GC30-3001-2

Systems

IBM 3735 Programmer's Guide (OS and DOS Systems)

IBM

GC30-3001-2

Systems

IBM 3735 Programmer's Guide (OS and DOS Systems)

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

IBM

Preface

This programmer's guide provides system programmers, application programmers, and system operators with the information they need to use IBM 3735 Programmable Buffered Terminal facilities in a teleprocessing system.

All readers should have a general knowledge of System/360 and System/370 data processing techniques in a teleprocessing environment. Those responsible for designing or installing part or all of a system that includes 3735 terminals as remote devices must have a more detailed knowledge about teleprocessing systems. An annotated bibliography at the back of this book directs readers to publications containing this type of information. All users of this book should be familiar with the information contained in the *IBM 3735 Programmable Buffered Terminal Concept and Application* publication, Order No. GA27-3043, which describes the operating characteristics and features of the 3735 terminal.

The system programmer's main concern is with the system generation and storage requirements that are necessary to include the Form Description (FD) macro instructions, the Form Description (FD) utility, and the appropriate access method support in his system. This information is found in the "System Design Considerations" section. He also needs to establish sequences of job control statements that permit the application programmer to use these facilities. This information is found under "Assembling the Form Description Macro Instructions," "Using the OS Form Description Utility," and "Using the DOS Form Description Utility."

The application programmer is responsible for writing the form description programs (FDPs), and the application programs that transmit the completed FDPs and data to the 3735 and process the data captured at the 3735. Since these programming responsibilities may be divided among different people in a data processing installation, this publication distinguishes between the person who writes the FDPs (called a *forms encoder*) and other programmers. In installations where one person is responsible for programming the entire package, this distinction should be ignored. The application programmer should be generally familiar with the contents of the entire publication (with the possible exception of the "System Generation" and "Storage Estimates" sections). He should have a thorough understanding of the use of the FD utility, the available telecommunications access methods, and the relationships between the FDPs and the data transmitted from the 3735 to the central computer. The application programmer need not be proficient in Assembler Language programming, but he should be acquainted with IBM System/360 and System/370 Assembler Language macro instructions. If he lacks this general knowledge, he should have the services of an Assembler Language programmer available to him.

Programmers and system operators both need to know what action to take when various messages are directed to them. Appendix E describes the Operating System (OS) messages, and Appendix F describes the Disk Operating System (DOS) messages. The system operator should also read the "Introduction," which describes the general functions of the 3735 terminal, the FD macro instructions, and the FD utility.

Readers who use this book for reference purposes will find a page following the list of illustrations that allows quick access, via tabs, to the discussions they seek. In addition, the reference user will find that several of the appendixes contain condensed information, such as a macro format summary (Appendix B), a summary of 3735 operating procedures (Appendix I), and a summary of 3735 data and command functions (Appendix K).

Third Edition (July 1972)

This is a major revision of, and obsoletes, GC30-3001-1 and Technical Newsletter GN30-3000. Significant new material has been added throughout, and existing material has been changed extensively; therefore, no vertical lines appear in the margins, and the manual should be reread in its entirety. This revision contains information on the File Storage capability supported for the 3735. This information includes descriptions of two new FD macros, FDLOAD and FDSYNTAX, as well as additional operand specifications for the FDFORM, FDFIELD, and FDCTRL macros.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems or 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.

This manual has been prepared by the IBM System Development Division, Publications Center, Dept. E01, P.O.Box 12275, Research Triangle Park, North Carolina 22709. A form for reader's comments has been provided at the back of this manual. If the form has been removed, comments may be sent to the above address.

Contents

Illustrations

The IBM 3735 Programmable Buffered Terminal (hereafter referred to as the 3735) combines some of the features of an interactive terminal with the efficiency of buffered batch transmission to provide a new approach to source document (form) preparation and data capture. Designed primarily for applications using preprinted (fixed format) business forms and batch processing, the 3735 can be tailored, via user-designed form description programs (FDPs), to fit the needs of a variety of data processing environments.

This programmer's guide describes methods and techniques that can be used to design, write, and generate form description programs (FDPs). The major sections of the book discuss:

- The Form Description (FD) macro instructions, which a forms encoder uses to describe forms and specify the functions desired for processing forms at the 3735 ("How to Write a Form Description Program").
- The Form Description (FD) utility program, which transforms assembled form description programs (FDPs) into records that can be transmitted to a 3735 and interperted by its control program ("How to Use the Form Description Utility").
- Other factors that must be considered in designing a teleprocessing system that uses 3735 terminals as remote stations ("System Design Considerations").

What The 3735 Is And What It Can Do

The 3735 is a programmed terminal consisting of a desk-side control unit and a cable-connected IBM Selectric ® I/O-II Printer Keyboard that can be placed on the typewriter pedestal of a standard secretarial desk (see Figure 1). The 3735 control unit contains

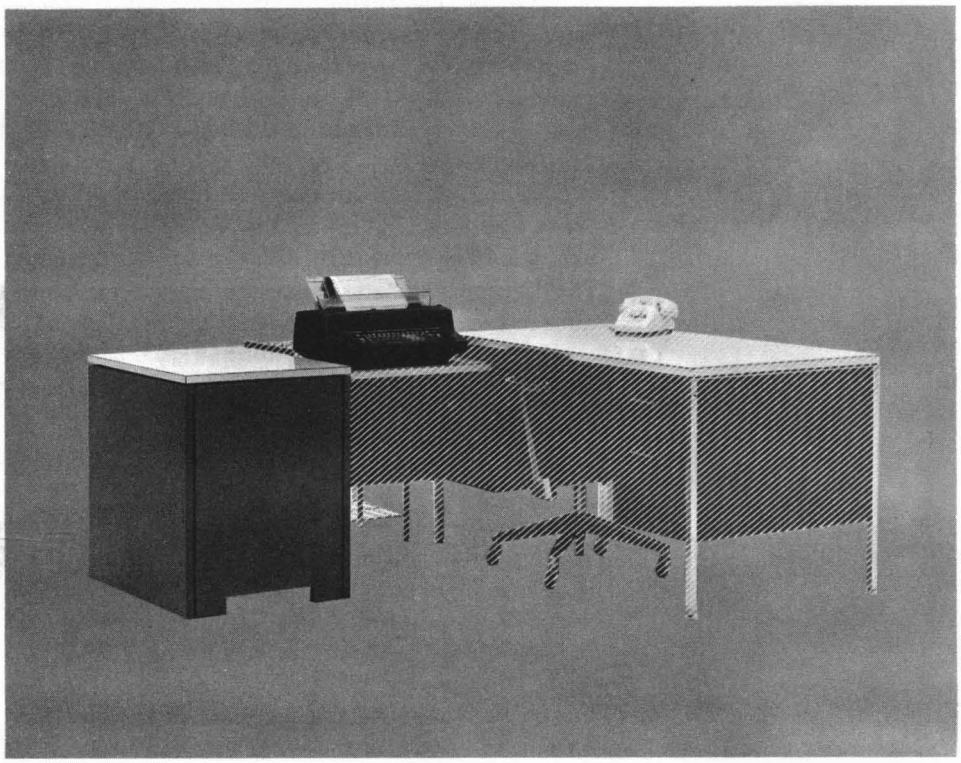


Figure 1. IBM 3735 Programmable Buffered Terminal–Typical Configuration (Telephone, Desk, and Chair Not Furnished by IBM)

a communication interface disk storage space for control information and recorded data, and a microcoded resident terminal control program (TCP) that permits flexible control of the terminal/operator interaction.

During document preparation, the 3735 provides:

- Operator guidance: The 3735 can display set-up instructions, exception messages, and indications of keying or procedural errors.
- Programmed forms control: The 3735 automatically positions the form for printing each line.
- Automatic print-element positioning: The 3735 automatically positions the print element for printing data within predefined fields.
- Data validation: The 3735 can examine input data for character set membership, character count, logical comparison with other data, or self-checking of numeric data.
- Format and editing: The 3735 provides centering, left and right justification, underlining, character filling, and numeric-data editing.
- Logical capabilities: The 3735 can conditionally process or bypass pages, lines, and data fields.
- Arithmetic capabilities: The 3735 can add, subtract, multiply, divide, and divide and round.
- Power typing: The 3735 can automatically print previously entered or internally generated information from storage.

The information captured during document preparation is stored, under program control, for later transmission to a central computer. An entire day's transactions can be stored for unattended transmission to the central computer, and processed data can be returned from the central computer for use in the next day's operation. Figure 2 illustrates the 3735 operating environment.

Configuration

The 3735 terminal can communicate with an IBM System/360 (except Model 20 or Model 67 in Time-Sharing mode) or System/370 (Model 135, 145, 155, 165, or 195) central processing unit (CPU) through an IBM 2701 Data Adapter Unit, IBM 2703 Transmission Control, IBM 3705 Communications Controller, or an Integrated Communications Attachment (System/360 Model 25, System/370 Model 135, or System/3 only) over switched communication lines at 1200 or 2000 bps (or at 2400 bps with an IBM 3872 Modem).

control unit. A magnetic disk storage device within the control unit contains the terminal control program (TCP), the form description programs (FDPs), and an area for storage of user data. The number of FDPs that can be stored on the disk depends on their length. The "Storage Estimates" discussion in the "System Design Considerations" section contains further information concerning FDP and data storage requirements.

The basic storage capacity for FDPs and user data is 62,832 bytes. This storage capacity can be expanded (by special features) in two increments of 41,888 bytes to a total of 146,608 bytes. Other features that can be provided are:

- Automatic Answer, which allows unattended communication with a CPU.
- A synchronous clock, which allows transmission rates of 600 or 1200 bps (600 bps is available only for IBM World Trade Corporation customers).
- Multipoint communication on leased lines at 1200, 2000, or 2400 bps.
- A keylock to prevent unauthorized use of the 3735.
- An Operator Identification Card Reader, which may be used to read either an IBM Magnetic Stripe Identification Card or an IBM Credit Card.
- A 5496 Data Recorder adapter, which allows an IBM 5496 Data Recorder to be attached to the 3735. (The 5496 is a buffered, operator-oriented, key-entry unit that punches and reads 96-column data cards.)
- A 3286 Printer adapter, which allows an IBM 3286 Printer Model 3 to be attached to to the 3735 for use as an auxiliary printer.

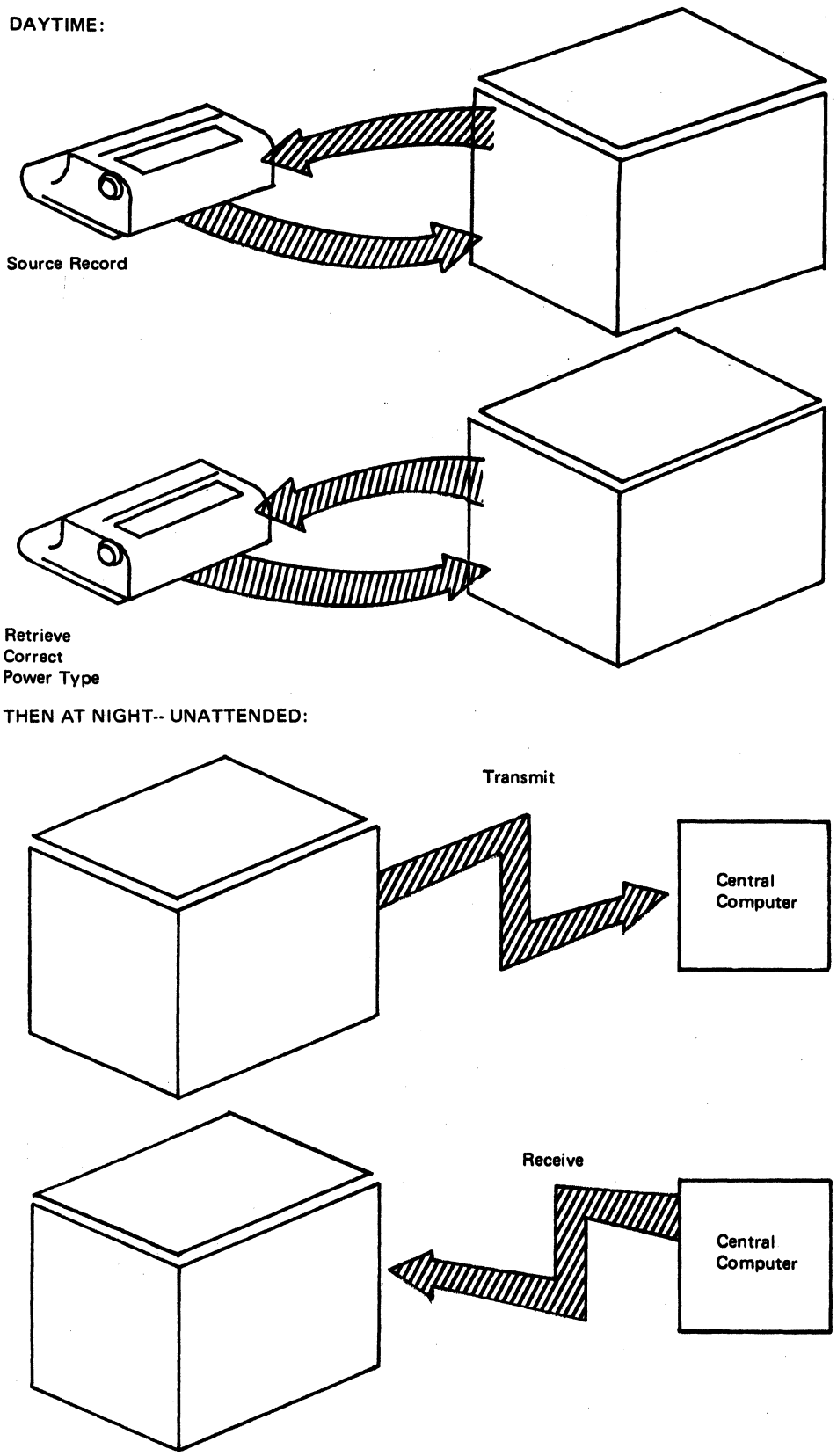


Figure 2. IBM 3735 Operating Environment

- A File Storage capability, which permits the user to code an FDP that performs read, write, and update operations on a user area of the 3735 disk storage file.
- A combination File Storage and external numpad capability, which permits the user to code an FDP that performs read, write, and update operations on a user area of the 3735 disk storage file. If this combination is selected, the 5496 Data Recorder adapter cannot be installed.

Program Support

IBM System/360 and System/370 program support for the 3735 is provided under the Operating System (OS) and the Disk Operating System (DOS). This support provides for assembling user-written FDPs, for preparing assembled FDPs for transmission to the 3735, and for transmitting programs and data between a central computer and 3735 terminals.

The 3735 uses the binary synchronous communications (BSC) method of line control. The BSC support provided in OS TCAM, OS BTAM, and DOS BTAM handles message transmission between the computer and the 3735. Figure 3 shows the general data flow and interaction between the central computer (CPU) and a 3735.

Two levels of program control are used in the 3735. The forms encoder generates FDPs by coding FD macro statements that specify the structure (layout) of a form and how it is to be processed. The terminal control program (TCP), recorded in the 3735 control unit during manufacture, interprets the FDPs and provides detailed 3735 control. This two-level approach relieves the forms encoder of much of the detailed programming required to capture form data at the 3735.

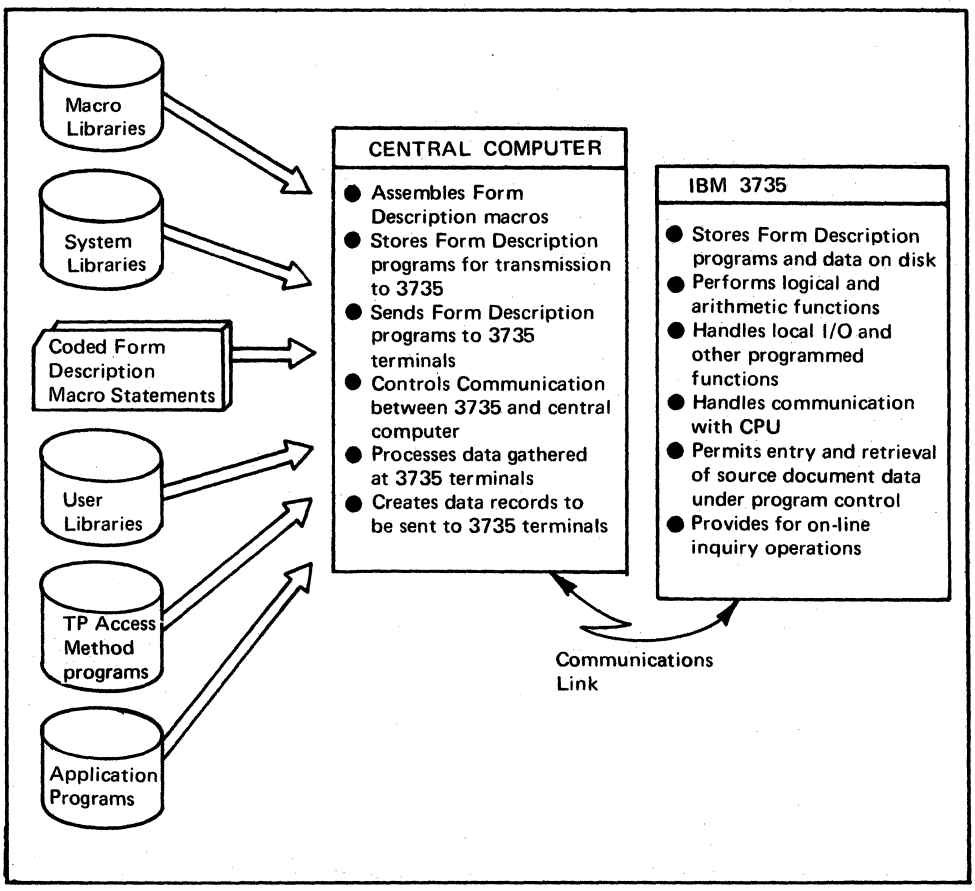


Figure 3. IBM 3735-CPU Interaction

Form Description Macro Instructions

The Form Description (FD) macros consist of four *structural* macro instructions (FDFORM, FDPAGE, FDLINE, and FDFIELD), two *procedural* macro instructions (FDCTRL and FDLOAD), one *delimiting* macro instruction (FDEND), and one *diagnostic* macro instruction (FDSYNTAX). The macros may be coded in many different ways to describe many different kinds of forms. Each FD macro statement consists of:

- A name entry (optional in all but the FDFORM macro), which is a symbol created by the forms encoded to identify the statement.
- An operation entry, which specifies the macro-instruction operation desired.
- Operand entries, which specify additional details that further define the macro-instruction operation. Depending on the needs of the FDP, one, several, or no operands can be coded.

Every form description program (FDP) must begin with an FDFORM macro statement and end with an FDEND macro statement.

The *structural* macro instructions describe the physical structure of a form. Their general functions are as follows:

- *FDFORM:* You code this macro only once, as the first statement in an FDP. It provides the FDP identification (in two ways, by form name and by code number), data condensation options, a device option, a buffer option, an object deck selection option, an assembly mode option, a mechanical left-margin setting, an operator message, and horizontal tabular-stop settings.
- *FDPAGE:* You code this macro to describe each page within a form. It names the page, specifies a number for the page, defines the vertical size (height) of the page, specifies the extent of the vertical margins, and provides for backward references to the macro statement.
- *FDLINE:* You code this macro to describe each line within a page. It names the line, specifies a number for the line, defines the horizontal size (width) of the line, specifies the extent of the horizontal margins, provides for repeated execution of a group of macro statements, and provides for backward references to the macro statement.
- *FDFIELD:* You code this macro to describe each data field within a line. It names the field, locates the field within the line, specifies the origin and destination of the field data, specifies the processing to be performed on the field data, provides for repeated execution of a group of macro statements, permits specification of conditional branches, and provides for backward references to the macro statement.

The *procedural* macro instruction FDCTRL provides much of the decision-making power in your FDP. You can code it anywhere between FDFORM and FDEND to test various indicators, alter the contents of the 3735 counters and program logic indicators, initiate repeated execution of a group of macro statements, select conditional branches that permit nonsequential processing of a form, accumulate batch totals, and perform I/O operations (including communication with a CPU).

The procedural macro instruction FDLOAD is used to create a specialized FDP that loads the user's data area on the 3735 disk storage file with data sent from the CPU. Such an FDP consists of only FDFORM, FDLOAD, and FDEND macro statements.

You code the *delimiting* macro instruction FDEND only once, as the last statement in your FDP. It indicates to the Assembler that no more macro statements describing the current form are to follow, and it reinitializes the Assembler for a possible following FDP in the same assembly.

The *diagnostic* macro instruction FDSYNTAX may be used to aid in locating errors in an FDP. It may be coded before or between other FD macros and instructs the Assembler to suppress code generation, but to check all macro statements for syntactic integrity.

Form Description Utility Program

The output module from your error-free FDP assembly must be further processed before it can be sent to the 3735. This additional processing is done in the central computer by the Form Description (FD) utility, which transforms the assembled FDPs into blocks of instruction sequences that the 3735 can interpret and use. The FD utility is executed in three steps:

- The *control* step examines the FDP object modules for integrity and generates the Linkage Editor control statements needed to produce an overlay program that is executed in the storage step.
- The *link-edit* step combines the output of the control step with an IBM-supplied module to create the program that is executed in the storage step.
- The *storage* step places the FDPs in a user-specified data set for later transmission from the central computer to a 3735.

Each step of the utility produces an output listing that describes the results of processing, including any errors that may have been detected. When the utility has finished its processing without error, the FDPs in your data set are ready for transmission from the central computer to 3735 terminals.

System Design

The user is responsible for providing the teleprocessing application programs that transmit the FDPs and other data to his 3735 terminals. In addition, the user needs to provide the application programs that process the data collected at the 3735. The design of such application programs depends on the telecommunications access method (OS TCAM, OS BTAM, or DOS BTAM) provided in the central computer. Operation with these access methods is discussed in the "System Design Considerations" section. This section also provides system design and usage information that will help the system programmer and application programmer to tailor their particular computing facilities to include 3735 terminals.

