457A4271	40724477	45724270	4072447C	44724071	417E4075	41794270	40714077
0069A0	417E4374	40724271	417B4075	43744570	40724078	40714070	407A4075	K L
0069C0	427A427A	4576427E	43794379	477F4071	45734071	4179407A	45704270	46744675
0069E0	407A4076	4274427A	427C427A	427A427A	4576427E	43794379	477F4471	40704571
006A00	47704472	45704179	4C734170	4070407A	40734379	43794379	427F477F	40704571
006A20	41794073	41704070	407A4073	43794379	4379477F	42714573	40734171	407A4170
006A40	4070407B	40784274	427A427A	427A427C	427A427A	427A4576	427E4379	4379477F
006A60	40714576	4C7C0000	0C000000	03107000				

GR 0-7	00006EF0	00006ED0	00006D0E	00006692	0000667E	00006D78	9000612C	00005F98
GR 8-F	00007B0A	0A0407F1	8000617A	40006092	00006FD8	60006648	800064A4	00000000
FP REG	00000000	00000000	00000000	00000000	00A80600	00A80500	00A90100	00000004
006880		0000C6C4	C6D6D9D4	40400002	4070457B	41794073	41704070	407A4073
0068A0	4379427E	43794379	477F4070	45734073	4171407A	44704078	40744070	45724171
0068C0	40734371	43704370	44704072	40744070	47704071	45734074	4179407A	45704270
0068E0	40734070	407A4076	4274427A	427C427A	427A427A	4576427E	43794379	477F4071
006900	417B4076	42704570	42704073	4070407A	40764274	427A M	427A427A	427A4576
006920	427E4379	4379477F	40714573	40734179	407A4170	4070	40784274	427A427A
006940	427A427C	427A427A	427A4576	427E4379	4379477F	40704679	401E4070	40704070
0C6960	40704070	--SAME--						
006A60	40704070	4070FFFF	FFFFFFF	03107000				



- A** 303236 → FID=026
- B** 40 → PACKING=DELIMIT
- C** 0042 → Form Length is 66 Lines
- D** 7F → Begin Tab Definition
- E** 00 → Begin The FCD

**F** 560018041441 → FDFIELD (ADDRESS 1) Specification

<u>56</u> Source = 5496 Buffer	<u>0018</u> Buffer Start Position Relative to 4, Zero Origin	<u>04</u> Data Type Byte for PRT	<u>14</u> Char Count = 20	<u>41</u> End Control Indicator = New Line 1
---	--	---	------------------------------------	--

**G** 03012A2020014B20014B0000 → FDFIELD Control Character Input Specification

<u>03</u> Data Type Byte for Validity & Function Bytes to Follow	<u>01</u> Char Count = 1	<u>2A</u> Validity Byte, Value Check Follows*	<u>20</u> Function Byte for COMPARE	<u>20</u> Value Byte for EQUAL	<u>01</u> Char Count = 1
--	-----------------------------------	--	--	--	-----------------------------------

<u>4B</u> Char Count = K	<u>20</u> Compare Byte = EQ	<u>01</u> Char Count = 1	<u>4B</u> Char Count = K	<u>00</u> IND=1	<u>00</u> End Control Indicator = Space Zero
-----------------------------------	--------------------------------------	-----------------------------------	-----------------------------------	--------------------	---

\* Specifies enter mode required, fixed length records.

**H**

110610000A0659392F39392F39397F



FDFIELD (DATEFLD) Specification

<u>11</u>	<u>06</u>	<u>10</u>	<u>00</u>	<u>0A</u>
Data Type Byte for Numeric Function Byte to Follow	Char Count = 6	Function Byte for Data Sink Group Bytes to Follow	Data Sink Byte for PRT Selectric	Selectric Sink Byte for EDIT and PRT

<u>06</u>	<u>59392F39392F3939</u>	<u>7F</u>
PICTURE Digit Count = 6	PICTURE = Y9/99/99	PICTURE Delimiter

**I**

620010000D7021700361005F606060606062...



FDCTRL IF = (IND (2)), GOTO=DELIVER Specification

<u>62</u>	<u>00</u>	<u>10</u>	<u>000D</u>
GOTO On Condition	Condition Byte = NOT	IND = 01	Branch (IND = OFF) If IND = OFF, Displacement = +13 Branch to the Next FDCTRL Macro

<u>7021700361005F606060606060</u>	<u>62...</u>
If IND = ON, Branch to the FDFIELD Macro Named DELIVER	Next FDCTRL Macro

**J**

63001C001C006B



CYCLE Beginning at FDLINE (BODY) Specification

<u>63</u>	<u>001C</u>	<u>001C</u>	<u>006B</u>
Begin Cycle Byte	Number of Lines in Cycle = 28	Number of Repetitions in Cycle = 28	Displacement to FDLINE Macro Named GROSS

**K**

64



Repeat Cycle Byte

**L**

65



End Cycle Byte (in FDLINE Named GROSS)

**M**

69



End Form Byte (in CSECT Named IJLF1001)

## Glossary

The following terms are defined as they are used in this manual. If you do not find the term you are looking for, refer to the index or to the *IBM Data Processing Glossary*, GC20-1699.