Further information concerning 3735 operations can be found in the IBM 3735 Programmable Buffered Terminal Concept and Application publication, Order No. GA27-3043.

After a forms designer has decided what forms are to be used for various 3735 applications and what kinds of information he needs from the forms, a forms encoder must write form description programs (FDPs) that the 3735 can interpret and use to process the forms. The Form Description (FD) macro instructions provide the forms encoder with a flexible, comprehensive source language that he can use to describe how the forms are structured, and how the 3735 is to process each data field of a form.

Only those instructions generated during the assembly of the FD macro statements may be included in an FDP. The forms encoder may, however, use the Assembler instructions TITLE, EJECT, SPACE, and PRINT to control the format of his output listing. In fact, use of the Assembler PRINT NOGEN instruction will substantially reduce the size of the FDP listings. Such Assembler instructions are not considered to be part of an FDP.

Macro Coding Conventions

An FD macro statement is coded in columns 1 to 71 of a standard 80-column card. If more than one card is needed to complete a macro statement, a nonblank character must be placed in column 72, and the statement continued on the following card, starting in column 16. All columns to the left of column 16 on the following card must be blank.

Statements may consist of from one to four entries in the statement field (columns 1-71), as shown in Figure 4. They are, from left to right: a name entry, an operation entry, an operand entry, and a comments entry. These entries must be separated by one or more blanks, and must be written in the order stated. A name entry, if used, must begin in column 1. If no name entry is used, the operation entry can begin anywhere beyond column 1, but must be completed before column 71.

A name entry (symbol) may be from 1 to 8 characters long. It must begin with an alphabetic character (A-Z, or the characters #, @, and $), and may not contain any blanks or special characters. Numeric characters may be used following the initial alphabetic character.

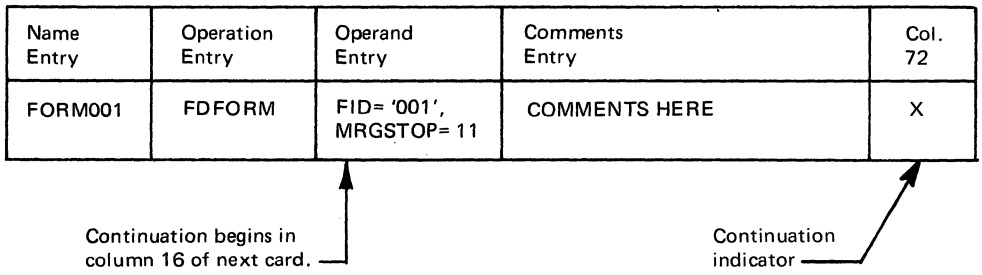


Figure 4. Macro Instruction Statement Format

Operands are of two general types, positional and keyword. Positional operands must be coded in a particular order. Keyword operands, on the other hand, may usually be coded in any order. A keyword operand is identified by its format, which is:

KEYWORD = value

For example, in the FDPAGE macro instruction, pagenum is a positional operand, and HEIGHT = value is a keyword operand. Keyword operands may themselves contain positional or keyword suboperands (without an = sign).

Every macro instruction operand, except the last in a given macro statement, must be immediately followed by a comma. No blanks are permitted between operands and the commas that separate them, unless you are using more than one card to code the macro statement. In this case, you can place a comma immediately after the operand, place a nonblank character in column 72, and begin the next operand on the following card (in column 16). When an optional keyword operand is not coded, its associated comma should also be omitted. If excess suboperands are coded in some operand, they are ignored, and the FD macros issue a warning message to that effect.

An entire statement field can be used for a comment by placing an asterisk (*) in column 1.

Sample programs in Figure 21, and in Appendix C, illustrate these coding conventions, and show how comments can be used effectively to describe the function of each statement entry. Further information on Assembler Language coding conventions can be found in the *OS Assembler Language* publication, Order No. GC28-6514, or the *Disk and Tape Operating Systems Assembler Language* publication, Order No. GC24-3414.

Operand Promotion

Many Form Description (FD) macro instruction operands may be coded in more than one type of structural macro statement. Such coding results in "promoting" an operand to a position of higher authority than normal.

The *authority* of a macro statement extends through an FDP until the same type of macro statement (or a macro statement of higher authority) appears later in the FDP. The values specified in a promoted operand are effective throughout the authority of a macro statement, except where modified temporarily in a macro statement of lower authority. The levels of authority are suggested by the macro names themselves; FDFORM is the highest-authority macro, FDPAGE the next lower, FDLINE the next lower, and FDFIELD the lowest. The delimiting macro FDEND terminates the authority of FDFORM. (No promotion to or from FDCTRL is possible since its operands do not directly relate to form structure.)

Consider the case where all pages of a form have the same vertical and horizontal output space requirements. If you code the operands that control these functions in the FDFORM macro statement, their authority extends throughout the entire FDP. You need not specify these operands again in another macro statement unless you wish to temporarily override the promoted specification.

Operand promotion can save a substantial amount of coding in many fixed-format form descriptions. However, not all operands can be promoted to a macro statement of higher authority. Figure 5 lists the operands that can be promoted, their origins, and their functions. Any operand in the table can be coded in any macro of higher authority than the macro in which it normally appears. For example, the SOURCE operand, normally coded in FDFIELD, can be coded in any of the higher-authority macros (FDLINE, FDPAGE, or FDFORM).

FUNCTION	OPERAND	PROMOTED FROM
page height (vertical output space) vertical margin checking	HEIGHT VMRG	FDPAGE
line width (horizontal output space horizontal margin checking	WIDTH HMRG	FDLINE
orgin of input data kind of input data (character set) self-checking of input data destination of output data justification of output data fill chars in unused positions underscoring of data fields	SOURCE* KIND SELFCHK SINK JUSTIFY FILL UL	FDFIELD

*Except SOURCE = 'string' and SOURCE = X1 or X2.

Figure 5. Promotable Operands

If an operand that is *not* promotable is coded in some other macro than the one in which it is permitted, the Assembler issues an error message that flags the operand as an undefined symbol.

Operand Chaining

Some 3735 applications may require the coding of one or more operands that contain more characters than Assembler character-handling limitations permit. When coding FDPs to support such applications, you can overcome these limitations by chaining together successive macros of the same type. (Operands that may require this treatment are also those which cause large numbers of FDP bytes to be generated at the time the operand is assembled.) The chaining function is invoked and controlled according to the following rules and procedures:

- When chaining is used, operands must be coded in the order shown in Figure 6.
- The general technique used to specify chaining is to code the character C as the last suboperand of the chained operand, then to code the macro and its continued operand again. For example:

```
F001    FDFORM  FID = '001',MESSAGE = ('long text block',C)
        FDFORM  MESSAGE = 'remainder of long text block'
```

MACRO	OPERANDS	PERMITTED CHAINING SPECIFICATIONS
FDFORM	FID OBJECT PACKING MRGSTOP MODE DEVICES BUFFERS	none
	MESSAGE	MESSAGE = ('string', C) or MESSAGE = (cc (d), C)
	HTAB	HTAB = (d, d, d, . . . , d, C)
	HEIGHT VMRG WIDTH HMRG SOURCE SELFCHK KIND SINK FILL JUSTIFY UL	none

Figure 6. Permitted Operand Chaining Formats (Part 1 of 2)

The C suboperand tells the FD macros to interpret the next sequential macro statement as a continuation of the macro statement that requested chaining.

- The chaining function is performed by the FD macros, and is separate from the card continuation facility provided by the Assembler. Thus, you should *not* code a "continuation" character in column 72 of the card image that requests chaining unless you are using the next card only for a comment.
- Macro statements chained together in this manner may not have a name entry coded in any macro except the first one in the chain.

Figure 6 lists, for each macro instruction that uses chaining, (1) the operands that may be coded in the macro, (2) the order in which the operands must be coded, and (3) the places in the operand fields at which the chaining indicator, C, may be inserted.

MACRO	OPERANDS	PERMITTED CHAINING SPECIFICATIONS
FDFIELD	SAVELOC CYCLE BATCH	} none
	SOURCE	SOURCE = ('string', C)
	COUNT SELFCHK KIND SINK FILL JUSTIFY UL	} none
	COMPARE	The C may be coded only after AND or OR. For example: COMPARE = (EQ, 'A', OR, C)
	CTR	CTR = ((d, operation) , C)
	IND	The C may be coded following a complete suboperand specification, as follows: IND = ((1, EQ, 'A') , C) It may also be coded following an AND or OR within a suboperand specification, as follows: IND = ((1, EQ, 'A'), (2, EQ, 'B', AND, C))
	PICTURE	PICTURE = ('picturespec' , C)
FDCTRL	SAVELOC	none
	CYCLE	none
	IF	The C may be coded only after an AND or OR (if any). For example: IF = (IND (3) ,OR , C)
	IND	IND = ((d, value) ,C)
	CTR	CTR = ((d, or, opnd) , C)
	TOTAL	TOTAL = ((totalspec) ,C)
	COMMAND	COMMAND = ((cmndgrp) ,C)
	GOTO	none

Figure 6. Permitted Operand Chaining Formats (Part 2 of 2)

Macro Format Conventions

Several conventions are followed in describing the order and arrangement of the Form Description (FD) macro instructions:

- Code uppercase letters and all special characters exactly as shown in the individual macro descriptions. Exceptions to this convention are brackets, [ ] ; braces, {} ; ellipses, . . . ; and subscripts. These are not coded.
- Lowercase letters and words represent variables for which you must substitute specific information or specific values.
- Braces, {} , indicate that you must code one of the alternatives shown within the braces. The braces themselves are not coded.
- Brackets, [ ], indicate an optional item. You can omit the items within the brackets at your discretion. Any item not within brackets must be coded. The brackets themselves are not coded.
- Items in a stack, one over another, represent alternative operand entries. You may code only one of a group of stacked items. If the stacked items are within braces, you must code one of the items. If the stacked items are within brackets, you may code one or none.
- If one of a group of stacked items within brackets is underscored, and you do not supply a specific value, that item is implied by default. Such items are therefore called *default values.* They are provided by the FD macros.
- An ellipsis, . . . , indicates that the preceding item or group of items can be entered more than once in succession. The ellipsis is never coded.
- Character strings (indicated by 'string') must be enclosed by apostrophes ('). If an apostrophe or an ampersand appears in the character string, it must appear doubled (for example, 'O' 'KELLY WORKS FOR SMITH && JONES, INC.').

- Certain graphic characters available on the 3735 are not found on many card punch machines. Such characters can be coded in several ways. Lowercase alphabetic characters, for example, may be coded by multi-punching the appropriate combinations, or by special coding, as explained in the next bullet item. For a detailed list of the valid EBCDIC and ASCII characters, refer to Appendix M in the rear of this publication.
- o Lowercase alphabetic characters may be specified not only by multi-punching, but also by coding an underscore character (–) at the beginning and end of each character string that is to be lowercase. The underscore character is not translated; instead, the first one encountered directs the FD macros to interpret any following uppercase Roman letters as lowercase until underscore character is encountered (or the character string ends). Uppercase is assumed at the start of a string, and use of the underscore character is prohibited where lowercase characters are illegal (such as in Katakana strings). Any multi-punched lowercase letters in the character string are not affected by use of the underscore character.
- The letter d indicates a term coded as one or more decimal digits, with leading zeros optional. Whenever a specific number of digits is required, that number is indicated by the number of d characters shown in the format illustration.
- Other notational symbols are explained when they occur.

Structural Form Description Macro Instructions

The structural Form Description (FD) macro instructions describe the structural organization of the form and the processing required for each data field. They are normally coded so that forward progression is maintained through the entire form (that is, from page to page, from top to bottom on a page, and from left to right on a line). However, nonsequential encoding can sometimes be used to advantage in the structural macros FDLINE and FDFIELD, and in the procedural macro FDCTRL.

FDFORM Macro Instruction

An FDFORM macro instruction must be coded as the first macro instruction for each FDP you write. The FDFORM macro specifies:

- The name by which the FDP is stored in the user's data set (symbol). A name entry must be coded.
- A 3-digit decimal character string that the 3735 operator uses to request the program at the terminal (FID). An FID number must be coded.
- The condensation that is desired for data sent from the 3735 to the central computer (PACKING).
- Specify Katakana character code for IBM Japan terminals (DEVICES).
- Use of certain buffers for additional data storage (BUFFERS).
- Selection of the type of object deck that is to be prepared (OBJECT).
- Creation of a specialized FDP to load when File Storage is present the user's data area of the 3735 disk storage file with CPU-generated date (MODE).
- A mechanical left margin setting (MRGSTOP).
- A 3735 operator message (MESSAGE).
- Horizontal tabular stop setting (HTAB).

If you want card-image identification and assembly-listing headings in your Assembler output, you should place an Assembler TITLE statement in front of the FDFORM statement. For example:

```
F001    TITLE  'FID 001 – GENERAL WHOLESALE COMPANY INVOICE'
```

You should also consider using a PRINT NOGEN statement to reduce the size of the assembler output listing. See the discussion under "Listing Control Instructions" in

either the *DOS Assembler Language* publication, Order No. GC24-3414, or the *OS Assembler Language* publication, Order No. GC28-6514, for further information on the use of the TITLE statement, the PRINT statment, and other listing control statements.

Figure 7 shows the format of the FDFORM macro instruction.

Name	Operation	Operands
symbol	FDFORM	FID = 'ddd' [,PACKING = { NO / YES / DELIMIT }] [,DEVICES = (3735, K [D [ )] [,BUFFERS = ([RPB] [,(LPB [, { 132 / 126 / 120 / d }])])] [,OBJECT = { OS / DOS }] * [,MODE = { NONLOAD / LOAD }] [,MRGSTOP = { 0 / d }] [,MESSAGE = ({ cc[({ 1 / d }) / 'string' } [, { cc[({ 1 / d })] / 'string' }] ...)] [,HTAB = (d[,d] . . .)]

Figure 7. Format of the FDFORM Macro Instruction

symbol

The name entry (symbol) specifies the name of the form and must be coded. The characters you code become the name by which the operating system and the system programmer refer to the form.

FID = 'ddd'

The FID operand specifies the form identification number by which the FDP is selected at the 3735 terminal. The FID operand must be coded. The ddd characters are the digits that the 3735 operator uses to request the FDP. You must code three decimal digits in this operand field, and the framing apostrophes are required.

The value of ddd may range from 000 to 989. FID number 999 is reserved for the 3735 Functional Test Form, and FID numbers 990 to 998 are reserved for other purposes. If forms using any of these reserved FID numbers are sent from the central computer, they are accepted as valid by the 3735. However, if the 3735 operator selects one of these reserved numbers, the reserved function is performed. You should be especially careful to code a unique FID number for each FDP you write, since no tests are duplicate FDP numbers are performed by the FD macros, the FD utility, or the 3735. If duplicate FDP numbers are sent to the 3735, the 3735 will accept them and catalog them, but only the first such FID number found in the disk directory can be used. All other FDPs with the same number are inaccessible to the 3735 operator.

When the FDP is executed at the 3735, this three-digit identification number is placed in the first three bytes of a data record that is to be sent to the CPU. This number is always sent to the CPU when the 3735 transmits its operator-created data, as described in the "System Design Considerations" section under the heading "Form Records".

An FDFORM macro statement may be coded as simply as:

```
FORM001   FDFORM      FID = '001'
```

The name entry (symbol) and the FID operand (FID) are required. When specified in this way, the default values are assumed for the PACKING and MRGSTOP operands, and no MESSAGE or HTAB functions are provided when this FDP is used at a 3735. In

addition, RPB may not be specified as a data source or sink, no Katakana coding is allowed, and the line printer buffer (LPB) checking limit is 132.

Figure 8 is a chart that will help you decide how to code the other FDFORM operands. A detailed discussion of each operand follows the chart.

If you want to. . .	Code. . .	Unless you want to use the default value of. . .
Remove extra null character from the end of each data field sent to the CPU. . . and. . . have the 3735 insert a delimiter between the data fields. . .	PACKING = YES (If you want any field delimiters, you must put them in yourself.) PACKING = DELIMIT, which removes null character and puts a X'1C' delimiter between data fields.	PACKING = NO, which does not remove any blanks from the data fields or insert any delimiters.
Specify Katakana code (for IBM Japan terminals). . .	DEVICES = (3735,K), which specifies that this FDP will use Katakana code.	No Katakana code.
Use the RDR and PCH buffers as a single 192-character storage buffer (RPB). . . Use more than 132 bytes of the 236-byte line printer buffer (LPB) for data storage. . .	BUFFERS = RPB (Use of this buffer is discussed following the chart.) BUFFERS = (LPB,d), where d is a value from 133 to 236.	Normal use of the RDR (input only) and PCH (output only) buffers. Use of only 12 bytes in the line printer buffer.
Prepare an object deck for a particular FD utility (either OS or DOS).	OBJECT = systype, where systype is either OS or DOS.	An object deck created for system on which the FDP assembly is performed.
Use this FDP to load the 3735 disk storage file with CPU-generated data records.	MODE = LOAD, which allows the FDLOAD macro (and no others) to be used in this FDP.	MODE = NONLOAD, which indicates that the FDP is not used to load the 3735 disk with CPU data.
Specify a mechanical left margin to be set on the 3735 Selectric printer. . .	MRGSTOP = d, where d is a number from 0 to 129. Output begins in position d + 1.	MRGSTOP = 0, which means that the operator should set the stop at the form's left edge.
Provide a 1-line message for the 3735 operator (for setup instructions or status information). . .	MESSAGE = 'string', where 'string' is the message text of up to 127 characters. If more than one message line is required, see the operand description following the chart.	No message provided to the operator (not recommended). *Note:* The framing apostrophes are required, but not counted.
Define the horizontal tabular stops to be set on the 3735 Selectric printer. . .	HTAB = (d,d,d,. . . ,d), where d, d, d, . . . , d are successive tab stop positions. The stops may be coded from MRGSTOP + 2 to MRGSTOP + 129.	No tab stops to be set at the 3735.

Figure 8. Coding the FDFORM Macro Instruction

PACKING = { NO / YES / DELIMIT }

The PACKING operand specifies how the 3735 is to pack (condense) data records created under this FDP before transmitting them to the central computer. PACKING = NO is the default specification, and indicates that no packing is to be done. (NO implies that the

application program that processes the data collected under this FDP excepts to find fixed-length data fields.) PACKING = YES specifies that consecutive trailing blanks in every data field are to be deleted, and PACKING = DELIMIT specifies both that consecutive trailing blanks are to be deleted and that a delimiter is to be inserted between data fields. The delimiter used by the 3735 is the IFS character in EBCDIC or the FS character in ASCII (both have a hexadecimal representation of '1C').

The packing option selected remains in effect throughout the entire form. If you want to supply your own unique graphic delimiters, code PACKING = YES and insert your delimiters as data between fields that you are going to send to the central computer. You can use a data sink specification or require the operator to do this. Such coding may be inefficient when large numbers of fields require delimiters.

If a form created at a 3735 can contain several completely blank or partially blank left-justified data fields, you should consider specifying PACKING = YES or PACKING = DELIMIT. Such coding eliminates unnecessary trailing blanks from the data field, and increases the amount of "real" data that can be transmitted to the central computer in a given message. However, when PACKING = YES is coded, the application program at the central computer cannot easily tell where one data field ends and the next begins. The YES option is available in case you want to use your own graphic delimiting character instead of the X'1C' provided by the 3735. Use of PACKING = YES prohibits the use of transmitted data at another 3735.

DEVICES = (3735 , K[D])

Coding DEVICES = (3735,K) specifies that this FDP is to be used on a 3735 that uses Katakana code. The resulting FDP can then be used only on a 3735 with this code set. (This code facility is intended primarily for IBM Japan terminals.)

When coding FD macro statements for an FDP that will be used on a 3735 with the Katakana character set, you should code DEVICES = (3735,K) or DEVICES = (3735, KD) so that correct code can be generated during the FD macro assembly. Coding DEVICES = (3735,KD) specifies that the phonetic equivalents of the Katakana characters are to be printed in the Assembler listing. Complete details on the Katakana support are in Appendix J.

BUFFERS = ([RPB]
[, (LPB[, { 132 / 126 / 120 / d }])])

The 3735 provides several different buffers that your FDP can use for temporary data storage while it is being executed at the 3735. Four of these buffers are:

- The card reader buffer (RDR), which is a 96-character read-only storage area that is used to hold data read in from a 5496 data card by an FDCTRL READ (RDR) command.
- The card punch buffer (PCH), which is a 96-character write-only storage area that is used to hold data that is to be punched on a 5496 data card by an FDCTRL PUNCH command.
- The storage buffer (STG), which is a 236-character write/read storage area that can be used for storage of data in your FDP.
- The line printer buffer (LPB), which is a 236-character write/read storage area that is used primarily to hold data that is to be printed on a 3286 matrix printer.

If you have forms that require more write/read storage space for data than the 236 bytes provided in the storage buffer (STG), you can use the storage space that is normally used for reading and punching 5496 data cards by coding BUFFERS = RPB (read/punch buffer) in the FDFORM macro. You can also use the 236 character positions in the line printer buffer (LPB) for similar purposes if space for the buffer was allocated during the 3735 system installation. See the "System Generation Considerations" section under the heading "3735 System Generation Options" for further details.

Coding BUFFERS = RPB specifies that the 96-character reader buffer (RDR) and the 96-character punch buffer (PCH) are to be treated as a single 192-character read/punch

buffer (RPB). If only RPB is coded, the enclosing parentheses may be omitted. This specification allows you to use the storage normally provided for 5496 input/output operations just as you would the storage buffer (STG). Use of the combined read/punch buffer in this manner prohibits use of the SOURCE and SINK specifications that direct data to PCH or request data from RDR. However, you can still read or punch 5496 cards by using the first 96 characters of RPB (1 to 96) when you want to get from the reader buffer, and the last 96 characters of RPB (97 to 192) when you want to put data in the punch buffer. As when using RDR or PCH, you must then issue the appropriate FDCTRL command, READ (RDR) or PUNCH.