IBM is grateful to the American National Standard Institute (ANSI) for permission to reprint its definitions from the American National Standard Vocabulary for Information Processing (copyright © 1970 by American National Standards Institute, Incorporated), which was prepared by Subcommittee X3.5 on Terminology and Glossary of American National Standards Committee X3.

**Access lines.** The communication lines that join the central computer and the remote terminal to common-carrier exchange equipment.

**Access method.** A combination of an access technique (either queued or basic) and a given data set organization (for instance, sequential, partitioned, indexed sequential, or direct) that allows the transfer of data between main storage and I/O devices.

**Array.** An arrangement of elements in one or more dimensions.\* In this publication, an array is an arrangement of all the bits of a particular global variable. For example, the &PIB array contains 48 separate bits: &PIB(1), &PIB(2) and so forth.

**ASCII.** American Standard Code for Information Interchange. The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems, communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.\*

**Assemble.** To prepare a machine language program from a symbolic language program by substituting absolute operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.\*

**Assembler.** A computer program that assembles.\*

**Authority.** The extent of a macro statement through an FD program until the same type of macro statement (or one of higher authority) appears later in the FD program.

**Binary synchronous communications (BSC).** Data transmission in which character synchronization is controlled by timing signals generated by the device that originates a message (and the device that obtains the message recognizes the *sync pattern* at the beginning of the transmission - the devices are locked in step with one another).

**BSC.** See binary synchronous communications.

**Common carrier.** A company that furnishes communication services to the general public and is regulated by appropriate local, state, or federal agencies.

**Communication line.** The medium over which data signals are transmitted.

**Control character.** A character whose occurrence in a particular context initiates, modifies, or stops a control operation - for example, a character to control carriage return.\*

**Control section.** The smallest separately relocatable unit of a program. That portion of text specified by the programmer to be an entity, all elements of which are to be loaded into contiguous main storage locations.

**Copy book.** Source language coding that is copied, via a COPY instruction, from a library and is included in the program currently being assembled. A member of a partitioned data set can be copied from either the system macro library or a user library concatenated to it.

**Delimiting macro.** The FDEND Form Description macro statement that closes the description of each form and prepares for a form description that may follow.

**Diagnostic macros.** The optional FDTRACE and FDDSPPLY Form Description macro statements that provide a trace facility and a display facility for aid in diagnosing error conditions.

**Encoder.** Person who designs and defines forms by coding Form Description macro statements.

**FCD.** See Field control descriptor.

**FD macro.** See Form Description macro.

**FD program.** See Form Description program.

**FD utility.** See Form Description utility.

**Field control descriptor.** In the 3735 terminal, the part of a Form Description program that the 3735 interprets as the directions for the processing of one data field or the performance of one logical or device-control operation.

**Form Description macro.** One of a set of specialized macro instructions with which a forms encoder can describe symbolically the structure of a data processing form, the characteristics of each field on the form, and the processing to be done on each field by the 3735 terminal and its operator.

**Form Description program.** A set of control information interpreted or executed by the 3735 terminal or other processor as the complete set of instructions for the processing of one type of form.

**Form Description utility.** A computer program that restructures one or more object modules obtained from the assembly of FD macro statements into unpacked program blocks (UPBs) and writes the blocks into a user's data set.

**Forms encoder.** See encoder.

**Global variable.** SET symbols defined in one macro definition used to vary the statements that appear in other macro definitions.

**Identification (ID) characters.** Characters sent by a BSC terminal on a switched line to identify the terminal.

**Inner macro.** Macro statement called by an outer (external) macro or by another inner (internal) macro.

**Keyed unpacked program block (KUPB).** A physical 486-byte block of auxiliary storage containing a subset of an FD program that results from the assembly of a set of FD macro statements describing one type of form to be processed.

**Macro definition.** A set of statements that provides the assembler with: (1) the mnemonic operation code and the format of the macro instruction, and (2) the sequence of statements the assembler generates when the macro instruction appears in the source program.

**MNOTE message.** A message appearing on the diagnostic listings that result from the assembly of macro statements. The message provides diagnostic information regarding coding errors in the macro statements and provides descriptive information for verifying the correctness of each macro specification.

---

\* American National Standard Definition

*Modulo.* The remainder after any division has been performed.

*Multipoint line.* A communication line that connects more than one terminal; also known as multidrop line.

*Nonswitched line.* A communication line that connects a terminal and the computer for a continuous period or for regularly recurring periods of time at stated hours for the exclusive use of one installation; also known as a private, leased, or dedicated line.

*Object module.* A module that is the output of an assembler or compiler and is input to a linkage editor.\*

*Outer macro.* Macro statement specified externally by the 3735 forms encoder.

*Path.* One portion of an FD program created by one of the following actions: (1) the start of the FD program; (2) the encoding of the SAVELOC operand outside of a cycle; (3) the joining of two paths.

*Point-to-point line.* A communication line that connects a single remote terminal to the computer. It may be either switched or non-switched.

*Procedural macro.* The FDCTRL Form Description macro that enables the checking of terminal control program status.

*Promotability.* The ability of a keyword operand to be coded in some particular macro instruction and also to be coded in one or more macros of higher authority.

*Resource.* Any system facility that is required by a job or task; for example, main storage, I/O devices, data sets.