Coding BUFFERS = (LPB,d) specifies that up to 236 positions of the line printer buffer (LPB), as specified by d, are to be used for data storage. Note, however, that only the first 132 characters in the buffer are moved to the 3286 printer when a PRINT command is executed. You should take care, therefore, to use LPB positions 133-236 only for data that is not to be printed. The qualifiers 120 and 126 indicate that the 3286 printer uses a 120-character or a 126-character platen. The number of characters moved when a PRINT command is executed is adjusted accordingly. If no qualifying value is coded, or if BUFFERS = LPB is not coded at all, the useful length of the line printer buffer is assumed to be 132 characters.

When coding an FDP that uses the 3286 printer, you should understand that the 3286 is simply a sequential output device, and that you control the format of its forms in two ways:

1. By the contents of the data strings you place in the line printer buffer (LPB).
2. By the sequences of PRINT, CLEAR, SKIP(d), and SKIPTO(d) commands that you code in FDCTRL macro instructions.

Forms used on a 3286 need not be related to forms used on the 3735 Selectric ® printer in size or content. The appearance of 3286 output is limited only by the size of its platen and the paper stock used (except that reverse forms motion is not permitted on either the 3286 printer or the Selectric ® printer).

The use of these buffers is explained more fully in the discussions of the FDFIELD SOURCE and SINK operands, and the FDCTRL COMMAND operand.

Note: If both buffer options are desired, code BUFFERS = (RPB, (LPB,d)). If BUFFERS is coded, then at least one of the suboperands must be coded.

OBJECT = { OS / DOS }

The OBJECT operand specifies that an object deck is to be prepared for a particular FD utility, either OS or DOS. If this operand is coded, then either OS or DOS must be coded. If this operand is not coded, the resulting object deck is prepared for the FD utility of the system on which the FDP assembly occurs. For compiling multiple FDPs in a single assembly, all programs must follow the same object format.

MODE = { NONLOAD / LOAD }

The MODE operand specifies whether or not the current FDP is to be used for creating or updating data records on the 3735 disk storage file from CPU-generated data. The File Storage capability must be presented in order to execute a file load FDP. These operations are specified by coding the FDLOAD macro. If this type of FDP is desired, then MODE = LOAD must be coded. For all other FDPs, the MODE operand should be omitted or specified as MODE = NONLOAD. FDPs having FDLOAD macros must be executed in playback mode at the terminal. If these FDPs are not executed in playback mode, the terminal operator receives a macro-generated message and the FDP is canceled. The only macros that can be coded following an FDFORM macro with MODE = LOAD specified are FDLOAD macros and an FDEND.

MRGSTOP = { 0 / d }

The MRGSTOP operand specifies, relative to the form (rather than to the 3735's position index scale), the character position at which the 3735 operator is to set the terminal's mechanical left-margin stop. The value of d may range from 0 to 129, and has a default of 0. The 0 setting is equivalent to a margin stop at the left edge (or "tear line") of the form. Actual output can begin in position d + 1.

For example, if you are describing a form that never uses the leftmost 15 character positions of each line, you can achieve more efficient Selectric printer motion by specifying MRGSTOP = 15. Such coding would allow output to begin in character position 16 (MRGSTOP + 1).

MESSAGE = ({ cc[({ 1 / d })] / 'string' } [,[{ cc[({ 1 / d })] / 'string' }] ...)

The MESSAGE operand specifies a character-string message that the 3735 operator can have printed before starting to process the form. The string length (excluding the framing apostrophes, which are required) may range from 1 to a maximum number of characters that depends on the particular Assembler used. You can use the MESSAGE operand to provide the 3735 operator with descriptive information about the form, instructions on how to set up the form, and so forth.

The 3735 operator can obtain the message not only by selecting the FDP, but also by requesting a listing of all resident FDPs and their associated operator messages. (When this listing is requested, only the first 30 characters of the operator message are printed, and any carriage control characters in the message are printed as blanks.) The 3735 control program performs a carriage return (CR) function both before and after displaying the message data to the 3735 operator. If you want to use any additional position control functions (for example, to print your message on more than one line), you must use the appropriate carriage control sequences. The carriage control (cc) characters permitted are:

HT (d) - horizontal tab d times
SP (d) - forward space d times
BS (d) - backspace d times
NL (d) - new line d times
LF (d) - line feed d times
CR - carriage return

The repetition factor (d) allows you to request that the carriage control operation be performed more than once. If no repetition factor is coded, a default value of 1 is used. The CR character may not specify any repetition factor at all. The other carriage control characters may specify repetition factors from 1 to 127. However, you should take care not to violate the page height and width limitations of the paper stock on which the message is to print. For example, to print your message on two lines, you could code:

MESSAGE = ('TEXT FOR LINE 1' ,NL, 'TEXT FOR LINE 2')

In this example, the NL characters inform the 3735 that a new line function should be performed. If only one MESSAGE suboperand is coded, the outer framing parentheses may be omitted.

Note: The new line (NL) operation combines the functions of one carriage return (CR) and d line feeds (LF). The horizontal tab (HT) function should be used with extreme caution, since the tab setting routine is not executed at the 3735 until after the message is printed.

To enable an operator to know the status and use of each FDP in the 3735, the MESSAGE operand data should include such information as:

- The purpose of the FDP.
- The type of form required.

- Mechanical set-up instructions.
- FDP version level.
- Date of creation or last revision.

As much of this information as possible should be included in the first 30 characters of the message. With this information available, an operator should be able to work with different FDPs with little difficulty.

HTAB = (d[,d] . . .)

The HTAB operand specifies the horizontal tabular stop positions on the Selectric ® printer. Values of d may range from MRGSTOP + 2 to 130, except that the 3735 cannot perform the HTAB exercise if MRGSTOP is greater than 129. The number of character positions between successive stops, or between MRGSTOP +1 and the lowest-numbered stop, cannot exceed 127. When an FDP has been selected, the 3735 operator can request the terminal to perform the tabular-stop-setting exercise. The exercise can be bypassed when working on subsequent copies of the same form. You should take care to place as few tabular stops as possible within field boundaries, since the 3735 must then space across the remainder of the field instead of tabulation, thus reducing mechanical efficiency.

The values coded in HTAB specify tabular stop settings in relation to the left edge of the form. Thus, to have the operator set the margin stop at column 9, and a tab stop at column 54, you should code MRGSTOP = 9 and HTAB = 54. In this case, output may begin in column 10 (MRGSTOP +1). Specification of one or more tabular-stop settings can often increase 3735 operating efficiency, thus allowing an operator to enter more data in a given amount of time.

Notes:

1. The smallest amount of code necessary to control mechanical motion on the Selectric ® printer is generated when 15 or fewer stops are specified, and when no adjacent stops are more than 31 character positions apart.
2. If only one tabular stop is specified, the enclosing parentheses may be omitted.

The following are some examples of correctly coded FDFORM macro statements.

```
FORMONE   FDFORM  FID = '001' ,PACKING = DELIMIT, MRGSTOP = 10,
                  HTAB = (15,33,49)
F989      FDFORM  FID = '989' ,MESSAGE = 'GWC ORDER - USE FORM
                  383901 - SET X MRGSTOP AT 0 - VERS 4 - 05/25/72'
ACME036   FDFORM  FID = '036' ,PACKING = YES,BUFFERS = RPB
```

FDPAGE Macro Instruction

The FDPAGE macro instruction marks the beginning of each page description within a form. This macro may not be coded before FDFORM is coded. An FDPAGE macro can be coded to:

- Name the page (symbol).
- Specify a page number within the form (pagenum).
- Define the vertical output space for the page (HEIGHT).
- Specify the setting of the vertical-margin checking limits (VMRG).
- Provide for backward references to the macro statement (SAVELOC).

Figure 9 shows the format of the FDPAGE macro instruction.

Name	Operation	Operands
[symbol]	FDPAGE	[pagenum] [, HEIGHT = {66 / d}] [, VMRG =) {1 / dt}] [, {height / db}])] [, SAVELOC = {NO / YES / d}]

Figure 9. Format of the FDPAGE Macro Instruction

symbol

The name entry (symbol) specifies the name of the page, if coded. A name entry is required if SAVELOC is coded, or if the macro is an explicit CYCLE target or limit or a branch target (that is, if the macro is referred to elsewhere).

When used to describe an 11-inch-high page, an FDPAGE macro may be coded simply as:

FDPAGE

In this case, default values provide a page number one greater than the previous page number (starting with page 1), a height of 66 lines, vertical margin checking limits of 1 and 66, and no retention of the macro-statement location. Figure 10 is a chart that will help you decide how to code the FDPAGE operands. A detailed discussion of each operand follows the chart.

pagenum

The pagenum operand marks the beginning of each page description within the current form. This positional operand may be coded with values from 1 to 16383. If you do not code it in the first FDPAGE macro of an FDP, a default of 1 is used. If you do not code a page number in a subsequent FDPAGE macro statement in the same form description program, a default value one greater than the previous page number is used. Coding this operand does not physically place a number on the form at the 3735. You can perform this function, if desired, by coding an appropriate FDFIELD macro instruction somewhere in the page.

Page numbers usually form an increasing series within a single form description, but they need not be consecutive. If pages are not numbered consecutively, undescribed interventing pages are assumed, for control purposes, to have a height equal to the value of the HEIGHT operand in force at the FDFORM level. Where this is inappropriate for one or more such pages, you can define dummy (null) pages by coding adjacent FDPAGE macros that specify the necessary HEIGHT values, but with no intervening line descriptions.

The purpose of such coding is to provide an accurate count of lines throughout a form in which page HEIGHT varies from page to page, and in which more than one processing sequence is possible. When coding standard forms of uniform page size, you need not be concerned with specifying dummy pages. In such cases, all you need to do is code the proper HEIGHT value as necessary.

If you want to. . .	Code. . .	Unless you want to use the default value of. . .
Specify a page number within the form. . .	pagenum as a number from 1 to 16383.	1 for the first FDPAGE macro in an FDP, and the previous page number plus 1 for all other pages.
Define the height of the page. . .	HEIGHT = d, where d is a number from 1 to 16383 that specifies the number of lines on the page.	HEIGHT = 66, which defines a page 11 inches high at 6 lines per inch (66 lines in all).
Define vertical margin limits for the page. . .	VMRG = (dt , db) , where dt is the top margin line and db is the bottom margin line. Neither dt nor db can be less than 1 or greater than HEIGHT, and dt cannot be greater than db. Output is allowed in the dt, db lines, and anywhere in between.	dt = 1 and db = HEIGHT. Enclosing parenthese may be omitted when only dt is coded. When only db is coded, it must be preceded by a comma, and the parentheses are required.
Save the location of the page so that you can make backward references to it. . .	SAVELOC = YES, if you want to save the location for the entire form, or SAVELOC = d, where d is the number of backward references you will make to this statement.	SAVELOC = NO, which does not save the statement location. *Note:* If SAVELOC is coded YES or d, a name entry is required in the FDPAGE statement.

Figure 10. Coding the FDPAGE Macro Instruction

HEIGHT = {66 / d}

The HEIGHT operand specifies the number of physically usable line positions on the current page. You can specify the value of d from 1 to 16383. If you do not specify HEIGHT in an FDPAGE macro, the default value is that coded in FDFORM, if one is. If you do not specify HEIGHT in either of these macros, a default value of 66 is used, corresponding to a page height of 11 inches and a vertical spacing of 6 lines per inch (66 lines in all).

The HEIGHT operand provides the 3735 with some of the information it needs to properly advance forms in the Selectric ® printer. Thus, if you are not using 11-inch high forms, you should code this operand with a value that describes the height of the forms you are using.

VMRG = ([{1 / dt}][, {height / db}])

The VMRG operand delimits the range of contiguous line positions on the page in which output is allowed. The first suboperand (dt) specifies the top vertical margin line, and the second suboperand (db) specifies the bottom vertical margin line. Output is allowed in the specified lines, and in all lines between them. The minimum value that you can specify in either of these positional suboperands is 1, the maximum is the current value of HEIGHT, and dt may not be greater than db. If you do not supply VMRG values in an FDPAGE macro, the default values are those coded in FDFORM, if any are. If you do not supply VMRG values in either of these macros, default values of 1 and HEIGHT are used (that is, output is allowed on the entire page as defined by HEIGHT). If a db value coded in an FDFORM macro statement is greater than the current value of HEIGHT, VMRG values of 1 and HEIGHT are used.

Note: If only the dt suboperand is coded, the enclosing parenthese may be omitted. If only the db suboperand is coded, the enclosing parentheses are *required*, and the db specification must be preceded by a comma to indicate the absence of dt.

Many forms do not need to use all the available output space on a page. For example, an invoice may not require data entry before line 15 because a preprinted company name and address occupies the space from line 1 to line 14. In such cases, you can use the VMRG operand to establish vertical-margin limits on the page. To continue with the preprinted invoice example, you can set a top margin at line 15, and allow the bottom margin to default to the page height, by coding:

VMRG = 15

Such coding enables the FD macros to cross-check the line numbers you will code in FDLINE macro statements. In addition, you may find that your FDPs are easier to read and interpret when one or both of the vertical margins are explicitly specified.

SAVELOC = { NO / YES / d }

A backward reference to a macro statement involves a branch (transfer of control) to it. For a branch of this type to be successful, the location of the branch target must be known when the branch instruction is encountered. During an FDP assembly, macro statement locations are not usually saved. Thus, when you know that you are going to make a backward reference to a particular macro statement, you need to code SAVELOC = YES or SAVELOC = d (where d is the number of references you will make to the statement) in the macro statement that is the branch target. (Forward references—such as those made by the CYCLE operand of FDLINE, FDFIELD, or FDCTRL, and by the GOTO operand of FDCTRL—are always retained until they can be resolved.)

The default action (NO) is not to save the statement location. Coding SAVELOC = YES causes the FD macros to save the location for the duration of the current FDP. Coding SAVELOC = d causes the FD macros to save the location only until d backward references to the location have been resolved, at which time the location is discarded.

You can specify from 1 to 255 backward references to this macro location. If the specified value of d is greater than 225, the FD macros supply the value YES instead, causing the location to be saved for the entire FD macro assembly. For efficiency, as few locations as possible should be saved concurrently. SAVELOC may not be coded to imply reverse form movement. Backward branching is permitted in the FDP logic only– not in the 3735 movement. An example of the use of SAVELOC is shown in the discussion of the FDCTRL macro instruction.

Note: The SAVELOC operand is not promotable, but it may be coded in any FD macro except FDFORM and FDEND.

The following are some examples of correctly coded FDPAGE macro statements:

```
         FDPAGE 5,HEIGHT=44,VMRG=21
PAGE2    FDPAGE 2,HEIGHT=68,VMRG=(5,63),SAVELOC=YES
NONUM    FDPAGE HEIGHT=75,SAVELOC=3
LONGPAGE FDPAGE 10,HEIGHT=300,VMRG=(,278)
```

FDLINE Macro Instruction

The FDLINE macro instruction marks the beginning of each line description within a page. If an FDPAGE macro is coded, FDLINE must be coded at least once in the form description (following the FDPAGE). An FDLINE macro can be coded to:

- Name the line (symbol).
- Specify a line number within a page (linenum) or skip lines in a summary block (SKIP). (Summary blocks are described in connection with the CYCLE operand.)
- Define the horizontal output space for the line (WIDTH).
- Specify the setting of the horizontal-margin checking limits (HMRG).
- Provide for the repeated execution of sequences of macro statements (CYCLE).
- Provide for backward references to the macro statement (SAVELOC).

Figure 11 shows the format of the FDLINE macro instruction.

Name	Operation	Operands
[symbol]	FDLINE	[{linenum / SKIP (d)}] [, WIDTH = {85 / d}] [, HMRG = ([{mrgstop + 1 / dl}] [, {width / dr}])] [, CYCLE = ([d] [, limit] [, target])] [, SAVELOC = {NO / YES / d}]

Figure 11. Format of the FDLINE Macro Instruction

symbol

The name entry (symbol) specifies the name of the line, if coded. A name entry is required if SAVELOC is coded, or if the macro is an explicit CYCLE limit or target, or a GOTO target (that is, if the macro is referred to elsewhere).

When used to describe a line on an 8.5-inch wide page, an FDLINE macro may be coded as simply as:

FDLINE

In this case, default values provide a line number one greater than the previous line number (beginning with the top vertical margin (VMRG) value), a width of 85 characters, horizontal margin checking limits of MRGSTOP + 1 and 85, and no cyclic processing or retention of the macro statement location.

Figure 12 is a set of charts that will help you decide how to code the FDLINE operands. A detailed discussion of each operand follows the chart in which the operand is introduced.

linenum

The linenum operand marks the beginning of each line within a page. This positional operand may be specified with values from 1 to 16383, or as SKIP (d), as discussed later. Coding this operand does not physically place a number on the form at the 3735. You can perform this function, if desired, by coding an appropriate FDFIELD macro instruction somewhere in the line.

The entire form may consist of no more than 16,383 lines, which can be distributed throughout the necessary pages in any manner suitable to the needs of the application. Thus, even though you could define 16,383 pages, they would have to be one-line pages. You can define as many as 248 complete 66-line pages (a total of 16,368 lines).

If you do not specify a line number in the first FDLINE macro of a page description, a default value equal to the VMRG-determined top margin is used. (See the description of the VMRG operand in the FDPAGE discussion.) If you do not specify a line number in a subsequent FDLINE macro statement in the same page description, a default value one greater than the previous line number is used.

Line numbers usually form an increasing series within a single page description, but they need not be consecutive. In fact, when more than one path through the lines of a given page is possible, you can often obtain more efficient FDP processing by coding the lines out of normal sequence.

For example, if one processing sequence uses lines 1, 2, 4, 6, and 8, while another processing sequence uses lines 1, 3, 5, 7, and 8, then the line coding sequences 1, 2, 4, 6,

If you want to. . .	Code. . .	Unless you want to use the default value of. . .
Specify a line number within a page. . .	linenum as a number from 1 to 16383.	The VMRG-determined top margin for the first FDLINE on a page, and the previous line number + 1 for all other lines.
Skip one or more lines at the end of a CYCLE to or within a summary block. . .	SKIP (d), where d is a number from 0 to 16382. See the CYCLE operand description following the chart for more details.	No skipping to or within a summary block.
Define the width of the line. . .	WIDTH = d, where d is a number from 1 to 130 that specifies the number of character positions on the line.	WIDTH = 85, which defines a line 8.5 inches long at 10 character positions per inch (85 positions in all).
Define horizontal margin limits for the line. . .	HMRG = (dl ,dr), where dl is the left margin and dr is the right margin. Neither dl nor dr can be less than MRGSTOP + 1 (from FDFORM) or larger than WIDTH, and dl can not be larger than dr. Output is allowed in the dl , dr positions and all positions between.	dl = MRGSTOP + 1 (from FDFORM), and dr = WIDTH. Enclosing parentheses may be omitted when only dl is coded. When only dr is coded, it must be preceded by a comma, and parentheses are required.
Repeat the execution of a group of macro statements. . .	CYCLE = (d , limit , target), where: • d is a number from 1 to 16383 that specifies tha maximum number of times the cycle can operate. • Limit is the name entry of the last macro in the repeated group. • target is the name entry of the macro to be executed when the cycle is finished.	No repeated execution of macro statements. When CYCLE is coded, at least one of the suboperands should be coded. The default values are described following the chart in the CYCLE operand discussion.

Figure 12. Coding the FDLINE Macro Instruction (Part 1 of 2)

and 3, 5, 7, 8 require only two instances of nonsequential FDP processing. The conventional sequence 1, 2, . . . , 7, 8, on the other hand, requires six. The coding of such processing sequences might be as follows:

```
FDLINE   1
FDCTRL   IF = IND (10) , GOTO = THREE
FDLINE   2
FDLINE   4
FDLINE   6
FDCTRL   GOTO = EIGHT                BRANCH AROUND 3-7
```

```
THREE   FDLINE  3
        FDLINE  5
        FDLINE  7
EIGHT   FDLINE  8
```

In another example, suppose that you want to describe two ways of entering a customer number onto a form, one using the 5496 card reader, and the other using the Selectric ® keyboard. You could, in this case, define two lines that differ only in their data source specifications, and control which line is used by setting and testing an indicator based on an operator-entered character. (Data sources are described in the discussion of the FDFIELD macro instruction.) If the output space for the customer number is in line 10, positions 26-35, and you are going to use indicator 1, the coded macro statements might appear as follows (the indicator should be set before entering this code segment):

```
KCUSTO    FDLINE   10
          FDCTRL   IF = IND (1) ,GOTO = RCUSTNO
          FDFIELD  26, 35, SOURCE = KBD
          FDCTRL   GOTO = NEXT
RCUSTNO   FDFIELD  26, 35, SOURCE = (RDR,1,10)
NEXT      [ macros for further processing ]
```

Still another example of this type of coding is shown in the sample program in Appendix C.

SKIP (d)

The SKIP (d) specification causes the 3735 to skip d form lines before processing the line, as discussed more fully under "target" in the description of the CYCLE operand. Values of d may range from 0 to 16382.

WIDTH = { 85 / d }

The WIDTH operand specifies the number of physically usable character positions on the current line. You can specify the value of d from 1 to 130. If you do not specify WIDTH in an FDLINE macro, the default value is that coded in the previous FDPAGE macro, if one is, or else that coded in FDFORM, if one is. If you do not specify WIDTH in any of these macros, a default value of 85 is used, corresponding to a line length of 8.5 inches and a horizontal spacing of 10 character positions per inch (85 positions in all).

Note: If a value of MRGSTOP or HTAB coded in FDFORM is greater than 85, then the WIDTH default is set at 130 instead of 85.

The WIDTH operand provides the 3735 with some of the information it needs to properly position the Selectric type element for printing on the form. Thus, if you are not using 8.5-inch wide forms, you should code this operand with a value that describes the width of the forms you are using.