*Scope.* The extent of a structural FD macro instruction through the form description until the same type of macro statement (or a macro statement of a higher level) appears later in the description. Also called *authority*.

*Segment.* A part of a computer program which has structure such that the program can be executed without the entire program being in internal storage at one time. Several conditions create the segments comprising a path of an FD program: (1) the start of a path; (2) the issuance of a branch; (3) the creation of a cycle; (4) the encoding of a SAVELOC operand within a cycle; (5) the appearance of the post-limit macro; (6) the joining of several segments in the path.

*Structural macro.* One of four Form Description macro statements (FDFORM, FDPAGE, FDLINE, or FDFIELD) that define the structural organization of a form and the processing required by each field in the form.

*Switched line.* A communication line on which the connection between the computer and a remote terminal is established by dialing; also known as a dial or dial-up line.

*Target.* The name specified on the subsequent outer FD macro that is to be processed when cyclic or GOTO processing is either complete or stopped by the 3735 operator.

*Telecommunications.* Any transmission or reception of signals, writing, sounds, or intelligence of any nature, by wire, radio, or other electromagnetic media.

*Teleprocessing.* The processing by a computer of data entered at a remote terminal.

*Terminal.* A point in a system at which data can enter, leave, or enter and leave;\* also a control unit to which one or more I/O devices can be attached.

*Terminal control program.* The microcoded control program recorded in the terminal control unit during manufacture of the 3735 that interprets FD programs as a set of directions for processing one type of form and provides detailed terminal control.

*Transmission.* The transfer of coded data by an electromagnetic medium between two points in a telecommunications network.

*Unpacked program block (UPB).* A physical 476-byte block of secondary storage containing a subset of an FD program suitable for transmission to the 3735 terminal.

*Variable.* See global variable.

---

\* American National Standard Definition

- A
  - access lines 4-27
  - access method 4-27
  - access methods, 3735 supported 1-3
  - actions, error, in FD utility 3-4
  - address resolution and motion control assembly 2-26
  - arithmetic bytes
    - function byte 2-44
    - function group 2-46
  - array, definition 4-27
  - ASCII 4-27
  - assemble, definition 4-27
  - assembler, definition 4-27
  - assembly
    - address resolution 2-26
    - arithmetic group 2-29
    - BATCH operand 2-27
    - cancel form byte 2-27
    - character-count byte 2-28
    - clear counter bytes 2-27
    - clear STG bytes 2-27
    - compare group 2-29
    - conditional GOTO bytes 2-27
    - cycle bytes 2-27
    - data source group 2-27
    - data-sink group 2-29
    - data-type byte 2-28
    - end form byte 2-27
    - FD program 2-24
    - FD program header 2-24
    - field control descriptor 2-27
    - function byte 2-28
    - GOTO bytes 2-27
    - hexadecimal dump 4-15
    - IDR/CCR command bytes 2-27
    - immediate bytes 2-27
    - index space bytes 2-27
    - INQ command bytes 2-27
    - internal 3735 code 4-15
    - minimum character count 2-29
    - motion control 2-26
    - NOP byte 2-27
    - PRINT GEN listing 4-15
    - Selectric command bytes 2-27
    - self-check byte 2-29
    - set indicator bytes 2-27
    - total batch bytes 2-27
    - validity byte 2-28
    - value-check group 2-29
    - 3286-3 command bytes 2-27
    - 5496 command bytes 2-27
  - assembly input 2-7
  - assembly of FD programs, general 2-24
  - assembly of macro instructions, introduction 1-5
  - assembly output 2-7
  - authority
    - definition 4-27
    - introduction 2-4
  - auxiliary storage requirements, FD utility 3-3
  
- B
  - basic configuration 1-3
  - basic storage capacity 1-3
  - begin byte 2-32
  - Binary Synchronous Communication, definition 4-27
  - Binary Synchronous Communications (BSC), general 1-3
  - book use iv
  - BSC (*see* Binary Synchronous Communications)
  - BSC line adapter 1-3
  
- C
  - CALLOAD routine
    - in IJLFLOAD module 3-49
    - in IJLFUPDT module 3-51
  - CNTLEND routine
    - in IDFCT module 3-39
    - in IJLFCT module 3-43
  - CNTLINIT routine
    - in IDFCT module 3-35
    - in IJLFCT module 3-40
  - CNTLSTMT routine
    - in IDFCT module 3-36
    - in IJLFCT module 3-40
  - codes, 3735 transmission 2-19
  - common carrier 4-27
  - communication line 4-27
  - conditional GOTO bytes
    - feature indicators 2-34
    - general purpose indicators 2-34
    - special indicators 2-34
  - configuration
    - basic 3735 1-3
    - System/360 1-3
    - System/370 1-3
  - considerations
    - FD macro operation 2-7
    - operational, FD utility 3-4
    - programming 1-5
    - storage 1-3
    - system 1-4
  - control character 4-27
  - control information
    - DOS 3-5
    - OS 3-4
  - control section (*see* CSECT)
    - DOS 2-7
    - OS 2-7
  - control statements
    - for DOS Linkage Editor step 3-17
    - for DOS Storage step 3-21
      - DEVICE 3-28
      - OPTION 3-28
      - RPLACE 3-28
    - for OS Linkage Editor step 3-14
  - Control step
    - DOS method of operation 3-17
    - functions 3-14
    - OS method of operation 3-14
    - program organization
      - IDFCT 3-35
      - IJLFCT 3-40
  - control storage 1-3
  - copy book
    - definition 4-27
    - IDFGBL 2-12
    - IJLFGBL 2-12
  - cross references, FD utility modules 3-76
  - CSECT (control section)
    - description of 3-7
    - names 3-7
  - CTLSYNER routine 3-39
  