HMRG = ([{ mrgstop + 1 / d1 }] [, { width / dr }])

The HMRG operand delimits the range of contiguous character positions on the line in which output is allowed. The first suboperand (d1) specifies the left margin character, and the second suboperand (dr) specifies the right margin character. Output is allowed in the specified character positions, and in all positions between them. The minimum value that you can specify in either suboperand is the value of MRGSTOP + 1 (see the description of the MRGSTOP operand in the FDFORM discussion), the maximum is the current value of WIDTH, and d1 may not be greater than dr. If you do not supply HMRG values in the FDLINE macro, the default values are those coded in the including FDPAGE macro, if any are, or else those coded in FDFORM, if any are. If you do not supply HMRG values in any of these macros, default values of MRGSTOP + 1 and WIDTH are assumed (that is, output is allowed on the entire line as defined by MRGSTOP + 1 and WIDTH).

If a d1 value coded in a higher-level macro statement is incompatible with the value of MRGSTOP, a default value of MRGSTOP + 1 is used. If the value of a dr coded in a higher-level macro statement is incompatible with the current value of WIDTH, a default value of WIDTH is used. In each case, the incompatible values are ignored.

Note: If only the d1 suboperand is coded, the enclosing parentheses may be omitted. If only the dr suboperand is coded, the enclosing parentheses are ***required,*** and the dr specification must be preceded by a comma to indicate the absence of d1.

Many forms do not need to use all the available output space on a line. In fact, most preprinted forms provide for blank margins on each side of the line. In such cases, you can use the HMRG operand to establish horizontal-margin limits on the line. If, for example, the form you are describing has 10-character margins on each side of an 85-character line, you can set the HMRG values at 11 and 75 by coding:

HMRG = (11,75)

Such coding enables the FD macros to cross-check the document field boundary positions you will code in FDFIELD macros. In addition, you may find that your FDPs are easier to read and interpret when one or both of the horizontal margins are explicitly specified.

CYCLE = ([d] [,limit] [,target])

The CYCLE operand specifies a cycle count (d), delimits a group of sequential macro statements that are to be processed repeatedly as a unit (limit), and specifies where sequential processing is to resume after the cyclic processing ends (target).

Note: The CYCLE operand is not promotable, but it can be coded in FDLINE, FDFIELD, or FDCTRL.

If you describe a form, such as an invoice, that may have several lines of similar data, you can use the CYCLE operand to save coding the same line format repeatedly. The number of lines may be variable, up to the maximum specified by the cycle count (d). All you need to to is code a single CYCLE operand that specifies:

1. The maximum number of times you want the line to repeat (d) – default is 1.
2. The name entry of the last macro statement in the repeated group (limit) – default is the current macro statement.
3. The name entry of the macro that is to be processed when cyclic processing is completed (target) – default is the next sequential macro statement after the limit.

Specification of each suboperand is optional, but if CYCLE is coded, then at least one suboperand should be coded.

d

Specifies the maximum number of times that the group of macro statements is to be processed. If no value is specified for d, a value of 1 is assumed. The value of d may range from 1 to 16383, but is limited to the largest value that cannot cause processing of the resulting last line to pass beyond the VMRG-determined bottom margin of the current page. (See the description of the VMRG operand in the FDPAGE discussion.) To circumvent this restriction, you can define one large page that, when printed at the 3735, has the appearance of several pages.

limit

Is the name of the subsequent macro statement that is the last of the consecutive macro statements that describe the repeated group. The macro statements in such a group frequently describe an integral number of whole lines. When processing reaches the macro statement named by limit, control is returned to the current FDLINE macro statement (the one containing the CYCLE operand), unless this is the last time through the group, as determined by the cycle count (d) or by operator intervention. When processing reaches the limit statement during the last processing cycle, control is passed to the macro statement named by target. If the limit suboperand is not coded, the current macro statement (the one issuing the CYCLE) is assumed to be the limit.

You may not code an FDPAGE macro between the current FDLINE macro and the limit statement, nor may you code another macro statement that also specifies a CYCLE operand. (That is, you may not perform cyclic processing from one page to the next, and you may not nest CYCLE operands.)

The macro statement named by limit must be an FDLINE, FDFIELD, or FDCTRL macro.

Note: An unconditional GOTO should not be specified in an FDCTRL macro statement that is a CYCLE limit.

target

Is the name of the subsequent macro statement that is to be processed when cyclic processing is either completed (the cycle count, d, is reached) or stopped by the 3735 operator. The named macro statement may be of any type except FDFORM. It may not be within the range of any cyclic repetition, except that a macro statement that is a CYCLE target may itself contain a CYCLE operand (unless the target is in a summary block).

If you do not code the target suboperand, the macro statement immediately following the limit statement is assumed to be the target. You can thus omit the target suboperand when the target statement immediately follows the repeated group of macros. Examples of cyclic processing are found following the next chart, in Figure 21, and in the sample program in Appendix C.

Note: At the end of a cyclic repetition, the Selectric print element is positioned at the column where the CYCLE started, in anticipation of another repetition. If the CYCLE starting column is less than or equal to the CYCLE ending column, then each repetition of the CYCLE must start on a new line. Thus, when a CYCLE begins in a column less than or equal to the column in which it ends, the Selectric print element is positioned to a new line when the CYCLE ends.

If you want to . . .	Code. . .	Unless you want to use the default value of . . .
Define a summary block to be executed at the end of the CYCLE. . .	SKIP(d) as the linenum operand of a target FDLINE macro statement (including conditional GOTO targets).	No summary block. See the discussion following the chart for more details.
Save the location of the line so that you can make backward reference to it. . .	SAVELOC = YES, if you want to save the location for the entire form, or SAVELOC = d, where d is the number of backward references you will make to this statement.	SAVELOC = NO, which does not save the statement location. *Note:* If SAVELOC is coded YES or d, a name entry is required in the FDLINE statement.

Figure 12. Coding the FDLINE Macro Instruction (Part 2 of 2)

If the CYCLE target is an FDLINE statement with its linenum operand coded SKIP (d), then d lines are skipped upon exit from the cycle before the target line is processed. Consecutive following line descriptions whose FDLINE statements also specify SKIP (d) are processed in immediate succession. This procedure allows you to define a *summary block,* which follows the last repeated line a fixed number of form lines away, and whose lines are processed with fixed spacing. Thus, a summary block is like a fixed-format "floating" block of data that follows the last cyclically repeated line exactly d lines away, regardless of whether cyclic processing stops after one line or after the maximum number specified in the CYCLE operand.

You can exit to different summary blocks from the same cycle by using the CYCLE target plus one or more conditional FDCTRL GOTO targets. Each summary block in such a collection (except, optionally, the last) must be closed with an unconditional FDCTRL GOTO statement that branches outside of the summary block region of the FDP. The contiguous summary blocks are automatically terminated by the appearance of an FDPAGE statement, an FDEND statement, or an FDLINE statement not specifying SKIP (d).

Summary blocks are logically connected to the preceding cycle. Thus, the number of lines of output that a summary block can produce, added to the number of lines that the preceding cycle can produce, must not violate the page's bottom margin.

For example, suppose you want a line that says "THANK YOU" to follow the last line of a 20-time CYCLE exactly two lines away, even if the CYCLE executes fewer than the permitted maximum of 20 times, then you want to go to the end of the form and print some control information. On a 35-line form, the code segment might look like this:

```
         FDLINE   11,CYCLE = (20,LAST, THANKS)
LAST     FDFIELD  10,60, SOURCE = KBD, SINK = (PRT,TMT)
*                   BEGIN SUMMARY BLOCK
THANKS   FDLINE   SKIP (2)
         FDFIELD  10,60, SOURCE = 'THANK YOU' , JUSTIFY = C,SINK = PRT
*                   END SUMMARY BLOCK
         FDLINE   35
         FDFIELD  10,12, SOURCE = FID,SINK = PRT
         FDFIELD  13,13, SOURCE = '/',SINK = PRT
         FDFIELD  14,16, SOURCE = RSN, SINK = (PRT,TMT)
         FDEND
```

The SOURCE and SINK operands illustrated in this example are described in the discussion of the FDFIELD macro instruction.

The only branching allowed to the lines of a summary block is either from within the preceding CYCLE or from an FDCTRL GOTO within the same summary block or a preceding summary block.

SAVELOC = { NO / YES / d }

The SAVELOC operand directs the FD macros to save the location of the current macro statement within the form description, for the purpose of resolving backward references to the statement. The coding of this operand is explained in the discussion of the FDPAGE macro instruction.

Note: Neither CYCLE nor SAVELOC may be coded within a summary block.

The following are some examples of correctly coded FDLINE macro statements:

```
         FDLINE   1,WIDTH = 125,HMRG = (11,115)
CYCLE1   FDLINE   CYCLE = (10, STOPMAC,GOMAC) ,SAVELOC = YES
FIRSTMAC FDLINE   35, CYCLE = (25, LASTMAC)
```

FDFIELD Macro Instruction

The FDFIELD macro instruction describes a data field within a line. You must code one FDFIELD macro for each data field you want to define. An FDFIELD macro may not be coded before FDLINE has been coded. An FDFIELD macro can be coded to:

- Name the field (symbol).
- Locate the field within the line (dl, dr).
- Perform arithmetic operations with numeric field data and one or more counters (CTR).
- Set or reset indicators, or conditionally branch (when the File Storage capability is present), based on the field data (IND).
- Provide for the repeated execution of sequences of macro statements (CYCLE).
- Provide for backward references to the macro statement (SAVELOC).
- Define the origin of the data (SOURCE).
- Validate the character-set membership of each incoming character (KIND).
- Perform self-checking of numeric data (SELFCHK).
- Validate the number of characters entered by the 3735 operator (COUNT).
- Compare the entered data with one or more fixed comparands (COMPARE).
- Describe the destination of the field data (SINK).
- Indicate what editing functions are to be performed on the data (JUSTIFY, FILL, UL, and PICTURE).
- Flag numeric data as belonging to a particular data batch of a particular FDP (BATCH).

Figure 13 shows the format of the FDFIELD macro instruction.

Name	Operation	Operands
[symbol]	FDFIELD	[{ hmrgdl / prevdr + 1 / dl / DUMMY }] [, { complng / LNG (d) / dr }] [,CTR = ((d, op [,FIELD]) [, (d, op [,FIELD])] ...)] [,IND = (({ d / targ } ,logexp) [, ({ d / targ } ,logexp)] ...)] [, CYCLE = ([d] [, limit] [, target])] [, SAVELOC = { NO / YES / d }] [, SOURCE = (origin[, qualifier]...)] [, KIND = { U / A / N / AN / K }] [, SELFCHK = { NO / ({ 10 / 11 } [, GENONLY]) }] [, COUNT = { ([MIN ,] { 1 / d1 } , [MAX ,] { compmax / d2 }) / d }] [, COMPARE = ([FIELD ,] comparopr , comparand [, { AND / OR } ,[FIELD ,] comparopr , comparand]...)] [, SINK = ([(destination[, qualifier] ...)] [,[(destination[,qualifier] ...)]]...)] [, JUSTIFY = ([justcode] [, [justcode]] ...)] [, FILL = (['char'] [, ['char']] ...)] [, UL = ([{ NO / YES }] [,[{ NO / YES }]] ...)] [, PICTURE = (['picturespec'] [,['picturespec']] ...)] [, BATCH = d]

Figure 13. Format of the FDFIELD Macro Instruction

symbol

The name entry (symbol) specifies the name of the field, if coded. A name entry is required if SAVELOC is coded, or if the macro is an explicit CYCLE limit or target, or a branch target (that is, if the macro is referred to elsewhere).

Figure 14 is a set of charts that will help you decide how to code the FDFIELD operands. A detailed discussion of each operand follows the chart in which the operand is introduced.

[{ **hmrgd1** / **prevdr + 1** / **d1** / **DUMMY** }][, { **complng** / **LNG (d)** / **dr** }]

The field location operands specify the continuous character positions within the current line in which output for this field is allowed. These positional operands usually specify the leftmost character position (d1) and either the field length, – LNG (d), with d being the number of character positions – or the rightmost character position – dr. The value of d1 cannot be less than the current HMRG-determined left margin, and the value of dr cannot be greater than the current HMRG-determined right margin or less than d1. (See the description of the HMRG operand in the FDLINE discussion.) Similar restrictions apply to LNG (d), since dr = d1 + d-1.

You should code the field location operands only for fields that are directed to the Selectric ® printer. If the field boundaries are coded for fields directed to other sinks, the Selectric printer moves as though some data was directed to it. Such unnecessary movement might confuse the 3735 operator, and result in inefficient FDP execution at the 3735.

For example, to define a 10-character field that begins in position 11 and ends in position 20, you can code:

```
FDFIELD 11,20 or FDFIELD 11,LNG (10)
```

If the left boundary operand is coded DUMMY, the FD macros create a null (dummy) field. Such coding is permitted only in an FDFIELD macro that is a CYCLE limit. The second operand may then be coded dr or omitted. If it is coded dr, the next available logical position within the current line is moved rightward to the value of dr + 1. If it is omitted, the logical position within the form is unchanged. Coding DUMMY,LNG(d) is not permitted. Dummy is normally used to define or logically extend a place-holding field, in order to force the first Selectric output of successive cycles to be on a new line. For example, assume that you have a cyclically repeating group of macros that begin in position 23 of one line and end in position 22 of the next line. If you do not want to begin the next repetition on the same print line, you can code a dummy field as the limit of your cyclic group to logically extend the last repeated field, as follows:

```
LIMIT   FDFIELD DUMMY ,23
```

This overlap forces the next cyclic repetition to begin on a new line.

If the left boundary operand is not coded, the default value is the HMRG-determined left margin for the first field defined within the current line (hmrgd1), and the previous dr plus one for all other fields within the line (prevdr + 1). If the right boundary operand is not coded, the default value (complng) is obtained (if possible) from the current values of other FDFIELD operands, such as SOURCE and COUNT.

CTR = ((d,op[,FIELD])
[,(d,op[,FIELD])] . . .)

The CTR operand provides for the performance of arithmetic operations (op) on one or more 10-digit counters (d), with the completed numeric data for the current field as the operand. The counter arithmetic operations are performed in all operating modes. The operations are performed in the order in which they are coded, and the result of each operation is stored in the counter that is operated on. Each operation may be symbolized by the assignment expression:

If you want to. . .	Code. . .	Unless you want to use the default value of. . .
Specify the location of the field within the line. . . (The use of DUMMY is discussed in the operand description following the chart.)	dl , dr, where dl is the starting (left) position, and dr is the ending (right) position or. . . dl , LNG (d), where dl is the starting (left) position, and d is the length of the field. The field boundries must not violate the HMRG-determined margins, nor can dr be less than dl .	dl = the dl value of HMRG (from FDLINE) for the first field on a line, and the previous FDFIELD dr + 1 for all other fields on a line. dr = a value obtained from other FDFIELD operands, such as SOURCE and COUNT. For clarity, it is suggested that both dl , dr or dl , LNG (d) be coded for all data fields.
Use the field data in arithmetic operations with a counter. . . such as. . . Add the field data to the value in CTR 1. . . Subtract the field data from the value in CTR 2. . . Multiply the value in CTR 3 by the field data. . . Divide the value in CTR 4 by the field data (ignore remainder), then divide-and-round the value in CTR 5 by the field data (use remainder to round off quotient). . .	CTR = (d, operation), where d is the counter number (from 1 to 21), the operation is either ADD, SUB, MPY, DIV, or DVR, and the field data is the other arithmetic operand. Coding the word FIELD as shown in the format illustration (Figure 11), is optional. CTR = (1, ADD) CTR = (2, SUB) CTR = (3, MPY) CTR = (4, DIV), (5, DVR)	No arithmetic operations with the data from this field. Operations may be performed on more than one counter, as described following the chart. In all cases, the result of the arithmetic operation replaces the contents of the counter.

Figure 14. Coding the FDFIELD Macro Instruction (Part 1 of 7)

(counter d) < – (counter d) (operator) (field data)

The values of the counter number (d) may range from 1 to 21. The 3735 does not clear the indicated counters before performing the requested operations. If you want to clear a counter before starting arithmetic operations on it, you should code an FDCTRL macro statement that specifies CTR = (d,CLR). If you want to start with some specific value in a counter, you can use an FDCTRL macro that specifies CTR = (d, CLR, ADD, value) or, if the value can be obtained from the current field, the FDFIELD operand SINK = CTR (d). No overflow indication is provided if the value in a counter exceeds the capacity for the counter ($\pm$ 10^{10}-1).

The codes allowed for op in the CTR operand of the FDFIELD macro, and their meanings, are as follows (framing apostrophes are required where shown):

ADD or ' + ' The numeric value of the field data is added to the contents of the counter, and the sum replaces the contents of the counter.

SUB or ' - ' The numeric value of the field data is subtracted from the contents of the counter, and the difference replaces the contents of the counter.

MPY or ' * ' The contents of the counter are multiplied by the numeric value of the field data, and the product replaces the contents of the counter.

DIV or ' / ' The contents of the counter are divided by the numeric value of the field data, the remainder is discarded, and the quotient replaces the contents of the counter. The counter is cleared to zero if an attempt is made to divide by zero.

DVR or ' /+ ' The contents of the counter are divided by the numeric value of the field data, the remainder is used to round ("half-adjust") the quotient, and the adjusted quotient replaces the contents of the counter. The counter is cleared to zero if an attempt is made to divide by zero.

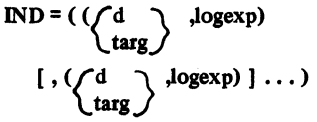
The FIELD specification is an optional encoding allowed for its value as a memory aid, but is not required.

When a CTR arithmetic operation is coded for a counter that is also named in a SINK = CTR(d) specification, the counter contains the value of the field input when the operation begins. For example, coding

FDFIELD SOURCE = KBD,SINK = CTR (1), CTR = ((1,SUB) , (2 ,SUB)

subtracts the field input from the value in counter 2 and sets the final value of counter 1 to zero.

Note: The CTR operand is not promotable. When only one counter is to be operated on by the field data, the inner grouping parentheses may be omitted. When the CTR operand is coded, the KIND option is forced to numeric, and any coding of KIND is ignored. Counter operations are not permitted with SOURCE = 'string' data.



The IND operand provides for the testing of logical conditions related to the data input for the current field (logexp), and either for the modification of program logic indicators (d) to the truth values of a logical expression, or for a conditional branch to be taken to the named macro statement (targ) when the File Storage capability is present. The possible truth values are TRUE = ON = '1' and FALSE = OFF = '0'. The affected indicators may be tested later by an FDCTRL statement. The 3735 resets all indicators to '0' before beginning to process each copy of a form. The values of the indicator number (d) may range from 1 to 84.

If the name of a target macro statement (targ) is used instead of an indicator number (d), the branch to the target occurs when the logical expression is evaluated as TRUE. If the branch occurs, certain FDFIELD operations are not performed; they are to place data in a buffered sink, print or transmit edited data, print or transmit AFTER entry, generate SELFCHK values, and perform any other IND operations following the one that caused the branch. All other specified FDFIELD operations, including CTR and SINK = CTR operations, are performed unconditionally.

The format of logexp (logical expression) is as follows:

[FIELD ,] comparopr , comparand [, { AND / OR } , [FIELD ,] comparopr , comparand]

Each logical test consists of comparing the data input for the current field to some fixed data (comparand) as specified by a comparison operator (comparopr). When the comparison has been made, the specified indicator is set to '1' if the result is true, or '0' if the result is false. The FIELD specification is an optional encoding allowed for its value as a memory aid, but is not required.

All comparisons except numeric-only are made against character strings. If one string is shorter than the other, it is extended to the right with blanks until its length equals that of the longer string. All string comparisons are made with the collating sequence implicit in the transmission code of the particular 3735 (EBCDIC or ASCII).

The allowable comparison operators (comparopr) are as follows (framing apostrophes are required where shown):

GT or ' > ' Greater than
NG or '¬> ' Not greater than
EQ or ' = ' Equal to

NE or '¬ =' Not equal to
LT or '< ' Less than
NL or '¬< ' Not less than
GE or '> =' Greater than or equal to
LE or '< =' Less than or equal to

The format of comparand is either [+] decdigits or -decdigits for numeric comparison, or 'string' for character comparison. A + is assumed if no sign is specified. All comparands must be of the same type and appropriate to the data being processed. (You cannot, for example, code a character comparand and a numeric comparand in the same IND operand.)

If you want to. . .	Code. . .	Unless you want to use the default value of. . .
Perform logical tests on the field data and set indicators that describe the results of the tests. . . such as. . . If the field data is the character A, set indicator 1 on. . . If the field data is a number between 10 and 20, or the number 99, set indicator 2 on. . .	IND = (d, logical-expression), which sets indicator d ON or OFF, depending on the results of the testing. The value of d can range from 1 to 84. IND = (target, logical-expression), which causes the branch to be taken when the evaluation of the logical expression yields a TRUE result. IND = (1, EQ, 'A') IND = (2, GE, 10, AND, LE, 20, OR, EQ, 99) (As shown above, tests may be combined by the operators AND and OR.)	No logical tests and no setting of indicators. The comparison operators are: GT greater than NG not greater than EQ equal to NE not equal to LT less than NL not less than GE greater than or equal to LE less than or equal to or their symbolic equivalents.
Repeat the execution of a group of macro statements. . . and. . . Define a summary block to be executed at the end of the cycle. . .	CYCLE = (d, limit, target), where: • d is a number from 1 to 16383 that specifies the maximum number of times the cycle is to operate. • limit is the name entry of the last macro in the repeated group. • target is the name entry of the macro to be executed when the cycle is finished. SKIP (d) as the linenum operand of a target FDLINE macro statement.	No repeated execution of macro statements. When CYCLE is coded, at least one of the suboperands should be coded. The default values are described following Figure 12 (Part 1). No summary block. See the discussion following Figure 12 (Part 2) for more details.

Figure 14. Coding the FDFIELD Macro Instruction (Part 2 of 7)

The number of characters used to encode comparands, exclusive of the framing apostrophes, may range from 1 through a maximum number that depends on the Assembler used and on the number of comparisons coded. In no case, however, may the net number of characters per comparand exceed 127.

The results of the individual tests can be combined with the logical operators AND (or '&&') and OR (or '|'). Evaluation and combination proceeds from left to right through the operand terms, with AND taking precedence over OR. Grouping to force OR to take precedence over AND is not permitted.

The 3735 program logic indicators enable you to design an FDP that can make delayed processing decisions while it is being used at a 3735. For example, suppose that your FDP must make a decision at some point that depends on operator input in the current field. One way to retain such information for later use is to set one or more indicators that describe the current activity. Thus, if the operator could enter an "A", "E" or "C" from the keyboard, you could set indicators to describe the entered data in this way:

```
IND = ( (1, EQ , 'A') , (2 , EQ , 'B') , (3 , EQ , 'C') )
```

The indicators can then be tested later by an FDCTRL macro statement to determine which character was entered.

On the other hand, if a particular processing path is to be taken immediately when a number falling within certain limits (say, from 2 to 10) is entered, you can cause a branch to be taken to some other macro statement. For example:

```
IND = (TWOTEN, GE, 2, AND, LE, 10)
```

When the number entered falls within these limits, the branch to the macro statement named TWOTEN is taken. Otherwise, no branch is taken.

Note: The IND operand is not promotable. The File Storage capability must be present for IND branching to be allowed. When only one indicator is to be modified, or only one target is specified, the inner grouping parentheses may be omitted. Indicator operations are not permitted with SOURCE = 'string' or SOURCE = FID data.

CYCLE = ([d] [, limit] [,target])

The CYCLE operand specifies a cycle count (d), delimits a group of sequential macro statements that are to be processed repeatedly as a unit (limit), and specifies where sequential processing is to resume after the cyclic processing ends (target). An FDFIELD CYCLE specification can be used to advantage when you want to repeat only part of a line. Such coding can result in more efficient mechanical motion on the Selectric ® printer. The coding of this operand is explained in the discussion of the FDLINE macro instruction. Note, however, that there is no facility uniquely associated with FDFIELD which is analogous to the summary block facility of FDLINE, and that cycling cannot be used to cause field repetition across a line.

If you want to. . .	Code. . .	Unless you want to use the default value of. . .
Save the location of the field so that you can make backward references to it. . .	SAVELOC = YES, if you want to save the entire form, or SAVELOC = d, where d is the number of backward references you will make to this statement.	SAVELOC = NO, which does not save the statement location. *Note:* If SAVELOC is coded YES or d, a name entry is required in the FDFIELD statement.

Figure 14. Coding the FDFIELD Macro Instruction (Part 3 of 7)

SAVELOC = { NO / YES / d }

The SAVELOC operand directs the FD macros to save the location of the current macro statement within the FDP, for the purpose of resolving backward references to the statement. The coding of this operand is explained in the discussion of the FDPAGE macro instruction. An example of its use is found in the discussion of the FDCTRL macro instruction.

Source Keyword Operands

SOURCE = (origin[,qualifier] . . .)

The Source operand indicates, for each document field, the origin of the data for that field. If this operand is not coded, a default value of SOURCE = KBD is used. Each field may have only one data source that cannot exceed 127 characters in length. The allowable values of origin and qualifier are described below.

Note: When an unqualified origin is specified, the enclosing parentheses may be omitted.

Non-Buffered Sources: The non-buffered sources are those origins of data for the field which either do not reside in the 3735 or are of fixed length and fixed content-type. Data is obtained from the specified source (except for KBD) in all 3735 operating modes (enter form, error correct, and playback). When SOURCE = KBD is coded, data is obtained from the keyboard in enter form and error correct modes, and from the 3735 disk in playback mode.

SOURCE = (KBD[,OPTIONAL] [,NUMPAD] [,AUTOEOF])

The SOURCE = KBD (keyboard) specification directs the 3735 to get the data for the current field from its Selectric keyboard (KBD).

The qualifying suboperand OPTIONAL specifies that the 3735 operator is not obliged to enter data. If this qualifier is not coded, and if the operator enters no data, the 3735 stops, turns on an error indicator, and gives the operator another opportunity to enter the required data.

The qualifying suboperand NUMPAD (numeric pad) directs the 3735 to treat as numeric characters any input from an array of ten of its keys arranged like those of an adding machine. When SOURCE = (KBD,NUMPAD) is coded, the operator may key non-numeric characters outside the ten-key area unless KIND = N is coded.

The qualifying suboperand AUTOEOF (automatic end-of-field) specifies that the 3735 operator is not obliged to strike the "entry completed" key when the full number of characters allowed for the current field are entered.

Keyboard qualifiers may be coded in any order, but only once each. If you promote a qualified SOURCE = KBD operand to a higher-authority macro instruction, then code SOURCE = KBD in some lower-authority macro, you should be aware that any keyboard qualifiers coded at a particular level of authority are the only ones in effect. If no qualifiers are coded, and the macro of next-higher authority specifies SOURCE = KBD, the qualifiers at that higher level (if any) are used. If the next-higher level does not specify SOURCE = KBD, no qualifiers are used.

For 3735 forms that have many fixed-length numeric fields, a specification of SOURCE = (KBD, NUMPAD, AUTOEOF), promoted to a higher-level macro statement, should be useful in many cases. Where some other data source is desired for a particular field, the SOURCE specification may be modified in the FDFIELD macro defining that field.

If the primary data source is the Selectric ® keyboard, and the operator is not required to enter data in some fields, you should use the OPTIONAL qualifier to indicate this, by coding SOURCE = (KBD, OPTIONAL).

If you want to. . .	Code. . .	Unless you want to use the default value of. . .
Specify the origin of the data for this field. . .	SOURCE = origin, where the origin can be one of the following: • Selectric keyboard (KBD - the default) • Card reader (RDR) • Storage buffer (STG) • Inquiry buffer (INQ) • Line printer buffer (LPB) • Read/punch buffer (RPB) • Operator Identification Card Reader (IDR or CCR) • A character string ('string') • A counter (CTR (d), where d is the number of the counter) • The FDP number (FID) • The record sequence number of the current record (RSN) record (RSN) • Input/output buffer (IOB) • Index counters (X1 or X2) *Note:* Buffered sources should be filled with data by some other macro statement before being used.	SOURCE = KBD, which may be qualified as follows: OPTIONAL - the 3735 operator does not have to enter data in this field. NUMPAD - a special set of 10 adding-machine-like keys on the Selectric keyboard may be used to enter numeric data. AUTOEOF - the 3735 operator does not have to press the "entry complete" key when the full number of characters for the field have been entered. The qualifiers may be coded in any order, but only once each.
such as. . .		
The special numeric section of the Selectric keyboard. . .	SOURCE = (KBD, NUMPAD)	
Positions 1-10 of the card reader buffer (RDR). . .	SOURCE = (RDR, 1, 10) or. . . SOURCE = (RDR, 1, LNG (10))	*Note:* All requests for data from a buffer (RDR, STG, INQ, LPB, RPB, IOB, IDR, or CCR) are made this way.
The character string "End OF INVOICE". . .	SOURCE = 'END OF INVOICE'	
The contents of counter 3. . .	SOURCE = CTR (3)	
The current form description program #. . .	SOURCE = FID	
The sequence number of the current record. . .	SOURCE = RSN	

Figure 14. Coding the FDFIELD Macro Instruction (Part 4 of 7)

SOURCE = 'string'

The SOURCE = 'string' specification defines a literal character string that the 3735 is to get directly from the FDP. The string length may range from 1 to a maximum number of characters that depends on the Assembler used (not counting the framing apostrophes, which must be coded). To avoid truncation, the number of characters in the string must not exceed the capacity of any currently active sink. If the COUNT, SELFCHK, COMPARE CTR, IND, FILL, or KIND operand is coded, it is ignored.

For example, if you want to provide a character string as data for the current field, such as "thank you for your order, you can code:

```
SOURCE = 'THANK YOU FOR YOUR ORDER.'
```

In addition to the limitations of Assembler and sink capacity, the 3735 limits the character string to 127 characters (not including the framing apostrophes).

Note: SOURCE = 'string' is not promotable.

SOURCE = CTR (d)

The SOURCE = CTR (d) (counter) specification directs the 3735 to retrieve a signed (+ or -) ten-digit number from the counter specified by d. A negative value is represented in the counter by an overpunch in the units position. The overpunch is stripped off for printing (unless a signed PICTURE specification is used), but is retained when the number is stored right-justified in buffered sinks or stored on the disk for transmission to the CPU. The counter is assumed to have been loaded in a previous operation. The value of d may range from 1 to 21. If the COUNT or KIND operand is coded, it is ignored.

For example, to get the data for the current field from a 10-digit counter, such as counter 18, you can code SOURCE = CTR (18).

Although a SOURCE = CTR (d) specification retrieves a 10-digit number from the counter, you may not want to use all 10 digits in subsequent operations. Two simple ways to adjust the number of digits are:

1. Use a PICTURE specification to define a numeric string of the desired length for some SINK. (Examples are shown in the sample program in Figure 21.)
2. Store all 10 digits in the storage buffer (SINK = STG), then get the desired number of digits from the buffer with another FDFIELD macro that requests data from the desired positions of the storage buffer (SOURCE = STG).

SOURCE = {X1 / X2}

The SOURCE = X1 or SOURCE = X2 specification directs the 3735 to use as data the contents of the selected index counter (X1 or X2), which are provided with the File Storage capability. The contents of the selected index counter are then printed as three decimal digits, varying from 000 to 255.

Note: SOURCE = X1 or X2 is not promotable. The only other operands that may be coded or in effect through promotion are SINK = PRT and field boundaries that allow for at least three positions. If SINK = PRT is not coded, the macros force that specification. Any other operands are ignored.

SOURCE = FID

The SOURCE = FID (FDP identification) specification directs the 3735 to supply as data the identification of the currently active FDP, which was established at assembly time by the FDFORM operand FID = 'ddd'. Three decimal characters are supplied by the 3735. If the COUNT, IND, COMPARE, or KIND operand is coded, it is ignored.

SOURCE = RSN

The SOURCE = RSN (record sequence number) specification directs the 3735 to supply as data the sequence number of the record now being created. This number, consisting of three decimal digits, is automatically updated by the 3735 after each record is completed. If the COUNT or KIND operand is coded, it is ignored.

The SOURCE = FID and SOURCE = RSN specifications allow you to use as data the number of the FDP that is being used at the 3735 (FID) and the record sequence number of the form being created at the 3735 (RSN). The FID and RSN data should be printed somewhere on each form prepared at a 3735. Then, if corrections are required, the 3735 operator can tell immediately which FDP and record are needed. For example, the last line of the form might be coded as follows:

```
FDLINE 66,SINK = PRT
FDFIELD 1,3,SOURCE = FID
FDFIELD 4,4,SOURCE = '/'
FDFIELD 5,7,SOURCE = RSN
```

For record sequence number 023 and FDP 061, the resulting output would appear on the form as:

061/023

Note: The 3735 always inserts the FID number as the first field of a record that is to be sent to the CPU.

Buffered Sources: The buffered sources are those origins of data for the field which reside in the 3735, are of fixed length and variable content, and are byte-addressable. The coding format for all buffered sources is:

$$\text{SOURCE} = (\text{bufname}[\,,\ \left\{\begin{matrix}1\\ \text{prevd2+1}\\ \text{d1}\\ \text{Xn}\end{matrix}\right\}][\,,\ \left\{\begin{matrix}\text{complng}\\ \text{LNG (d)}\\ \text{d2}\end{matrix}\right\}])$$

where:

bufname specifies a code name for the buffer to be used as the data source.

The qualifying suboperands d1 (or Xn) and LNG(d) (or d2) specify the range of contiguous character positions within a buffer from which the 3735 is to get data. These suboperands specify either the lower-numbered character position (d1) or one of two index counters (Xn) that should be used to determine the starting position, and either the field length (LNG(d), with d being the number of character positions), or the higher-numbered character position (d2). Neither d1 nor d2 can be less than 1 or greater than a buffer-dependent value, nor can d1 exceed d2. Further, the data length may not exceed 127 (the maximum length of a 3735 field). Similar restrictions apply to LNG(d), since d2 = d1+d-1. If d1 is not coded and an index counter has not been promoted, the default value is position 1 if the present request for data movement is the first coded since the last encoding of certain buffer-dependent FDCTRL commands. Otherwise, the previous d2 value plus one is used. Note, however, that you can read the same buffer positions as many times as you wish by explicit encoding (the next d1 does not have to be greater than the previous d2).

If the starting position (d1) is omitted at the field level but the source was promoted with an index counter starting position, then the index counter is used at the field level. Otherwise, if the starting position is omitted and no index counter has been promoted, then the starting position is either 1 or the last end position plus 1.

The File Storage capability must be present in order to specify and use the index counters. When using one of the index counters (X1 or X2) to identify the starting position in the buffer, the position is offset from character position 1 of the buffer by the value stored in the index counter. When an index counter is used to locate the starting position, the second qualifier (if used) must be the data length, LNG(d). In addition, index counters may not be specified for the IDR or CCR buffer. You can place a value in an index counter by coding the FDCTRL CTR operand. (See the FDCTRL discussion for further details.) If d2 or LNG(d) is not coded, the default value (complng) is obtained (if possible) from the current values of other FDFIELD operands, such as the field length or COUNT.

Note: The d2 (or LNG (d)) specification is not promotable, since the size of a field is not significant above the field structural level.

Numeric data from buffered sources may have a leading sign (+ or -, which may be preceded by leading blanks), or may have an overpunch in the low-order digit position to represent a negative number. The overpunch is stripped off for printing (unless a signed PICTURE specification is used), but is retained when the number is stored right-justified

in buffered sinks or stored on disk for transmission to the CPU. Leading blanks are stripped off for printing.

SOURCE = RDR

The SOURCE = RDR (card reader) specification directs the terminal to get the data for the current field from an internal read-only card-image buffer that has already been filled by a READ(RDR) command (see the description of the COMMAND operand in the FDCTRL discussion). For example, to get the data for the current field from positions 81 to 96 of the card-image (RDR) buffer, you can code:

SOURCE = (RDR, 81, 96) or SOURCE = (RDR, 81, LNG (16))

Since the card image consists of exactly 96 characters, take care that d2 is not greater than 96, and observe a similar limit for LNG (d).

SOURCE = RDR cannot be coded if BUFFERS = RPB is coded in FDFORM. SOURCE = RDR and SOURCE = RPB are mutually exclusive. The RDR buffer is not saved at the beginning of each FDP, nor is it restored if the operator enters error correct mode. (During error correct mode and playback mode, all data is retrieved from the disk, since the data in the buffer may have been changed by the time a record is played back.)

The maximum values of d1 and d2 are 96 when SOURCE = RDR is coded. The default value of d1 is reset to 1 the next time an FDCTRL READ(RDR) command is encountered in the FD macro assembly.

Note: You are required to anticipate when the current card image is no longer useful and a new record from the 5496 is needed. The 3735 does not do this for you. The discussion of the FDCTRL READ (RDR) command describes the input techniques that should be used with the 5496.

SOURCE = STG

The SOURCE = STG (storage) specification directs the 3735 to get the data for the current field from an internal write/read character storage buffer that has already been filled by an FDFIELD macro that specifies SINK = STG. For example, to get the data for the current field from positions 26 to 50 of the storage buffer, you can code:

SOURCE = (STG, 26, 50) or SOURCE = (STG, 26, LNG (25))

The storage buffer (STG) is 236 bytes long, but the length of a data string requested from it may not exceed 127 characters (the maximum length of a 3735 field).

The contents of the storage buffer are saved at the beginning of each FDP (or the re-execution of a single FDP). If the record is canceled, the storage buffer is restored to its state at the beginning of the record. (During playback mode, all data is retrieved from the disk, since the data in the buffer may have been changed by the time a record is played back.)

The maximum values of d1 and d2 are 236 when SOURCE = STG is coded. The default value of d1 is reset to 1 the next time an FDCTRL CLEAR (STG) command is encountered in the FD macro assembly.

SOURCE = INQ

The SOURCE = INQ (inquiry) specification directs the 3735 to get the data for the current field from an internal write/read character buffer that has already been filled by a response transmitted from the central computer following its reception of a 3735 inquiry message. For example, to get the data for the current field from positions 4 to 35 of the inquiry buffer, you can code:

SOURCE = (INQ, 4, 35) or SOURCE = (INQ, 4, LNG (32))

The inquiry buffer (INQ) is 236 bytes long, but the length of a data string requested from it may not exceed 127 characters (the maximum length of a 3735 field).

When using the inquiry buffer for inquiry operations (instead of for temporary data storage), you should not expect to find data in the first three positions of the buffer, since these positions contain a message header. In addition, if the rest of the inquiry

block received from the CPU is less than 233 bytes long, an ETX character appears in the buffer to identify the end of the inquiry block. This ETX must not be in the buffer when the next inquiry message is sent. If the data for the next inquiry message will not overlay the ETX, you should clear the inquiry buffer before starting to place data in it. See the "System Design Considerations" section for further details on inquiry message handling.

The inquiry buffer is not saved at the beginning of each FDP, nor is it restored if the operator enters error correct mode. During error correct mode and playback mode, all data is retrieved from the disk, since the data in the buffer may have been changed by the time a record is played back.

The maximum values of d1 and d2 are 236 when SOURCE = INQ is coded. The default value of d1 is reset to 1 the next time an FDCTRL SEND or CLEAR (INQ) command is encountered in the FD macro assembly.

SOURCE = LPB

The SOURCE = LPB (line printer buffer) specification directs the 3735 to get the data for the current field from an internal write/read character buffer that has already been filled by an FDFIELD macro that specifies SINK = LPB. For example, to get the data for the current field from positions 100 to 104 of the line printer buffer, you can code:

SOURCE = (LPB, 100, 104) or SOURCE = (LPB, 100, LNG (5))

The length of a data string requested from the line printer buffer may not exceed 127 (the maximum length of a 3735 field).

During error correct mode and playback mode, all data is retrieved from the disk, since the data in the buffer may have been changed by the time a record is played back.

The maximum values of d1 and d2 are 132 when SOURCE = LPB is coded, unless BUFFERS = (LPB, ddd) was coded in FDFORM. In this case, the maximum is the value of ddd that was coded (up to 236). The default value of d1 is reset to 1 the next time a FDCTRL CLEAR (LPB) command is encountered in the FD macro assembly.

SOURCE = RPB

The SOURCE = RPB (read/punch buffer) specification directs the 3735 to get the data for the current field from an internal 192-character write/read buffer that has already been filled by an FDFIELD macro that specifies SINK = RPB. (RPB positions 1-96 can also be filled from a 5496 by an FDCTRL READ (RDR) command, since the read/punch buffer occupies the same internal storage as the 96-character RDR buffer and the 96-character PCH buffer combined.) SOURCE = RPB cannot be coded unless BUFFERS = RPB was coded in the FDFORM macro. SOURCE = RPB and SOURCE = RDR are mutually exclusive. For example, to get the data for the current field from positions 101 to 125 of the read/punch buffer, you can code:

SOURCE = (RPB, 101, 125) or SOURCE = (RPB, 101, LNG (25))

The length of a data string requested from the read/punch buffer may not exceed 127 (the maximum length of a 3735 field).

During error correct mode and playback mode, all data is retrieved from the disk, since the data in the buffer may have been changed by the time a record is played back.

The maximum values of d1 and d2 are 192 when SOURCE = RPB is coded. The default value of d1 is reset to 1 the next time an FDCTRL READ (RDR) or CLEAR (RPB) command is encountered in the FD macro assembly.

SOURCE = IOB

The SOURCE = IOB (input/output buffer) specification (for the File Storage capability only) directs the 3735 to get the data for the current field from an internal 236-character write/read buffer that has already been filled by an FDFIELD macro that specifies SINK = IOB or an FDCTRL macro that specifies COMMAND = READ (file qualifier).

For example, to get the data for the current field from positions 17 through 25 of the IOB, you could code:

SOURCE = (IOB, 17, 25)

Note that the separator characters used to format the record for the 3735 disk storage file are read into the IOB along with the key and data fields when the READ command is executed. These characters will appear as an asterisk if printed, punched, or transmitted. (See the SINK = IOB discussion for further details on the format of a data record.)

SOURCE = ({ IDR / CCR })

The SOURCE = IDR (ID Reader buffer) specification directs the 3735 to get the data for the current field from an internal write/read character buffer that has already been filled by an FDCTRL COMMAND = READ (IDR) specification. The SOURCE = CCR specification is similar to SOURCE = IDR except that a COMMAND = READ (CCR) is used to fill the character buffer. Because both SOURCE requests get data from the same physical buffer within the 3735, you should ensure that a SOURCE instruction follows each READ before another READ is issued.

The maximum values of d1 and d2 are 39 when either SOURCE = IDR or SOURCE = CCR is coded. The default value of d1 is reset to 1 the next time an FDCTRL READ (IDR or CCR) or CLEAR (IDR or CCR) command is encountered in the FD macro assembly. For SOURCE = IDR, the default value of d2 is 18 for the first SOURCE = IDR encountered after a READ (IDR) or CLEAR (IDR) command. If a second SOURCE = IDR is coded before a READ or CLEAR, the default values are 19 and 36. These default values are reset to 1 and 18 when a READ (IDR) or CLEAR (IDR) command is encountered. The starting position may not be specified by an index counter.

For example, to get the data for the current field from positions 1 to 10 of the IDR buffer, you can code:

SOURCE = (IDR, 1, 10) or SOURCE = (IDR, 1, LNG (10))

The IBM Magnetic Stripe Identification Card can contain up to 18 characters; other magnetic identification cards can contain up to 39 characters. Thus, the length of a data string requested from the IDR buffer may not exceed 39 characters, and will usually be either 18 or 39 characters. You can use a similar specification to get data from the credit card reader (CCR) buffer; the length of the data string requested can be up to 39 characters.