- D
  - data descriptions, FD utility 3-6
  - data flow
    - among FD utility modules 3-10
    - in DOS FD utility 3-5
    - in OS FD utility 3-5
  - data format byte 2-31
  - data sets
    - for FD utility execution 3-3
    - intermediate, for DOS FD utility 3-5
    - output from FD utility 3-10
    - used by FD utility
      - DOS 3-4
      - OS 3-4
  - data source error message 4-14
  - data source group
    - counters 2-42
    - emitted data 2-40
    - FD program ID 2-40
    - IDR/CCR buffer 2-42
    - inquiry (INQ) buffer 2-42

- record number 2-40
- storage (STG) buffer 2-42
- 3286-3 buffer 2-42
- 5496 buffer 2-42
- data transmission 1-6
- default source 2-33
- delimiting macro (*see* FDEND)
  - definition 4-27
  - introduction 2-5
- descriptions of data, FD utility 3-6
- desk-side control unit 1-3
- DEVICE control statement, DOS FD utility
  - Storage step 3-28
- diagnostic macros (*see* FDTRACE or FDDSPY)
  - definition 4-27
  - display 4-5
  - introduction 2-5
  - trace 4-5
- diagnostic messages 4-6
- DIAGWTR routine
  - in IDFCT module 3-39
  - in IDFST module 3-46
  - in IJLFCT module 3-43
  - in IJLFLOAD module 3-51
  - in IJLFUPDT module 3-54
- DIMOD routine
  - in IJLFCT module 3-43
  - in IJLFLOAD module 3-51
  - in IJLFST module 3-49
  - in IJLFUPDT module 3-54
- directory references, FD utility 3-76
- display macro (FDDSPY) 4-5
- DOS
  - control section 2-7
  - duplicate form handling 1-6
  - minimum storage 1-4
  - program support 1-3
- DOS FD utility
  - control information 3-5
  - method of operation
    - Control step 3-17
    - Linkage Editor step 3-21
    - Storage step 3-21
- DOS teleprocessing restriction 2-6
- duplicate form handling
  - DOS 1-6
  - OS 1-6

E

- encoder 4-27
- end control byte 2-50
- END record, FD utility 3-14
- ENTAB (Linkage Editor entry table) 3-17
- entry table (*see* ENTAB)
- environmental errors 1-6
- EOFRTN routine 3-47
- EOJRTN routine
  - in IJLFLOAD module 3-51
  - in IJLFUPDT module 3-54
- ERREXT routine 3-47
- ERREXTI routine 3-47
- error actions
  - DOS FD utility
    - Control step 3-17
    - Storage step 3-29
  - FD utility 1-6
  - OS FD utility
    - Control step 3-17
    - Storage step 3-21
- error checking 1-3
- error messages
  - data source 4-14
  - FD program 4-14
  - MNOTE 4-6
- errors
  - environmental 1-6
  - FD utility handling 1-6
  - implementation 1-6
  - severity codes 2-7
  - space allocation 1-6
  - system control block 1-6
- ESD records
  - FD utility 3-14

- validation of 3-14
- ESDPROC routine
  - in IDFCT module 3-35
  - in IJLFCT module 3-40
- examination, of KUPBs 3-13
- execution
  - DOS FD utility 3-5
  - OS FD utility 3-4

F

FCD

- assembly 2-27
- batch group 2-37
- character-count byte 2-43
- data source group 2-37
- data-type byte 2-42
- emitted data group 2-43
- end control byte 2-50
- function byte 2-44
- function group 2-46
- validity byte 2-44
- validity group 2-44

FCD bytes, structure 2-37

FD macro, definition 4-27

FD macros
 

- assembly input 2-7
- assembly output 2-7
- format 4-3
- functions
  - introduction 1-5
  - summary 2-19
- introduction 2-4
- method of operation 2-12
- nesting concept 2-12
- operational considerations 2-7
- organization 2-6
- structure 2-12
- system requirements 2-6

FD program
 

- appendix A 4-15
- assembly 2-24
- definition 4-27
- introduction 1-3
- maintenance 2-50
- organization 2-31
- restriction 1-3
- sample 4-15
- storage 1-3
- transmission 1-6
- trouble shooting 2-50

FD program header
 

- assembly 2-24
- begin byte 2-32
- data format byte 2-31
- form ID 2-31
- lines form-printer 2-32
- lines form-Selectric 2-32
- operator message 2-32
- structure 2-31
- tab intervals 2-32

FD program message 4-14

FD program trailer 2-50

FD utility
 

- Control step 3-3
- definition 4-27
- DOS modules 3-3
- error actions 3-4
  - DOS Control step 3-17
  - DOS Storage step 3-29
  - OS Control step 3-17
  - OS Storage step 3-21
- execution 3-3
  - DOS 3-5
  - OS 3-4
- function 3-3
- functions, introduction 1-6
- general operation 3-6
- input 3-4
- introduction 1-6
- Linkage Editor step 3-3
- operation 3-3
- OS modules 3-3
- output 3-4

- overlay segments of storage step 3-13
- physical characteristics 3-3
  - DOS 3-3
  - OS 3-3
- program organization 3-35
- purpose 3-3
- scheduling 3-4
- storage 1-6
- Storage step 3-3
- system requirements 3-3
- FDCTRL outer macro
  - function 2-22
  - general 2-4
  - method of operation 2-22
- FDDSPY macro
  - function 4-5
  - introduction 2-6
- FDEND outer macro
  - function 2-23
  - general 2-5
  - method of operation 2-23
- FDFIELD outer macro
  - function 2-19
  - general 2-4
  - method of operation 2-19
- FDFORM outer macro
  - function 2-19
  - general 2-4
  - method of operation 2-19
- FDLINE outer macro
  - function 2-19
  - general 2-4
  - method of operation 2-19
- FDPAGE outer macro
  - function 2-19
  - general 2-4
  - method of operation 2-19
- FDTRACE macro
  - function 4-5
  - introduction 2-5
- features
  - optional 1-3
  - required 1-6
- field control descriptor
  - assembly 2-27
  - definition 4-27
- flow
  - data
    - in DOS FD utility 3-5
    - in OS FD utility 3-5
    - logical, in FD utility 3-6
- FLSHCARD routine 3-47
- FLUSH routine 3-53
- Form Description (*see* FD)
- Form Description utility (*see* FD utility)
- form descriptions
  - control 1-5
  - field 1-5
  - form 1-4
  - introduction 1-4
  - line 1-5
  - page 1-4
- form ID 2-31
- format, FD macro 4-3
- forms design
  - introduction 1-4
  - restrictions 1-4
- forms encoder (*see* encoder)
- function byte
  - arithmetic 2-44
  - compare 2-44
  - data sink 2-44
- function group
  - arithmetic bytes 2-46
  - compare bytes 2-49
  - data sink bytes 2-49
- functions
  - Control step 3-14
  - FD utility, introduction 1-4
  - Linkage Editor step 3-17
  - program support 1-3
  - Storage step 3-21

## G

- GET routine 3-47
- GETCARD routine 3-47
- GETCHECK routine
  - in IDFCT module 3-37
  - in IJLFCT module 3-42
- global variable
  - definition 4-27
  - general 2-6
  - symbols 2-12
- glossary 4-27

## H

- header, FD program 2-31
- hexadecimal dump, assembly code 4-15
- horizontal tab intervals 2-32
- how to use this book iv

## I

- IBM Selectric® I/O II Keyboard Printer 1-3
- IBM System/360, configuration 1-3
- IBM System/370, configuration 1-3
- IBM 3735, configuration 1-3
- identification (ID) characters 4-27
- IDFASM inner macro 2-24
- IDFCT module organization 3-35
  - CNTLEND 3-39
  - CNTLINIT 3-35
  - CNTLSTMT 3-36
  - CTLSYNER 3-39
  - DIAGWTR 3-39
  - ESDPROC 3-35
  - GETCHECK 3-37
  - INPUT DCB 3-40
  - MSGFAN 3-39
  - OBJCARD 3-40
  - OUTPUT DCB 3-40
  - PRINT DCB 3-40
  - PUTCARD 3-39
  - TXTPROC 3-35
- IDFDSP inner macro 4-5
- IDFM01 (message module) organization 3-54
- IDFST module, processing 3-12
- IDFST module organization 3-44
  - DIAGWTR 3-46
  - MSGFAN 3-46
  - NEWMEM 3-44
  - OUTPUT DCB 3-46
  - PARMPROC 3-44
  - PRINT DCB 3-46
  - REPLTAB 3-47
  - SEGPROC 3-44
  - STGEND 3-46
  - STGINIT 3-44
  - SYNADI 3-45
  - SYNAD2 3-45
- IDFTMP temporary names, FD utility 3-10
- IJLFASM inner macro 2-24
- IJLFCT module organization 3-40
  - CNTLEND 3-43
  - CNTLINIT 3-40
  - CNTLSTMT 3-40
  - DIAGWTR 3-43
  - DIMOD 3-43
  - ESDPROC 3-40
  - GETCHECK 3-42
  - MSGFAN 3-43
  - PUTCARD 3-42
  - SYSIPT DTFDI macro 3-43
  - SYSLST DTFDI macro 3-43
  - SYSFCH DTFDI macro 3-43
  - TXTPROC 3-41
- IJFDLIB DTFIS macro
  - in IJLFLOAD module 3-51
  - in IJLFUPDT module 3-54
- IJLFLOAD module, processing 3-13
- IJLFLOAD module organization 3-49
  - CALLOAD 3-49
  - DIAGWTR 3-51
  - DIMOD 3-51