Since both READ (IDR) and READ (CCR) get data from the same physical device (the Operator Identification Card Reader), you should take care to ensure that the operator has inserted the correct card type. The IBM Identification Card has as its first character an OID character, which is represented as a colon (:) in the buffer. The credit card has no such character. Thus, you can check the card type by coding a COMPARE operand to look for a colon in the first buffer position.

When the 3735 is in error correct mode or playback mode, coding COMMAND = READ (IDR) causes rereading of the magnetic card, while coding COMMAND = READ (CCR) causes the 3735 to get the previously-read data from disk storage.

Note: Since the 3735 performs no longitudinal redundancy check (LRC) on data from the ID reader, you may want to perform that function at the central computer if the field is transmitted to it. The record formats of IBM ID cards, user ID cards, and credit cards are shown in the *IBM 3735 Programmable Buffered Terminal Concept and Application* publication, Order No. GA27-3043. Each character consists of 4 bits (plus 1 bit for parity). Upon insertion into the buffer, a 3-bit zone digit is added to

make each character a graphic. When this data is sent to the CPU, the application program may strip out the zones, accumulate the low-order four bits of each character, and perform the LRC check function. The *(IBM 3735 Programmable Buffered Terminal Concept and Application)* publication, Order No. GA27-3043, describes the exact character representations for different character sets (including Katakana).

If you want to. . .	Code. . .	Unless you want to use the default value of. . .
Specify character set checking. . .	KIND = A for alphabetic and blank characters. KIND = N for numeric-only characters plus optional leading sign. KIND = AN for alphabetic, numeric, and blank characters (no sign or other special characters). KIND = K for Katakana characters (for IBM Japan terminals).	KIND = U (unrestricted character set)
Perform self-checking procedures on numeric data. . .	SELFCHK = d, where d is either 10 or 11, and tells the 3735 which procedure to use (modulo-10 or modulo-11). If the self-checking digit is not provided in the input data, but you want to generate one, code: SELFCHK = (d, GENONLY) and allow space in the output field for the generated digit.	SELFCHK = NO (no self-checking provided)
Check the number of characters entered by the operator. . .	COUNT = d, where d is a fixed number of characters (up to 127), or COUNT = (d1, d2), where d1 indicates the smallest number of characters that can be entered, and d2 the largest (up to 127).	d1 = 1. d2 = a value from other FDFIELD operands, such as the field length, SINK, and SOURCE.

Figure 14. Coding the FDFIELD Macro Instruction (Part 5 of 7)

Input Data Verification Operands

The input data verification operands are concerned with testing the character-set membership, quantity, and value of incoming data. Numeric self-checking procedures are also included. Any violation of the requested data checking results in an operator error indication.

KIND = { U | A | N | AN | K }

The KIND operand directs the 3735 to validate the character-set membership of each incoming character from the Selectric ® keyboard. The set codes allowed are U (unrestricted), A (alphabetic plus blank), N (numeric plus optional leading sign), AN (alphabetic, numeric, and blank), and K (Katakana). The default is U (unrestricted character set). The KIND operand coding is ignored if any of the following FDFIELD operands is coded: SOURCE = 'string', SINK = CTR (d), FILL = '0', PICTURE, CTR, BATCH, SELFCHK, or COMPARE or IND using numeric comparands.

If a character that does not belong to the specified character set appears in the field input, the 3735 operator is notified and, when SOURCE = KBD, is given an opportunity to correct the erroneous character. If the data is from a source other than the keyboard, a data source error indication is provided. For example, if you plan to use the data in some field for arithmetic computations, you can code KIND = N to ensure that only numeric characters are entered into the field.

Note: KIND = K may be coded only if DEVICES = (3735,K) or DEVICES = (3735,KD) was coded in FDFORM. For complete details of the Katakana support, refer to Appendix J.

SELFCHK = { NO | ({ 10 | 11 } [,GENONLY]) }

The SELFCHK operand directs the terminal to perform one of two standard accounting procedures (modulo-10 or modulo-11) to generate a self-check digit from numeric data characters. The qualifying suboperand GENONLY informs the 3735 that the incoming data contains no self-check digit against which the generated self-check digit is to be compared.

When SELFCHK is used with GENONLY, the source character count (whether specified or implied) must not allow a position for the self-check digit. Subsequent operations (such as COMPARE and SINK) must allow one position for the self-check digit, since the self-check digit is placed at the end of the numeric data string. When SELFCHK is coded, KIND is forced to numeric, and any KIND specification coded is ignored.

If the SELFCHK comparison detects an error, the 3735 gives the operator an opportunity to enter the correct data when SOURCE = KBD. If the data is from a source other than the keyboard, a data source error indication is provided. If SINK = PRT is coded, the erroneous data may be printed on the Selectric ® form. This can be avoided by coding SINK = (PRT, AFTER).

Note: If only a modulus number is coded, the enclosing parentheses may be omitted. Coding the qualifier GENONLY is required for the sources FID and RSN, and SELFCHK may not be coded for SOURCE = 'string'.

The modulo-10 routine is designed primarily to detect the most common types of errors, the incorrect keying of a single digit, and a single transposition of 2 digits. The modulo-11 routine is designed to detect single-digit keying errors, single transpositions, and double transpositions. The algorithms used to generate the self-check digit are described in Appendix A.

COUNT = { ([MIN,] { 1 | d1 }
,[MAX ,] { compmax | d2 }) }

The COUNT operand directs the 3735 to validate the number of characters entered by the operator. The suboperands can specify tests for minimum and maximum limiting counts (SOURCE = KBD only), or else for one exact count (any SOURCE). The values of d, d1, and d2 may range from 1 to 127. If the optional MIN and MAX suboperands are coded, d1 must be less than d2. Thus, while COUNT = (100,1) is permitted (and is interpreted as min = 1, max = 100), COUNT = (MIN, 100, MAX, 1) is not (and will be flagged as an error). If this operand is not coded, default values of a minimum count of 1 and a maximum count (compmax) obtained from the current values of other FDFIELD operands, such as the field length, SOURCE, and SINK are used. The COUNT operand is ignored for the sources FID, RSN, and CTR (d), and for all buffered sources with d2 or LNG (d) coded.

For example, if the operator can enter from two to five characters in the current field, you can check for a number within these limits by coding:

COUNT = (2,5)

Or, if the operator must enter exactly four characters, you can code:

COUNT = 4

If the count test fails, the 3735 turns on an error indicator and, when SOURCE = KBD, gives the operator another opportunity to enter the correct data. If the data is from a source other than the keyboard, a data source error indication is provided.

Note: The COUNT operand is not promotable. The MIN and MAX suboperands are optional encodings allowed for their value as memory aids, but are not required. However, if they are coded, d1 must be less than d2.

If you want to. . .	Code. . .	Unless you want to use the default value of. . .
Compare the field data with a particular comparand. . . such as. . . See if the entered data is the decimal number 123. . . See if the entered data is an "A" or a "B" character. . .	COMPARE = (operator, comparand) COMPARE = (EQ, 123) COMPARE = (EQ, 'A', OR, EQ, 'B') (Tests may be combined by the logical operators AND and OR.)	No comparsion test of the field data. The comparison operators are: GT greater than NG not greater than EQ equal to NE not equal to LT less than NL not less than GE greater than or equal to LE less than or equal to
Specify the destination of the data from this field. . . such as. . . The Selectric printer and the central computer. . . Positions 4 to 37 of the inquiry buffer. . . Counter 10. . .	SINK = (name, name, . . .), which may name up to 5 destinations for the data, including: • Selectric printer (PRT) • Sent to central computer (TMT) • Card punch buffer (PCH) • Storage buffer (STG) • Inquiry buffer (INQ) • Line printer buffer (LPB) • Read/punch buffer (RPB) • Input/output buffer (IOB) • A counter (CTR (d), where d is the number of the counter) SINK = (PRT, TMT) SINK = (INQ, 4, 37) or. . . SINK = (INQ, 4, LNG (34)) SINK = CTR (10)	SINK = NULL (no destination — which is useful for control operations) *Note:* All requests to place data in a buffer (PCH, STG, INQ, LPB, RPB, or IOB) should be made this way.

Figure 14. Coding the FDFIELD Macro Intruction (Part 6 of 7)

COMPARE = ([FIELD ,] comparopr , comparand [, {AND / OR} ,[FIELD ,] comparopr,comparand] . . .)

The COMPARE operand directs the 3735 to evaluate the characters entered by the operator. Each test consists of comparing the data input for the current field to a fixed comparand (comparand) as specified by a comparison operator (comparopr). The FIELD specification is an optional encoding allowed for its value as a memory aid, but is not required. The compare function is performed in all operating modes.

All comparisons except numeric-only are made against character strings. If one string is shorter than the other, it is extended to the right with blanks (SP characters) until its length equals that of the longer string. All string comparisons are made with the collating sequence implicit in the transmission code of the particular 3735 (EBCDIC or ASCII). The allowable comparison operators (comparopr) are as follows (framing apostrophes are required where shown):

GT or	'>'	Greater than
NG or	'¬>'	Not greater than
EQ or	'='	Equal to
NE or	'¬='	Not equal to
LT or	'<'	Less than
NL or	'¬<'	Not less than
GE or	'>='	Greater than or equal to
LE or	'<='	Less than or equal to

The format of comparand is either [+] decdigits or -decdigits for numeric comparison, or 'string' for character comparison. A + is assumed when no sign is indicated. The sizes of comparands may range from 1 through a maximum number of characters that depends on the Assembler used and on the number of comparisons coded. In no case, however, may the net number of characters per comparand exceed 127.

The results of the individual tests can be combined logically (logopr) with the logical operators AND (or '&&') and OR (or '|'). Evaluation and combination proceeds from left to right through the operand terms, with AND taking precedence over OR. Grouping to force OR to take precedence over AND is not permitted.

For example, if an operator must enter nothing or the character "Z" to indicate what processing is to follow, you can code:

```
FDFIELD SOURCE = (KBD, OPTIONAL) ,COUNT = 1 ,COMPARE = (EQ, 'Z')
```

Thus, if data is entered, it must be only the character "Z". If you also want to set an indicator, you could do so with the IND operand by coding:

```
IND = (1 ,EQ, 'Z')
```

You can then test this indicator when the processing decision must be made later in the program.

If the comparison fails (the entered data does not have the desired relation to the comparand), the 3735 turns on an error indicator and, if SOURCE = KBD, gives the operator another opportunity to enter the correct data. If the data is from a source other than the keyboard, a data source error indication is provided. If SINK = PRT is coded, the erroneous data may be printed on the Selectric ® form. This can be avoided by coding SINK = (PRT, AFTER).

Note: The COMPARE operand is not promotable. Compare operations are not permitted with SOURCE = 'string' or SOURCE = FID data.

Sink Keyword Operands

SINK = ([(destination[,qualifier] . . .)] [,[(destination[,qualifier] . . .)]] . . .)

The SINK operand indicates, for each document field, the destination of the data from that field. If no SINK operand is coded, a default value of SINK = NULL (no sink) is used.

From one to five destinations (sinks) may be specified for each field. The order in which they are coded establishes a corresponding positional dependence in the editing operands JUSTIFY, FILL, UL, and PICTURE. Only buffered sinks may be coded more than once in any particular SINK operand. When a buffered sink is coded more than once, the 3735 performs the specified sink operations in the indicated sequence. An example of a SINK specification that describes more than one type of data sink is found near the end of the "Output Data Editing Operands" discussion. The allowable values of destination and qualifier are described in the following paragraphs.

Note: When one of several sinks is unqualified, the inner grouping parentheses for that sink may be omitted. When only one sink is coded and it is unqualified, the outer parentheses may also be omitted.

Non-Buffered Sinks: The non-buffered sinks are those destinations for data from the field that are null, do not reside in the 3735, or are of fixed length and fixed content-type.

SINK = NULL

The SINK = NULL specification may be coded to override a higher-level value for the scope of the current macro statement. It is also the default value when no sink is specified. For example, if SINK = (PRT,TMT) is in effect from the FDFORM level, and if you do not want to transmit the data from the present field, then this FDFIELD macro statement can be coded with SINK = (,NULL). This permits printing, but inhibits transmission to the central computer.

SINK = (PRT [,AFTER]

The SINK = PRT (print) specification requests the 3735 to direct the data from the current field to the Selectric ® printer within the field space specified by dl and dr in field location operands. The optional qualifier AFTER may be coded to prevent input data from printing until data entry for the field is complete. Coding AFTER also suppresses printing until the input data has successfully passed all the input data verification checks that were coded.

If requests have been coded for editing of outgoing data, or if the qualifier AFTER is coded, the data is printed only after the requested functions are completed. AFTER occurs automatically if PICTURE, JUSTIFY = R or C, or FILL = 0 is specified for the PRT sink. Data is directed to the Selectric ® printer in all operating modes.

SINK = (TMT [,AFTER]

The SINK = TMT (transmit) specification directs the 3735 to mark the data from the current field as destined for transmission to a central computer through the transmission interface (the line adapter). You do not specify the range of character positions within the data record being prepared for transmission, since this is regulated internally by the 3735. This sink is active only in enter form mode.

The qualifier AFTER requests the 3735 to delay marking the data for transmission until all other FDFIELD operations are complete. This allows the TMT sink to become conditional based on an IND branch. AFTER occurs automatically if PICTURE, JUSTIFY = R or C, or FILL = 0 is specified for the TMT sink.

Two frequent destinations for field data are the Selectric printer and the central computer. In many forms, you may find that coding SINK = (PRT,TMT) in a higher-level macro statement saves much detailed coding at the field level. Where other data sinks are desired, the promoted specification may be modified by coding other SINK operands in the appropriate FDFIELD macros.

When an FDP creates a data record, the first three characters of the data record contain the FDP identification number that was coded in the FDFORM FID operand. This identification number is always sent to the CPU, whether or not any other data is sent.

SINK = CTR (d)

The SINK = CTR (d) specification directs the 3735 to store the data from the field in the counter specified by d. The value of d may range from 1 to 21. The data may consist of

up to 10 numeric characters (plus optional sign). (A negative value is represented in the counter by an overpunch in the units position.) If the data does not fill the counter, the 3735 pads the counter to the left by inserting high-order zeros. Data is placed in the specified counter in all operating modes. When SINK = CTR (d) is coded, the KIND option is forced to numeric, and any coding of KIND is ignored.

If SOURCE = 'string' is coded, SINK = CTR (d) may not be coded. To load a numeric constant into a 3735 counter, you should code an FDCTRL macro that specifies CTR = (d,CLR,ADD,constant).

Buffered Sinks: The buffered sinks are those destinations for data that are of fixed length and variable content, and are byte-addressable. The coding format for all buffered sinks is:

SINK = (bufname[,{1 / prevd2+1 / d1 / Xn}] [, { complng / LNG (d) / d2 }])

where:

bufname specifies a code name for the buffer to be used as the data sink.

The qualifying suboperands d1 (or Xn) and LNG (d) (or d2) specify the range of contiguous character positions within the buffer into which the terminal is to put the data. These suboperands specify either the lower-numbered character position (d1) or one of two index counters (Xn) that should be used to determine the starting position, and either the data field length (LNG (d), where d is the number of character positions), or the higher-numbered character position (d2). Neither d1 nor d2 can be less than 1 or greater than a buffer-dependent value, nor can d1 exceed d2. Further, the data length may not exceed 127 (the maximum length of a 3735 field). Similar restrictions apply to LNG (d), since d2 = d1 + d-1. If d1 is not coded and an index counter has not been promoted, the default value is position 1 if the present request for data movement is the first coded since the last encoding of certain buffer-dependent FDCTRL commands. Otherwise, the previous d2 plus one is used. Note, however, that you can direct data to the same buffer positions as many times as you wish by explicit encoding (the next d1 does not have to be greater than the previous d2). You should take care, therefore, to avoid overlaying data by mistake.

If the starting position (d1) is omitted at the field level but the sink was promoted with an index counter starting position, then the index counter is used at the field level. Otherwise, if the starting position is omitted and no index counter has been promoted, then the starting position is either 1 or the last end position plus 1.

The File Storage capability must be present in order to specify and use the index counters. When using one of the index counters (X1 or X2) to identify the starting position in the buffer, the position is offset from character position 1 of the buffer by the value stored in the index counter. When an index counter is used to locate the starting position, the second qualifier (if used) must be the data length, LNG (d). In addition, an index counter may not be specified for the PCH buffer. If you want to set data in this buffer using index counters to identify the starting point, you should use SINK = RPB instead. You can place a value in an index counter by coding the FDCTRL CTR operand. (See the FDCTRL discussion for further details.) If d2 or LNG (d) is not coded, the default value (complng) is obtained from the current values of other FDFIELD operands, such as the field length, COUNT, and SOURCE.

Note: The d2 (or LNG (d)) specification is not promotable, since the size of the field is not significant above the field structural level.

Numeric data directed to buffered sinks may have a leading sign (+ or -, which may be preceded by leading blanks), or may have an overpunch in the low-order digit position to represent a negative number. The overpunch is stripped off for printing (unless a signed PICTURE specification is used), but is retained when the number is stored right-justified in buffered sinks or stored on disk for transmission to the CPU.

SINK = PCH

The SINK = PCH (card punch) specification directs the 3735 to put the data from the current field into an internal write-only card-image buffer that is subsequently directed to the card punch by a PUNCH command (see the description of the COMMAND operand in the FDCTRL discussion). For example, to put the data from the current field into positions 21 to 30 of the card punch buffer (PCH), you can code:

SINK = (PCH ,21 , 30) or SINK = (PCH, 21 , LNG (10))

Since the card image consists of exactly 96 characters, take care that d2 is not greater than 96, and observe a similar limit for LNG (d).

SINK = PCH cannot be coded when BUFFERS = RPB is coded in FDFORM. SINK = PCH and SINK = RPB are mutually exclusive. The PCH buffer is not saved at the beginning of each FDP, nor is it restored if the operator enters error correct mode. Data is stored in the buffer in all operating modes.

The maximum values of d1 and d2 are 96 when SINK = PCH is coded. The default value of d1 is reset to 1 the next time an FDCTRL CLEAR (PCH) or PUNCH command is encountered in the FD macro assembly.

SINK = STG

The SINK = STG (storage) specification directs the 3735 to put the data from the current field into an internal write/read character buffer, where it may be extracted later by a macro statement that specifies SOURCE = STG. For example, to put the data from the current field into positions 26 to 50 of the storage buffer, you can code:

SINK = (STG, 26, 50) or SINK = (STG, 26, LNG (25))

The storage buffer (STG) is 236 bytes long, but the length of a data string directed to it may not exceed 127 characters (the maximum length of a 3735 field).

At the beginning of each FDP (or the re-execution of a single FDP), the contents of the storage buffer are saved. If the record is canceled, the storage buffer is restored to its state at the beginning of the record. During playback mode, no data is stored in the buffer.

The maximum values of d1 and d2 are 236 when SINK = STG is coded. The default value of d1 is reset to 1 the next time an FDCTRL CLEAR (STG) command is encountered in the FD macro assembly.

SINK = INQ

The SINK = INQ (inquiry) specification directs the 3735 to put the data from the current field into an internal write/read buffer, where it may be transmitted later to the central computer as a 3735 inquiry message. For example, to put the data from the current field into positions 201 to 236 of the inquiry buffer, you can code:

SINK = (ING, 201, 236) or SINK = (INQ, 201, LNG (36))

The inquiry buffer (INQ) is 236 bytes long, but the length of a data string directed to it may not exceed 127 characters (the maximum length of a 3735 field).

When using the inquiry buffer for inquiry operations (instead of for temporary data storage), you should not place data in the first three positions of the buffer, since the 3735 Terminal Control Program (TCP) places a three-byte message header in these positions when the inquiry is sent to the CPU. See the "System Design Considerations" section for further details on inquiry message handling.

The inquiry buffer is not saved at the beginning of each FDP, nor is it restored if the operator enters error correct mode. During playback mode, no data is stored in the buffer.

The maximum values of d1 and d2 are 236 when SINK = INQ is coded. The default value of d1 is reset to 1 the next time an FDCTRL CLEAR (INQ) or SEND command is encountered in the FD macro assembly.

SINK = LPB

The SINK = LPB (line printer buffer) specification directs the 3735 to put the data from the current field into an internal write/read character buffer, where it may be retrieved by a SOURCE = LPB specification or printed on an attached 3286 printer by an FDCTRL statement that specifies COMMAND = PRINT (see the description of the COMMAND operand in the FDCTRL discussion). Data is stored in the buffer in all operating modes.

For example, to put the data from the current field into positions 100 to 104 of the line printer buffer, you can code:

SINK = (LPB, 100, 104) or SINK = (LPB, 100 LNG (5))

The maximum values of d1 and d2 are 132 when SINK = LPB is coded, unless BUFFERS = (LPB,ddd) was coded in FDFORM. In this case, the maximum is the value of ddd that was coded (up to 236). Note, however, that only the first 132 bytes of the buffer are moved to the 3286 printer when a PRINT command is executed. (If BUFFERS = (LPB,ddd) was coded, and ddd is either 120 or 126, the number of bytes moved to the 3286 printer is adjusted accordingly.) In addition, the length of a data string directed to the buffer may not exceed 127 (the maximum length of a 3735 field). The default value of d1 is reset to 1 the next time an FDCTRL CLEAR (LPB) or PRINT command is encountered in the FD macro assembly.

SINK = RPB

The SINK = RPB (read/punch buffer) specification directs the 3735 to put the data from the current field into an internal 192-character write/read buffer, where it may be retrieved by a SOURCE = RPB specification. For example, to put the data from the current field into positions 101 to 125 of the read/punch buffer, you can code:

SINK = (RPB, 101, 125) or SINK = (RPB, 101, LNG (25))

(RPB positions 97 to 192 can also be punched on a 5496 by an FDCTRL PUNCH command, since the read/punch buffer occupies the same internal storage as the RDR buffer (96 characters) and the PCH buffer (96 characters) combined.) SINK = RPB cannot be coded unless BUFFERS = RPB was coded in the FDFORM macro. SINK = RPB and SINK = PCH are mutually exclusive. Data is stored in the buffer in all operating modes.