EOJRTN 3-51  
 IJFDLIB DTFIS MACRO 3-51  
 ISMOD 3-51  
 ISMPUT 3-49  
 LOADRTN 3-49  
 MSGFAN 3-49  
 RCDCHK 3-49  
 SYSLST DTFDI MACRO 3-51  
 IJLFM01 (message module) organization 3-54  
 IJLFST module, processing 3-13  
 IJLFST module organization 3-47  
 DIMOD 3-49  
 EOFRTN 3-47  
 ERREXT 3-47  
 ERREXT1 3-47  
 FLSHCARD 3-47  
 GET 3-47  
 GETCARD 3-47  
 OPEN 3-47  
 SYSIPT DTFDI MACRO 3-49  
 SYSLST DTFDI MACRO 3-49  
 IJLFM temporary names, FD utility 3-10  
 IJLFUPDT module, processing 3-13  
 IJLFUPDT module organization 3-51  
 CALLOAD 3-51  
 DIAGWTR 3-54  
 DIMOD 3-54  
 EOJRTN 3-54  
 FLUSH 3-53  
 IJFDLIB DTFIS MACRO 3-54  
 ISMOD 3-54  
 ISMPUT 3-52  
 LOADRTN 3-52  
 MSGFAN 3-53  
 RCDCHK 3-51  
 SYSLST DTFDI MACRO 3-54  
 UPDTNM 3-53  
 immediate bytes  
 assembly 2-27  
 begin cycle bytes 2-34  
 cancel form byte 2-35  
 clear counter bytes 2-35  
 clear storage (STG) bytes 2-35  
 conditional GOTO bytes  
 feature indicators 2-34  
 general 2-34  
 general purpose indicators 2-34  
 special indicators 2-34  
 end cycle byte 2-35  
 end form byte 2-35  
 GOTO bytes 2-34  
 IDR and CCR command bytes 2-37  
 index space bytes 2-35  
 inquiry (INQ) command bytes 2-37  
 line printer command bytes 2-37  
 NOP byte 2-33  
 repeat cycle bytes 2-35  
 Selectric command bytes 2-35  
 set indicator bytes 2-35  
 structure 2-33  
 total batch bytes 2-35  
 5496 command bytes 2-37  
 implementation errors 1-6  
 inner macro, definition 4-27  
 inner macros  
 hierarchy 2-19  
 IDFASM 2-24  
 IDFDSP 4-5  
 IDFTR 2-24  
 IJLFASM 2-24  
 IJLFTR 2-24  
 introduction 2-12  
 structure 2-19  
 input  
 FD macro assembly 2-7  
 object module, for FD utility 3-10  
 to FD utility 3-4  
 validation, FD utility 3-14  
 INPUT DCB, in IDFACT module 3-40  
 intermediate data sets, for DOS FD utility 3-5  
 introduction  
 assembly of macro instructions 1-5  
 authority 2-4  
 control 1-5  
 delimiting macro 2-5  
 diagnostic macros 2-5  
 FD macros  
 functions 1-5  
 general 1-3  
 FD program 1-3  
 FD utility  
 functions 1-6  
 general 1-3  
 field 1-5  
 form 1-4  
 form descriptions 1-4  
 forms design 1-4  
 IBM 3735 1-3  
 inner macros 2-12  
 line 1-5  
 MNOTE messages 2-7  
 outer macros 2-12  
 page 1-4  
 procedural macro 2-4  
 promotability 2-4  
 structural macros 2-4  
 transmission 1-6  
 I/O device requirements 1-6  
 I/O devices, required for FD utility 3-3  
 ISMOD macro  
 in IJLFLOAD module 3-51  
 in IJLFUPDT module 3-54  
 ISMPUT routine  
 in IJLFLOAD module 3-49  
 in IJLFUPDT module 3-52  
 K  
 keyed unpacked program block (see KUPB)  
 KUPB  
 definition 4-27  
 description of 3-7  
 format of 3-10  
 general 2-7  
 subfields 3-10  
 used by FD utility 3-4  
 L  
 line adapter, BSC 1-3  
 lines form-printer field 2-32  
 lines form-Selectric 2-32  
 Linkage Editor step  
 DOS method of operation 3-21  
 FD utility 3-3  
 functions 3-17  
 OS method of operation 3-17  
 program organization 3-43  
 segment table 3-17  
 LOAD option, DOS Storage step 3-28  
 LOADFST option, DOS Storage step 3-29  
 loading, of FD utility overlay segments 3-13  
 LOADRTN routine  
 in IJLFLOAD module 3-49  
 in IJLFUPDT module 3-52  
 logical flow, in FD utility 3-6  
 logical organization, FD utility object modules 3-4  
 M  
 macro, definition 4-27  
 macro method of operation  
 FDCTRL 2-22  
 FDEND 2-23  
 FDFIELD 2-19  
 FDFORM 2-19  
 FDLINE 2-19  
 FDPAGE 2-19  
 general 2-12  
 magnetic disk storage device 1-3  
 main storage  
 basic capacity 1-3  
 chaining information 1-3  
 control 1-3  
 DOS teleprocessing 1-4  
 DOS teleprocessing requirements 2-6  
 FD program 1-3  
 FD utility 1-6