The maximum values of d1 and d2 are 132 when SINK = RPB is coded. In addition, the length of a data string directed to the read/punch buffer may not exceed 127 (the maximum length of a 3735 field). The default value of d1 is reset to 1 the next time an FDCTRL CLEAR (RPB) or PUNCH command is encountered in the FD macro assembly.

SINK = IOB [,DELIMIT]

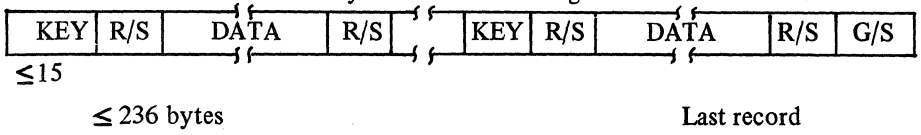
The SINK = IOB (input/output buffer) specification (for the File Storage capability only) directs the 3735 to put the data from the current field into an internal 236-character write/read buffer, where it may either be written to the 3735 disk storage file by an FDCTRL COMMAND = WRITE (file qualifier) specification or retrieved by a SOURCE = IOB specification. When used to update the 3735 disk storage file, the data must be organized as a key field starting in buffer position 1 and extending for up to 15 characters, a one-byte key field delimiter (record separator), a data field, and a one-byte data field delimiter (record separator). When the qualifier DELIMIT is used, the field delimiter is automatically placed after the last character currently being placed in the IOB. For example, if the data portion of the output record is to be 100 bytes from the current field, and you want the 3735 to insert the end-of-data delimiter automatically, and the starting position in the IOB is pointed to by index counter 2, you might code:

SINK = (IOB, X2, LNG (100), DELIMIT)

Use a similar coding to insert the key and its delimiter.

The I/O buffer (IOB) is 236 bytes long, but the length of a data string directed to it may not exceed 127 characters (the maximum length of a 3735 field). Further, since there are two record separator characters, the total key plus data length cannot exceed 234.

The maximum values of d1 and d2 are 236 when SINK = IOB is coded. The default value of d1 is reset to 1 the next time an FDCTRL CLEAR (IOB) or WRITE command is encountered in the FDP assembly. The file record organization is as follows:



where the key (KEY) may be up to 15 bytes long, the record separator character (R/S) is represented by X'14', and the combined length of the key, the data, and the delimiters can be up to 236 bytes long. The group separator character (G/S) is represented by X'1D' and acts as an end-of-file indicator for the 3735 disk storage file records.

Note: Although there is only one physical file, several logical files are possible by using unique keys or different length keys for each logical file. The terminal control program automatically handles the position of the group separator (G/S), which also follows the last valid data record in the file.

If you want to. . .	Code. . .	Unless you want to use the default value of. . .
Specify how the data for each sink is to be justified. . . such as. . . For SINK = (PRT, TMT) – print right justified, send left justified. . .	JUSTIFY = L for left justification JUSTIFY = R for right justification JUSTIFY = C centering JUSTIFY = (R, L)	JUSTIFY = L.
Specify how unused leading spaces in each sink are to be filled. .	FILL = ' ' for blanks FILL = '0' for zeros	FILL = ' '.
Underline the data at the Selectric sink. . .	UL = YES, which underlines all field data except blank FILL characters (if any).	UL = NO (no underlining)
Specify editing of numeric data (such as dollar sign, comma, and decimal point insertion). . .	PICTURE = 'specification' (The PICTURE specification symbols are shown in Figure 15, and some examples are shown in Figure 16.)	No editing of numeric data.
Identify the data from this field as belonging to a particular data batch (for later totaling by an FDCTRL TOTAL instruction). . .	BATCH = d where d is a number from 1 to 128.	No batch identification.

Figure 14. Coding the FDFIELD Macro Instruction (Part 7 of 7)

Output Data Editing Operands

The output data editing operands allow you to control the appearance of the output data. None of the editing operands applies to SINK = NULL or SINK = CTR (d); they are ignored if so coded.

JUSTIFY = ([justcode]
[, [justcode]] . . .)

The JUSTIFY operand specifies, sink by sink, how the 3735 is to position (justify) the data within the data sink. The justification codes (justcode) allowed are L (left), R (right), and C (center). The default value is L (left) for all sinks. The justification codes specified in a one-to-one positional correspondence with the sinks. For example, the third justification code applies to the third sink.

The JUSTIFY option does not apply to SINK = NULL, SINK = CTR (d), or to any buffered sink that is described by a PICTURE operand. Any JUSTIFY option coded for such sinks is ignored. When no JUSTIFY option is specified (or JUSTIFY = L is coded), the source data is placed flush left in the field, and any unused positions to the right of the data are filled with blanks. When JUSTIFY = C is coded, the source data is centered in the field. Any blanks in the source data are considered as characters for centering computations.

For JUSTIFY = L or JUSTIFY = C, if the source data is numeric and contains a negative sign overpunch in the low-order digit position, the overpunched digit is transferred to the sink field only if the overpunch digit is stored in the rightmost sink character position (that is, only if the data completely fills the sink).

When JUSTIFY = R coded, the source data is placed flush right in the field. Any leading zeros in the source data are filled with blanks (unless FILL = '0' is coded). If the source data is numeric and contains a negative sign overpunch in the low-order digit position, the overpunched digit is stored in the rightmost sink character position.

FILL = (['character']
[, ['character']] . . .)

The FILL operand specifies, sink by sink, strings each consisting of a single graphic character ('character') with which the 3735 is to fill all unused or non-significant leading positions in the source data before the data is moved to the specified sink. The specifications allowed are ' ' (blank) and '0' (zero). The default value is ' ' (blank) for all sinks. Coding FILL = '0' implies numeric KIND; any KIND option coded is ignored. The fill codes are specified in a one-to-one positional correspondence with the sinks. For example, the third fill code applies to the third sink. FILL should not be coded unless JUSTIFY = R is coded. The FILL option does not apply to SINK = NULL, SINK = CTR (d), or to any sink that is described by a PICTURE operand. Any FILL option coded for such sinks is ignored.

UL = ([{ NO / YES }]
[,[{ NO / YES }]] . . .)

The UL operand tells the 3735 whether or not to underline the printed field data. If YES is coded, the underlining begins at the leftmost nonblank character and ends at the rightmost nonblank character. Blank fill characters (if any) are not underlined. If this operand is not coded, or if it is coded as NO, no underlining is performed. The underlining codes are specified in a one-to-one positional correspondence with the sinks. For example, the third underlining code applies to the third sink.

The UL operand applies only to the Selectric ® printer, and is ignored if coded YES for any other sink. If only one UL specification is coded, the enclosing parentheses may be omitted.

PICTURE = (['picturespec']
[, ['picturespec ']] . . .)

The PICTURE operand specifies, sink by sink, the desired appearance of numeric data directed to the sinks. The format and allowable editing characters of picturespec are a subset of the PL/I PICTURE specification that governs decimal data output. Figure 15 is a chart that summarizes the function of each permitted character.

The lengths of picturespec definitions, not counting the framing apostrophes (which are required), may range from 1 to a maximum number of characters that depends on the Assembler used and the number of specifications coded. In no case, however, may the number of character positions implied by a single picturespec exceed 127. The picturespec codes are specified in a one-to-one positional correspondence with the sinks. For example, the third picturespec code applies to the third sink.

Note: The PICTURE operand is not promotable. If only one picturespec is coded, the framing parentheses may be omitted. When PICTURE is coded, the KIND option is forced to numeric, and any KIND specification coded is ignored. PICTURE may not be coded when SOURCE = 'string' is coded.

A PICTURE specification always describes a character representation of a *numeric character data item* – one in which the data itself can consist only of decimal digits and, optionally, a plus or minus sign. Other characters generally associated with arithmetic data, such as decimal points and currency symbols, can also be specified, but they are not a part of the arithmetic value of the numeric-character data. However, the editing PICTUR characters are considered to be a part of the character-string value of the data item. Thus, the total number of characters in a PICTURE specification must not be greater than the size of the output sink that is to receive the edited data.

The PICTURE characters for numeric-character specifications may be grouped into the following categories:

- Digit specifiers.
- Zero suppression characters.
- Insertion characters.
- Signs and currency symbol.
- Credit and debit signs.

Edit Char	Result
9	A decimal digit is accepted for output in this position.
V	Stops suppression of zeros and insertion characters.
Z	The position is made blank if it contains a leading zero.
*	An asterisk is placed in this position if it contains a leading zero.
Y	This position is made blank if it contains a zero.
,	A comma is placed in this position if zeros are not being suppressed.
.	A period is placed in this position if zeros are not being suppressed.
/	A slash is placed in this position if zeros are not being suppressed.
B	A blank is inserted in this position.
$	A dollar sign is inserted in this position. If more than one $ is coded, only one $ is placed to the left of the most significant digit.
S	A minus sign is placed in this position if the field is less than zero; a plus sign is placed in this position if the field is greater than zero. If more than one S is coded, only one sign is placed to the left of the most significant digit.
+	A plus sign is placed in this position if the field is greater than zero. If more than one + is coded, only one + is placed to the left of the most significant digit.
–	A minus sign is placed in this position if the field is less than zero. If more than one – is coded, only one – is placed to the left of the most significant digit.
CR	CR is placed in these positions if the field is less than zero; otherwise, these positions are left blank.
DB	DB is placed in these positions if the field is less than zero; otherwise, these positions are left blank.

Figure 15. Summary of PICTURE Character Functions

The PICTURE characters in these groups may be used in various combinations. Consequently, a numeric character specification can consist of two or more parts such as a sign specification, an integer subfield, and a fractional subfield.

A major requirement of the PICTURE specification for numeric-character data is that each 'picturespec' must contain, in the first two positions of the 'picturespec', at least one PICTURE character that specifies a digit position. This character, however, need not be the digit character 9. Other PICTURE characters, such as the zero suppression characters (Z or * or Y), also specify digit positions.

The PICTURE characters 9 and V control many kinds of numeric character specifications:

9 specifies that the associated position in the data item is to contain a decimal digit.

V stops suppression of zeros and insertion characters. The V character cannot appear more than once in a PICTURE specification. The V is considered to be a subfield delimiter in the PICTURE specification; that is, the portion preceding the V and the portion following it (if any) are each a subfield of the specification.

The zero suppression PICTURE characters specify conditional digit positions in the character-string value and may cause leading zeros to be replaced by asterisks or blanks and nonleading zeros to be replaced by blanks. Leading zeros are those that occur in the leftmost digit positions of numeric data strings. The leftmost nonzero digit in a number and all digits, zeros or not, to the right of it represent significant digits.

Z specifies a conditional digit position and causes a leading zero in the associated data position to be replaced by a blank character. When the associated data position does not contain a leading zero, the digit in the position is not replaced by a blank character. The PICTURE character Z cannot appear in the same subfield as the PICTURE character *, nor can it appear to the right of a drifting PICTURE character or the PICTURE character 9.

* specifies a conditional digit position and is used the way the PICTURE character Z is used, except that the leading zeros are replaced by asterisks. The PICTURE character * cannot appear with the PICTURE character Z in the same subfield, nor can it appear to the right of a drifting PICTURE character or the PICTURE character 9.

Y specifies a conditional digit position and causes a zero digit, leading or nonleading, in the associated position to be replaced by a blank character. When the associated position does not contain a zero digit, the digit in the position is not replaced by a blank character.

The PICTURE characters comma (,) , point (.) , slash (/), and blank (B) are insertion characters; they cause the specified character to be inserted into the associated position of the numeric character data. They do not indicate digit positions, but are inserted between digits. Each does, however, actually represent a character position in the character-string value, wheither or not the character is suppressed. The comma, point, and slash are conditional insertion characters; within a string of zero suppression characters, they, too, may be suppressed. The blank (B) is an unconditional insertion character; it always specifies that a blank is to appear in the associated position.

, causes a comma to be inserted into the associated position of the numeric character data when no zero suppression occurs. If zero suppression does occur, the comma is inserted only when an unsuppressed digit appears to the left of the comma position, or when a V appears immediately to the left of it. In all other cases when zero suppression occurs, one of three possible characters is inserted in place of the comma. The choice of character to replace the comma depends upon the first PICTURE character that both precedes the comma position and specifies a digit position:

If this character position is an asterisk, the comma position is assigned an asterisk.

If this character position is a drifting sign or a drifting currency symbol (discussed later), the drifting string is assumed to include the comma position, which is assigned the drifting character.
If this character position is not an asterisk or a drifting character, the comma position is assigned a blank character.

is used the same way the comma PICTURE character is used, except that a point (.) is assigned to the associated position.

is used the same way the comma PICTURE character is used, except that a slash (/) is assigned to the associated position.

B specifies that a blank character always be inserted into the associated position of the numeric character data.

The PICTURE characters S, +, and - specify signs in numeric character data. The PICTURE character $ specifies a currency symbol in the numeric character data.

These characters may be used in either a static or a drifting manner. A drifting character is similar to a zero suppression character in that it can cause zero suppression. However, the character specified by the drifting string is always inserted in the position specified by the end of the drifting string or in the position immediately to the left of the first significant digit.

The static use of these characters specifies that a sign, a currency symbol, or a blank *always* appears in the associated position. The drifting use specifies that leading zeros are to be suppressed. In this case, the rightmost suppressed position associated with the PICTURE character will contain a sign, a blank, or a currency symbol.

A drifting character is specified by multiple use of that character in a PICTURE field. Thus, if a field contains one currency symbol ($), it is interpreted as static; if it contains more than one, it is interpreted as drifting. The drifting character must be specified in each digit position through which it may drift.

Drifting characters must appear in strings. A string is a sequence of the same drifting character, optionally containing a V and one of the insertion characters comma, point, slash, or B. Any of the insertion characters following the last drifting symbol of the string is considered part of the drifting string. However, a following V terminates the drifting string and is not part of it. A field of a PICTURE specification can contain only one drifting string. A drifting string cannot be preceded by a digit position. The PICTURE characters *, Y, and Z cannot appear to the right of a drifting string in a field.

The position in the data associated with the characters slash, comma, point, and B appearing in a string of drifting characters will contain one of the following:

- slash, comma, point or blank if a significant digit has appeared to the left.
- the drifting symbol, if the next position to the right contains the leftmost significant digit of the field.
- blank, if the leftmost significant digit of the field is more than one position to the right.

If a drifting string contains the drifting character *n* times, then the string is associated with *n-1* conditional digit positions. The position associated with the leftmost drifting character can contain only the drifting character or blank, never a digit. If a drifting string is specified for a field, the other potentially drifting characters can appear only once in the field, that is, the other character represents a static sign or currency symbol.

Only one type of sign character can appear in each field. Any field that has a leading sign (+ or -) must be pictured with S, +, or - to display the sign. Otherwise, the sign is assumed to be a leading zero. An S, +, or - used as a static character can appear to the right or left of all digit positions of a PICTURE specification.

$ specifies the currency symbol. If this character appears more than once, it is a drifting character; otherwise it is a static character. The static character specifies that the character is to be placed in the associated position. The static character must appear either to the left of all digit positions in a specification or to the right of all digit positions in a specification.

S specifies the plus sign character (+) if the data value is greater than 0; it specifies the minus sign character (-) if the data value is less than 0. The character may be drifting or static. The rules are identical to those for the currency symbol.

+ specifies the plus sign character (+) if the data value is greater than 0; otherwise it specifies a blank. The character may be drifting or static. The rules are identical to those for the currency symbol.

- specifies the minus sign character (-) if the data value is less than 0; otherwise it specifies a blank. The character may be drifting or static. The rules are identical to those for the currency symbol.

The character pairs CR (credit) and DB (debit) specify the signs of real numeric character data items and are commonly used for business report forms.

CR specifies that the associated positions will contain the letters CR if the value of the data is less than zero. Otherwise, the positions will contain two blanks. The characters CR can appear only to the right of all digit positions of a field.

DB is used the same way that CR is used except that the letters DB appear in the associated positions when DB is coded.

Note: The PICTURE characters CR and DB cannot be used with any other sign characters in the same field.

A full description of the PICTURE specification characters is found in the *OS PL/I (F) Language Reference Manual,* Order No. GC28-8201, or the *Disk and Tape Operating Systems PL/I Subset Reference Manual,* Order No. GC28-8202. Note, however, that the character V has no radix significance for the 3735, which performs only integer arithmetic. The V is used only to halt the suppression of zeros and insertion characters by the 3735. The FD macros do not permit the use of replication factors, as in '(125) 9'. Figure 16 shows some sample PICTURE specifications.

The editing operands (JUSTIFY, FILL, UL, and PICTURE) must be coded so that the position of each suboperand specified corresponds to the position of the SINK suboperand to which it applies. For example, suppose you have a 10-character field that is to be filled from the Selectric ® keyboard, and that you want the data directed to the Selectric ® printer, the central computer, and storage buffer positions 16 to 25. At this point, your FDFIELD macro statement would look like this:

```
FDFIELD  SOURCE = KBD  ,  SINK = (PRT, TMT, (STG, 16, 25) )
```

The first sink is the Selectric printer.
The second sink is the central computer.
The third sink is storage buffer positions 16 to 25.

If you now decide to specify some editing operands to handle justification and unused character-position filling when the the operator enters fewer than 10 characters, you need to specify them in the same positional relationships as the sinks. Suppose that you want blanks in all unused spaces for all the sinks and right justification on the Selectric printer, but left justification for the other sinks. In this case, you would continue the FDFIELD coding like this:

```
JUSTIFY = (R, L, L) , FILL = (‘   ’ , ‘   ’ , ‘   ’)
```

When only one suboperand is coded for any particular editing operand, it applies only to the first sink, and the enclosing parentheses may be omitted.

BATCH = d

The BATCH operand instructs the 3735 to flag the data resulting from operations on the current field as being a member of a data batch that is identified by the number d. The value of d may be specified from 1 to 128. When this operand is coded, KIND = N is assumed. Any KIND option coded is ignored.

Such flagged data can then be accumulated at the end of a day's operations by an FDP designed for such processing. (The accumulations are performed by the FDCTRL TOTAL

operand.) The resulting totals can serve as a cross-check for data transmitted to the central computer (to make sure that all records have been transmitted correctly), and can provide the 3735 location with an up-to-date accounting of the day's activities.

Note: The BATCH operand is not promotable.

SOURCE DATA	PICTURE SPECIFICATION	OUTPUT FIELD
1234	9,999	1,234
123456	9,999.99	1,234.56
1234	ZZ.ZZ	12.34
1234	ZZV.99	12.34
0003	ZZ.ZZ	bbbb3
0003	ZZV.99	bb.03
0000	ZZ.ZZ	bbbbb
0000	ZZV.99	bb.00
123456789	9,999,999.99	1,234,567.89
1234567	**,999.99	12,345.67
0012345	**,999.99	***123.45
123456789	9.999.999,99	1.234.567,89
123456	99/99/99	12/34/56
123456	99.9/99.9	12.3/45.6
001234	ZZ/ZZ/ZZ	bbb12/34
000012	ZZ/ZZ/ZZ	bbbbbb12
000000	ZZ/ZZ/ZZ	bbbbbbbb
000000	**/**/**	********
123456	99B99B99	12b34b56
123	9BB9BB9	1bb2bb3
12	9BB/9BB	1bb/2bb

Examples of Insertion Characters

SOURCE DATA	PICTURE SPECIFICATION	RESULTANT OUTPUT
12345	ZZZ99	12345
00100	ZZZ99	bb100
00000	ZZZ99	bbb00
00100	ZZZZZ	bb100
00000	ZZZZZ	bbbbb
00100	*****	**100
00000	*****	*****
00100	YYYYY	bb1bb
10203	9Y9Y9	1b2b3b

Examples of Zero Suppression

SOURCE DATA	PICTURE SPECIFICATION	RESULTANT OUTPUT
12345	$999.99	$123.45
00123	$ZZZ.99	$bb1.23
00000	$ZZZ.ZZ	bbbbbb
12345	$$$9.99	$123.45
00123	$$$9.99	bb$1.23
12	$$$,999	bbb$012
1234	$$$,999	b$1,234
12345	S999.99	+123.45
-12345	S999.99	-123.45
-12345	+999.99	b123.45
12345	-999.99	b123.45
00123	++B+9.99	bbb+1.23
00123	--9.99	bbb1.23
-00123	SSS9.99	bb-1.23

Examples of Drifting Characters

Figure 16. Examples of PICTURE Specifications

Procedural Form Description Macro Instructions

Many of the logical functions available in the 3735 are controlled by the procedural FD macro instruction, FDCTRL. This macro provides decision-making facilities, allows your FDP to test and set indicators, operate on counters, branch around one or more FD macro statements, and perform I/O and batch totalling operations. Sample programs in this section and in Appendix C illustrate some typical uses of the FDCTRL macro.

The FDLOAD macro instruction is used to define a specialized FDP that updates or creates data records in the 3735 disk storage file from CPU-generated data when the File Storage capability is present in the system. Such a specialized FDP may consist only of FDFORM, FDLOAD, and FDEND macro statements, and may be used only with data sent to the 3735 from the CPU. For further information, refer to Appendix L for some examples of how to specify FDLOAD macros.

FDCTRL Macro Instruction

The FDCTRL macro instruction:

- Positions the Selectric print element at a user-specified or default location (d).
- Tests the states of the 3735 program logic and feature indicators (IF).
- Alters the states of the 3735 counters (CTR) and indicators (IND).
- Causes the accumulation of batch totals (TOTAL).
- Causes the execution of immediate commands (COMMAND).
- Provides a method of specifying nonsequential processing (GOTO).
- Provides for the repeated execution of sequences of macro statements (CYCLE).
- Provides for backward references to the macro statement (SAVELOC).

FDCTRL macro statements may be used wherever appropriate, because their functions do not specifically relate to form structure. Figure 17 shows the format of the FDCTRL macro instruction.