- program residence 1-6
- required for FD utility 3-3
- teleprocessing 1-1
- maintenance, FD program 2-50
- message module
  - for DOS FD utility 3-4
  - for OS FD utility 3-3
  - program organization 3-54
- messages
  - data source 4-14
  - FD program 4-14
  - MNOTE 4-6
- method of operation
  - DOS Control step 3-17
  - DOS Linkage Editor step 3-21
  - DOS Storage step 3-21
  - FD macros 2-12
  - FD utility 3-6
  - OS Control step 3-14
  - OS Linkage Editor step 3-17
  - OS Storage step 3-21
- minimum storage
  - DOS teleprocessing 1-4
  - linkage editor 1-6
  - System/360 1-4
  - System/370 1-4
- MNOTE message, definition 4-27
- MNOTE messages
  - introduction 2-7
  - listing 4-6
- MNOTE severity codes 2-7
- module references, FD utility 3-76
- modulo 4-28
- motion control assembly 2-26
- MSGFAN routine
  - in IDFCT module 3-39
  - in IDFST module 3-46
  - in IJLFCT module 3-43
  - in IJLFLOAD module 3-49
  - in IJLFUPDT module 3-53
- multipoint line, definition 4-28

**N**

- nesting concept of outer & inner macros 2-12
- NEWMEM routine 3-44
- nonswitched line, definition 4-28

**O**

- OBJCARD routine 3-40
- object module
  - definition 4-28
  - general 2-7
- object modules
  - FD utility
    - IDFST 3-12
    - IJLFLOAD 3-13
    - IJLFST 3-13
    - IJLFUPDT 3-13
    - logical organization 3-4
    - physical organization 3-4
- objectives, part 2 2-3
- open files, in FD utility 3-14
- OPEN routine 3-47
- operation, FD utility 3-3
- operational considerations
  - FD macros 2-7
  - FD utility 3-4
- operator message 2-32
- OPTION control statement, DOS FD utility
  - Storage Step 3-28
- optional features
  - FD utility
    - DOS 3-4
    - OS 3-4
    - 3735 1-3
- organization
  - Control step 3-35
  - FD program 2-31
  - FD utility 3-35
  - Linkage Editor step 3-43

- Storage step 3-44
- OS
  - control section 2-7
  - duplicate form handling 1-6
  - minimum storage 1-4
  - program support 1-3
- OS FD utility
  - control information 3-4
  - method of operation
    - Control step 3-14
    - Linkage Editor step 3-17
    - Storage step 3-21
- outer macro, definition 4-28
- outer macros
  - delimiting (*see* FDEND)
  - diagnostic (*see* FDTRACE or FDDSPY)
  - introduction 2-12
  - organization 2-6
  - procedural (*see* FDCTRL)
  - structural (*see* FDFORM, FDPAGE, FDLINE, or FDFIELD)
- output
  - FD macro assembly 2-7
  - from FD utility 3-4
- output data, FD utility 3-10
- OUTPUT DCB
  - in IDFCT module 3-40
  - in IDFST module 3-46
- overlay programs
  - DOS FD utility 3-4
  - OS FD utility 3-3
- overlay segments
  - examination 3-13
  - loading 3-13
  - of FD utility Storage step 3-13

**P**

- page number restriction 1-5
- PARMPROC routine 3-44
- part 2 objectives 2-3
- path, definition 4-28
- paths, general 2-9
- PAUSE statement, use of in DOS FD utility 3-5
- physical characteristics
  - FD utility 3-3
    - DOS systems 3-3
    - OS systems 3-3
- physical organization, FD utility object modules 3-4
- point-to-point line, definition 4-28
- PRINT DCB
  - in IDFCT module 3-40
  - in IDFST module 3-46
- PRINT GEN assembly listing 4-15
- procedural macro (*see* FDCTRL)
  - definition 4-28
  - introduction 2-4
- processing data description and flow 3-10
- program organization, FD utility message modules 3-54
- program support
  - default source 2-33
    - DOS 1-3
    - OS 1-3
    - System/360 1-3
    - System/370 1-3
- programming considerations 1-5
- promotability
  - definition 4-28
  - introduction 2-4
- promotable operands 2-12
- purpose, FD utility 3-3
- PUTCARD routine
  - in IDFCT module 3-39
  - in IJLFCT module 3-42

**R**

- RCDCHK routine
  - in IJLFLOAD module 3-49
  - in IJLFUPDT module 3-51
- record types, FD utility input 3-14
- references, directory, FD utility 3-76

REPLACE function  
   OS 1-6  
   OS FD utility Storage step 3-21  
 replacement, of programs by FD utility 3-13  
 replacement table (REPLTAB) 3-47  
 REPLTAB routine 3-47  
 requirements  
   I/O devices 1-6  
   system, FD utility 3-3  
 resource 4-28  
 restrictions  
   DOS teleprocessing storage 1-4  
   FD program 1-3  
   forms design 1-4  
   page number 1-5  
   transmission 1-6  
   World Trade transmission 1-3  
 return codes, OS FD utility 3-5  
 root phase, of DOS FD utility overlay program 3-4  
 RPLACE control statement, DOS FD utility  
   Storage step 3-28  
 RPLACE function, DOS 1-6

**S**  
 sample FD program 4-15  
 scheduling, FD utility 3-4  
 scope 4-28  
 segment table (*see* SEGTAB)  
 segments  
   definition 4-28  
   general 2-9  
 SEGPROC routine 3-44  
 SEGTAB (Linkage Editor segment table) 3-17  
 severity codes 2-7  
 space allocation errors 1-6  
 special features 1-3  
 STGEN routine 3-46  
 STGINIT routine 3-44  
 storage  
   basic capacity 1-3  
   chaining information 1-3  
   control 1-3  
   DOS teleprocessing 1-4  
   DOS teleprocessing requirement 2-6  
   FD program 1-3  
   FD utility 1-6  
   program residence 1-6  
   teleprocessing 1-1  
 storage considerations 1-3  
 Storage step  
   DOS method of operation 3-21  
   FD utility 3-3  
   overlay segments 3-13  
   functions 3-21  
   OS method of operation 3-21  
   OS REPLACE function 3-21  
   program organization  
     IDFST 3-44  
     IJLFLOAD 3-49  
     IJLFST 3-47  
     IJLFUPDT 3-51  
 STOW macro, issued by FD utility 3-13  
 structural macros (*see* FDFORM, FDPAGE,  
 FDLIN, or FDFIELD)  
   definition 4-28  
   introduction 2-4  
 structure  
   FCD 2-37  
   FD macro 2-12  
   FD program 2-31  
   immediate bytes 2-33  
   inner macros 2-19  
   outer macros 2-6  
   output 2-7  
 subfields, KUPB 3-10  
 summary of FD macro functions 2-19  
 switched line, definition 4-28  
 SYNAD1 routine 3-45  
 SYNAD2 routine 3-45  
 SYSIPT DTFDI macro  
   in IJLFCT module 3-43  
   in IJLFST module 3-49  
 SYSLST DTFDI macro  
   in IJLFCT module 3-43  
   in IJLFLOAD module 3-51  
   in IJLFST module 3-49  
   in IJLFUPDT module 3-54  
 SYSPCH DTFDI macro, in IJLFCT module 3-43  
 system considerations 1-4  
 system requirements  
   FD macros 2-6  
   FD utility 3-3  
 System/360, program support 1-3  
 System/370, program support 1-3

**T**  
 tab intervals 2-32  
 target, definition 4-28  
 telecommunications, definition 4-28  
 teleprocessing 4-28  
 temporary name, from FD utility 3-4  
 temporary names  
   assigned by FD utility 3-10  
   in DOS FD utility Storage step 3-29  
   in OS FD utility Storage step 3-21  
 terminal 4-28  
 terminal control program  
   definition 4-28  
   introduction 1-3  
 trace macro (FDTRACE) 4-5  
 trailer, FD program 2-50  
 transformation, of FD utility input data 3-12  
 transmission  
   definition 4-28  
   introduction 1-6  
 transmission codes 2-19  
 transmission restrictions 1-6  
 transmission speed  
   domestic 1-3  
   World Trade 1-3  
 trouble shooting  
   assembly time 2-51  
   execution time 2-52  
   FD program 2-50  
   transmission time 2-52  
   utility processing time 2-52  
 TXT records, FD utility 3-14  
 TXTPROC routine  
   in IDFCT module 3-35  
   in IJLFCT module 3-41

**U**  
 unpacked program block (*see* UPB)  
 UPB  
   definition 4-28  
   general 2-7  
 UPDATE option, DOS Storage step 3-29  
 UPDTNM routine 3-53  
 use of this book iv  
 utility  
   FD (*see* FD utility)  
   introduction 1-6

**V**  
 validation, of FD utility input 3-10  
 validity group  
   alphabetic 2-44  
   alphameric 2-44  
   numeric comparison 2-44  
   self-check 2-46  
   value-check 2-44  
 variable (*see* global variable)

**W**  
 World Trade transmission speed 1-3

**3**  
 3735 capabilities 1-3  
 3735 internal code, sample 4-15

READER'S COMMENT FORM

IBM 3735 Programmable Buffered Terminal  
Form Description Macro Instructions  
and Form Description Utility  
Program Logic Manual  
(OS and DOS System)

Order No. GY30-3000-0

- How did you use this publication?

As a reference source   
As a classroom text   
As .....

- Based on your own experience, rate this publication . . .

As a reference source:     .....     .....     .....     .....     .....  
                                  Very    Good    Fair    Poor    Very  
                                  Good                                    Poor

As a text:                    .....     .....     .....     .....     .....  
                                  Very    Good    Fair    Poor    Very  
                                  Good                                    Poor

- What is your occupation? .....
- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

● Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

**YOUR COMMENTS, PLEASE . . .**

Your answers to the questions on the back of this form, together with your comments, help us produce better publications for your use. Each reply is 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 using your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Cut Along Line

Fold

Fold

FIRST CLASS  
PERMIT NO. 569  
RESEARCH TRIANGLE PARK  
NORTH CAROLINA

**BUSINESS REPLY MAIL**  
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.



POSTAGE WILL BE PAID BY . . .

IBM Corporation  
P. O. Box 12275  
Research Triangle Park  
North Carolina 27709

Attention: Publications Center, Dept. E01

Fold

Fold

IBM CORPORATION, RESEARCH TRIANGLE PARK, NORTH CAROLINA 27709



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

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