Name	Operation	Operands
[symbol]	FDCTRL	[d] [,IF = (logterm[, {AND / OR} ,logterm] . . .)] [,CTR = (({d / Xn} [,CLR] [,op,opnd] . . .) [, ({d / Xn} [,CLR] [,op,opnd] . . .)] . . .)] [,IND = ((d, {ON / OFF / INV}) [, (d,X1,compar[,X2])] [,d,X2,compar[,X1])])] [,TOTAL = ((d, 'fid',CTR (d)) [, (d,'fid',CTR (d))] . . .)] [,COMMAND = ((cmndgrp) [, (cmndgrp)] . . .)] [,GOTO = target] [,CYCLE = ([d] [,limit] [,target])] [,SAVELOC = {NO / YES / d}]

Figure 17. Format of the FDCTRL Macro Instruction

symbol

The name entry (symbol) specifies the name of the control statement, if coded. A name entry is required if SAVELOC is coded, or if the macro is an explicit CYCLE limit or target or a branch target (that is, if the macro is referred to elsewhere).

d

The position suboperand (d) specifies the column on the Selectric printer to which the print element is to be moved. The operand may be coded as a decimal number from 1 to 130, but in no case should be less than the value of MRGSTOP+1. If this operand is omitted, a default position of the next sequential column is derived from the preceding macro statement.

Unnecessary motion of the Selectric print element can occur due to the placement of FDCTRL statements in your FDP. Such motion occurs when a branch is made to the FDCTRL statement, as shown in the following example:

```
         FDLINE    50
         FDCTRL    (IF = IND (1) ,OFF) ,GOTO = IND2
         FDFIELD   26,SINK = PRT,SOURCE = 'NOTE ONE'
         FDCTRL    GOTO = NEXT
IND2     FDCTRL    (IF = IND (2) ,OFF) ,GOTO = IND3
         FDFIELD   26,SINK = PRT,SOURCE = 'NOTE 2'
         FDCTRL    GOTO = NEXT
IND3     FDCTRL    (IF = IND (3) ,OFF) ,GOTO = IND4
         FDFIELD   26,SINK = PRT,SOURCE = 'NOTICE THREE'
         FDCTRL    GOTO = NEXT
IND4     FDCTRL    (IF = IND (4) ,OFF) ,GOTO = CONT
         FDFIELD   26,SINK = PRT,SOURCE = 'THANK YOU'
           .
           .
```

If only IND(4) is on, the print element motion is as follows: from line 50, column 1 to column 33 when IND2 is executed, then to column 31 when IND3 is executed, then to column 37 when IND4 is executed, then to column 1, then back to column 26 to print "THANK YOU".

The excessive print element motion occurs because each FDCTRL statement has an implied position based upon the previous FDFORM, FDPAGE, FDLINE, or FDFIELD statement. An FDCTRL statement that follows an FDFORM, FDPAGE, or FDLINE statement has an implied position of column 1. An FDCTRL statement that follows an FDFIELD statement has an implied position one column beyond the end of the field.

You can minimize unnecessary print element motion in two simple ways:

1. Place an FDCTRL statement that is a branch target so that the implied position of the statement is the left margin. Using this technique, the previous example could be coded as follows:

```
         FDLINE    50
         FDCTRL    IF = IND(1) ,GOTO = IND1
         FDCTRL    IF = IND(2) ,GOTO = IND2
           .
           .
IND1     FDFIELD   26,SINK = PRT,SOURCE = 'NOTE ONE'
         FDCTRL    GOTO = NEXT
IND2     FDFIELD   26,SINK = PRT,SOURCE = 'NOTE 2'
         FDCTRL    GOTO = NEXT
           .
           .
```

With this encoding, the print element waits at the left margin of line 50 until an indicator is found on, then moves to column 26 to print the appropriate data.

2. When it is not convenient to code an FDCTRL statement so that its implied position is at the left margin, you can code the position suboperand in the effected FDCTRL statement to force the print element to any desired position. For example, you can reduce the motion by coding the FDCTRL statement as follows:

```
          FDCTRL     GOTO = CTRL2
CTRL1     FDCTRL     1,IF = IND(1) ,GOTO = IND1
                .
                .
```

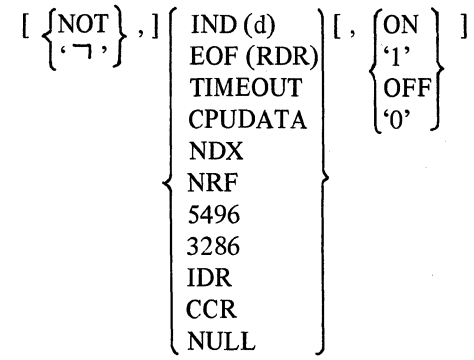
The position of the FDCTRL statement is not column 1 because the position suboperand explicitly defines it to be so.

Figure 18 is a set of charts that will help you decide how to code the other FDCTRL operands. A detailed description of each operand follows the chart in which the operand is introduced.

If you want to. . .	Code. . .	Unless you want to use the default value of . . .
Test one or more logical conditions. . .	IF = (logical expression) where the logical expression specifies the test to be perf performed.	No logical tests.
such as. . .		
To see if indicator 4 is on. . .	IF = IND (4)	*Note:* Logical tests can be combined by the operators AND and OR.
To see if indicator 3 or indicator 4 is on. . .	IF = (IND(3),OR,IND (4))	
To see if the three-minute inquiry timeout indicator has been set.	IF = TIMEOUT	The logical test terms are: IND(d) EOF (RDR) TIMEOUT CPUDATA NDX NRF 5496 IDR CCR NULL
To see if the 5496 card reader has more cards to read. . .	IF = (NOT,EOF (RDR))	
	When the test result is true, the operations symbolized by the remaining FDCTRL operands are performed: when the test result is false, control passes to the next macro statement in the FDP.	
Perform arithmetic operations on one or more 10-digit counters or a 3-digit index counter and set the NDX indicator if the result in zero. . .	CTR = (d,operation[,operand]) where d is the counter number (from 1 to 21) or an index counter specification (X1 or X2); the operation is either CLR (sets CTR to 0), ADD, SUB, MPY, DIV, or DVR; and the operand is either an integer of up to 10 digits or a counter.	No arithmetic operations on counters. *Note:* When X1 or X2 is used, the only operations permitted are to add a signed decimal number, an ordinary counter, or other index counter, to subtract a signed decimal number, or to clear ordinary or index counters.
such as. . .		
Clean counter 2 and add 100 to it. . .	CTR = (2,CLR,ADD,100)	
Multiply the value in counter 3 by the number 4. . .	CTR = (3,MPY,4)	
Clear all 21 counters (special case - use * instead of a number for d). . .	CTR = (*,CLR)	
Add 1 to index counter X1. . .	CTR = (X1,ADD,1)	

Figure 18. Coding the FDCTRL Macro Instruction (Part 1 of 4)

IF = (logterm[,{AND / OR} ,logterm]...) The IF operand directs the 3735 to evaluate the logical expression specified in parentheses. If the result is '1' (TRUE), execution of the activity specified by the remaining FDCTRL operands occurs; if the result is '0' (FALSE), execution of the other operands is suppressed and the next sequential macro statement is processed. The File Storage capability must be present for the CPUDATA, NDX, and NRF indicators to be specified. A combination of File Storage and external numpad must be available for the NULL indicator to be specified. If the IF operand is not coded, all FDCTRL operands are executed unconditionally. The format of logterm as used with the IF operand is as follows:



where IND(d) is the 3735 program logic indicator specified by d, EOF (RDR) is the simulated card reader end-of-file condition (produced by reading a card whose first two positions contain /*), TIMEOUT is the indicator that is set when the 3735 has waited longer than three minutes for a response to a inquiry message and has given control back to the FDP, and CPUDATA is the condition that the 3735 operator is executing the FDP in playback mode with CPU-generated data. When the TIMEOUT indicator is found on, your FDP should notify the operator, then pursue some alternate processing procedure (such as canceling the form). The EOF (RDR) indicator should be tested after every READ (RDR) command to determine when the end of the 5496 card file is reached.

The NDX indicator is set on if the result of an arithmetic operation on an index counter causes the counter to be set to zero. The arithmetic operation CLR does not change the state of the NDX indicator.

The NRF (no record found) indicator is set as a result of READ (file qualifier) and WRITE (file qualifier) commands. If the record requested in a READ command is not found, or if the record written by a WRITE command does not replace (overlay) an existing record in the 3735 disk storage file, the NRF indicator is set on. If the record requested by the READ command is found, or the record written by the WRITE command replaces (overlays) an existing record with the same key in the 3735 disk storage file, the NRF indicator is set off. For READ or WRITE commands that use the save pointer (such as, READ (SAVE), WRITE (SAVE), READ (FILE), or WRITE (FILE) the NRF indicator is set when trying to gain access to data beyond the end of the file.

The logic terms 5496, 3286, and IDR or CCR are used to test indicators that may be set when the 3735 system generation function is preformed. The individual indicators are set on when the corresponding device is included in the 3735 configuration. By testing these indicators and processing different FDP segments depending on the results of the tests, you can write a single FDP that can be used on 3735 terminals with different equipment configurations.

The NULL indicator is set on if the operator fails to enter at least one character from the Selectric keyboard numeric pad area when SOURCE = (KBD, NUMPAD) is in effect. Otherwise, the NULL indicator is set off. The NULL indicator is supplied only with the external keypad feature and applies to all other sources when it is attached.

The logic terms are bit variables that can have values of TRUE = ON = '1' or FALSE = OFF = '0'. Coding a logic term without the prefix NOT or the suffix OFF specifies testing for the TRUE condition of that term. Prefixing of NOT (or '¬') or suffixing

of OFF (or '0') to the term specifies testing for the FALSE condition. The logic terms can be combined by AND (or '&&') and OR (or '1'). Evaluation and combination proceed from left to right through the operand terms, with AND taking precedence over OR. Grouping to force OR to take precedence over AND is not permitted.

The operational suboperands in logterm allow you to code the same test in several different ways. For example, all of the following specifications test for the condition that indicator 1 is ON:

```
IF = IND (1)
IF = (IND (1) ,ON)
IF = (IND (1) ,'1')
IF = (NOT ,IND (1) ,OFF)
IF = (NOT ,IND (1) ,'0')
IF = ('¬' ,IND (1) ,OFF)
IF = ('¬' ,IND (1) ,'0')
```

This variety of permitted specifications lets you choose the one that seems more comfortable.

To set up an FDP to make a delayed decision on which processing path to follow, you can set an indicator in some previous macro statement than test its value in an FDCTRL statement. Suppose that when indicator 10 is on, you want the program to branch to a statement named NEXT, and that when indicator 10 is off you want the program to execute the next sequential macro statement. You can indicate such a test-and-branch instruction by coding:

```
FDCTRL  IF = (IND (10) ,ON) ,GOTO = NEXT
```

When indicator 10 is on, the branch to NEXT is taken; when it is off, the next instruction is executed. See the description of the FDFIELD IND operand and the FDCTRL GOTO operand for discussions of permitted branching techniques.

Note: If the macro statement named by NEXT is a backward reference, then SAVELOC = YES or SAVELOC = d must have been coded in that macro statement for the correct branching code to be generated. Such a backward branch may not imply printer movement to a line that has already been physically passed.

CTR = (({ d / Xn } [,CLR] [,op,opnd] ...) [, ({ d / Xn } [,CLR] [,op,opnd] ...)] ...)

The CTR operand directs the 3735 to perform one or more sequential arithmetic operations on one or more 10-digit counters or on a 3-digit index counter. The operations on a counter are performed from right to left in all cases. The special operation CLR (clear), which may be coded only as the first operation for a given counter, replaces the previous contents of the counter with zero (+0). The other operations (op) are coded as follows (framing apostrophes are required where shown):

ADD or ' + '	The operand is added to the contents of the counter, and the sum replaces the contents of the counter.
SUB or ' - '	The operand is subtracted from the contents of the counter, and the difference replaces the contents of the counter.
MPY or ' * '	The contents of the counter are multiplied by the operand, and the product replaces the contents of the counter.
DIV or ' / '	The contents of the counter are divided by the operand, the remainder is discarded, and the quotient replaces the contents of the counter. The counter is cleared to zero if an attempt is made to divide by zero.
DVR or ' /+ '	The contents of the counter are divided by the operand, the remainder is used to round ("half-adjust") the quotient, and the adjusted quotient replaces the contents of the counter. The counter is clear to zero if an attempt is made to divide by zero.

Only an ADD or SUB operation may be coded immediately after a CLR operation. No overflow indication is provided if the value in a counter exceeds the capacity of the counter ($+10^{10}-1$) or 255 for index counters.

If either of the index counters (X1 or X2) is specified, then the only operations permitted are CLR, ADD a signed decimal number, an ordinary counter, or the other index counter, or SUB (subtract) a signed decimal number. The maximum value permitted in an index counter is 255. The CLR operation does not affect the NDX indicator, but NDX is set on if the result of an ADD or SUB operation is zero.

Note: The File Storage capability must be present before index counters can be specified.

The arithmetic operand (opnd) is either a counter (CTR (d), where d is the number of the counter), or a signed integer or an unsigned interger (considered positive) of one to ten decimal digits. Values of the counter number (d) may range from 1 to 21.

The first counter number specified in a CTR operand may be coded as an asterisk (*), meaning all counters, if it is followed only by the optional operation code CLR. If desired, the CLR specification may be omitted when d is coded as *, in which case the operation is assumed to be CLR. Only the first counter may be coded as an asterisk. The 3735 never clears or otherwise modifies any counter unless specifically instructed to do so.

Note: If only one counter (including all) is to be modified, the inner grouping parentheses may be omitted.

The FDCTRL counter operand (CTR) differs from the FDFIELD CTR operand in that it allows you to perform arithmetic operations on data other than the contents of a field. It also provides you with a convenient way of setting a counter to zero before beginning arithmetic operations. For example, to clear all counters to zero (0), you can code:

```
CTR = (*,CLR)
```

or, more briefly:

```
CTR = *
```

If you want to set a particular counter (counter 20, for example) to a specific value (say, +10), you can code:

```
CTR = (20,CLR,ADD, 10)
```

CLR can be followed only by ADD or SUB, and, once used, CLR may not be reused with the same counter.

To add the contents of one counter to another (for example, add the contents of counter 10 to the contents of counter 21), you can code:

```
CTR = (21,ADD,CTR (10) )
```

You can perform more than one operation on a single counter (and operate on more than one counter) in a single CTR operand. For example, suppose you want to compute a 4% sales tax on some amount that has already been placed in counter 1, then add the tax to the amount and put the result in counter 2. One way to code such a computation is:

```
CTR = ( (21,ADD,CTR (1),MPY,4,DVR,100) , (2,ADD,CTR (1) ,ADD,CTR (21) ) )
```

This example assumes that the counters used for computation have already been cleared by some prevoius instruction.

If you want to. . .	Code. . .	Unless you want to use the default value of. . .
Turn an indicator on or off, or invert it. . . such as. . . To turn indicator 50 on. . . To turn indicator 21 off and indicator 22 on. . . To invert indicator 29 (that is, if it is on, turn it off, and if it is off, turn it on). . . To turn all indicators off (special case: use * instead of a number for d). . .	IND = (d, operation), where d is the indicator number (from 1 to 84), and the operation is ON, OFF, or INV. IND = (50, ON) IND = ((21 , OFF), 22, on)) IND = (29,INV) IND = (*,OFF)	No indicator manipulation.
Compare index counters when File Storage is present. . .	IND = (d ,X {1 2} ,compar [,X {2 1}])	No index counter compare operation.
Accumulate batch totals for data marked by the FDFIELD BATCH operand. . . such as. . . To accumulate totals for data batch 6 of FDP 023 in CTR (2). . .	TOTAL = (d, 'fid', CTR (d)), where d is the number of the data batch, fid is the FDP number in which the batch was created, and CTR (d) identifies the counter in which the total is to be accumulated. TOTAL = (6, '023',CTR (2))	No batch totaling.

Figure 18. Coding the FDCTRL Macro Instruction (Part 2 of 4)

```
IND = ( ( d, { ON  } ) [ , (d,
             { OFF }
             { INV }
   X1,compar[ ,X2 ] ) ]
   [ , (d,X2,compar[ ,X1 ]) ]) ]
```

The IND operand provides for the setting, resetting, and inversion of one or more 3735 program logic indicators. The values of the indicator number (d) may range from 1 to 84. The first indicator number specified in an IND operand may be coded as an asterisk (*), meaning all indicators. Only the first indicator number may be coded in this way. The allowable encodings of the indicator value are as follows:

ON or '1' The specified indicator is set to the bit value '1'.
OFF or '0' The specified indicator is set to the bit value '0'.
INV or '¬' The specified indicator is inverted; that is, if it contains '1' it is set to '0', and if it contains '0' it is set to '1'.

The 3735 resets all indicators to '0' before beginning to process each copy of a form.

Note: When only one indicator (including all) is to be modified, the inner grouping parentheses may be omitted.

The FDCTRL indicator operand (IND) differs from the FDFIELD IND operand in that it allows you to set, reset, or invert indicators unconditionally. (Setting and resetting of indicators in the FDFIELD macro is based on a logical evalution of the field data.) For example, if you want to turn off indicator 10, you could specify an IND operand as:

IND = (10 , OFF)

In addition, this operand provides you with a means of turning all indicators on or off by coding

IND = (*, OFF)

to turn all 84 indicators off, and

IND = (* ,ON)

to turn all indicators on. Remember, though, that the 3735 turns all indicators off at the beginning of each execution of any FDP.

The allowable comparison operators (compar) for comparing index counters are as follows (framing apostrophes are required where shown):

GT or ' > '	Greater than
NG or ' ¬ '	Not greater than
EQ or ' = '	Equal to
NE or '¬= '	Not equal to
LT or ' < '	Less than
NL or '¬<'	Not less than
GE or '>= '	Greater than or equal to
LE or '<= '	Less than or equal to

Index counters may only be used when the File Storage capability is present in the system. When index counters are compared, they may only be compared to each other. Hence, if X1 is specified first in a compare operation, then X2 must be specified second. If there is no second counter specified (that is, X2), X2 is assumed to be the second counter. Likewise, if X2 is specified first in a compare operation and X1 is omitted as the second counter, X1 is assumed to be the second counter.

TOTAL = ((d, 'fid' , CTR (d))
[, (d, 'fid' ,CTR (d))] . . .)

The TOTAL operand causes the 3735 to add to specified counters the values of numeric data fields processed under specified FDPs and flagged (by the FDFIELD BATCH operand) as belonging to particular data batches within the FDPs.

Note: The 3735 does not clear any counters before starting the requested accumulation: See the description of the FDCTRL CTR operand for information on how to clear a counter.

The first d in each suboperand group specifies the number of the batch within the FDP, 'fid' identifies the FDP, and CTR (d) specifies the counter to be used for the accumulation. Values of the BATCH number (d) may range from 1 to 128. Values of the counter number (d) may range from 1 to 21.

Note: When only one total specification is coded, the inner grouping parentheses may be omitted. When both TOTAL and COMMAND are coded in the same FDCTRL statement, TOTAL is performed before COMMAND.

The TOTAL operand can be used to accumulate data items flagged by the FDFIELD BATCH operand. Since the arithmetic operation used in batch totaling is simple addition, you should make sure that the counters you will use to hold the results contain some known value or have been cleared. For example, if you want to accumulate totals for batch 3 of FDP 001 in counter 21, you should first clear the counter, then specify:

TOTAL = (3, '001' , CTR (21))

The resulting total can then be extracted from the counter and sent to the central computer by coding an FDFIELD macro that specifies SOURCE = CTR (21) and SINK = TMT.

COMMAND = ((cmndgrp) [, (cmndgrp)] . . .)

The COMMAND operand requests the 3735 to perform one or more immediate commands in the sequence in which the command groups (and commands within groups, except as noted) are coded. Each command group (cmndgrp) is a collection of commands applicable to the same 3735 device, buffer, or function. The format of each command group is:

command[,command] . . .

where command represents a single operation to be performed by the 3735. Where qualification is shown, such as READ (RDR) or CLEAR (PCH), it may be omitted if the buffer type can be inferred from other commands in the same command group, or if a default is shown. The allowable specifications of command are as follows:

READ[(RDR)]

Causes a card-read operation to be performed on the 5496. The previous 96-character card image in the buffer is completely replaced, and the default value for the lower-numbered character position from which source data is obtained from the card-image buffer is reset to one. If the qualifier is omitted, and if no previous command in the

If you want to. . .	Code. . .	Unless you want to use the default value of. . .
Issue one or more 3735 commands. . .	COMMAND = cmndgrp, where the command group may: • Start I/O operations (READ (RDR, IDR, CCR, or FILE); WRITE (FILE); PURGE; GETKEY; PUNCH; PRINT; SKIP (d); SKIPTO (d); or SEND). • Clear buffers (CLEAR (bufname), where bufname is PCH, STG, INQ, LPB, RPB, IOB, IDR, or CCR). • Disconnect the 3735 from a communications line (DISC). • Stop processing the form (STOP). • Cancel the form (CANCEL).	No 3735 command requests. A complete description of this operand follows the chart.
such as. . . To punch a card on the 5496, then clear the punch buffer. . . To print a line on the 3286, then read a card from the 5496. . . To stop processing the form. . .	COMMAND = (PUNCH, CLEAR) COMMAND = (PRINT, READ (RDR)) COMMAND = STOP	
Cause control to be passed to a macro that is not the next sequential macro statement in the program. . . such as. . . To branch to a macro named LASTLINE. . .	GOTO = target, where target is the name of the macro that has control passed to it. GOTO = LASTLINE	No nonsequential transfer of control. *Note:* A branch that passes control backward can be made only if SAVELOC was coded in the macro being branched to.

Figure 18. Coding the FDCTRL Macro Instruction (Part 3 of 4)

command group has identified a buffer, then READ (RDR) is assumed. This command may also be used for card input to positions 1-96 of RPB. The READ (RDR) command is not exectued in error correct mode or playback mode.

Whenever you use the 5496 as an input device, you should be aware that the 5496 uses a "read-ahead" buffer to help speed up input operations. Thus, when the first READ (RDR) command in the first FDP selected for the day's processing is encountered, two cards are actually read; the first is placed in the 3735 RDR buffer, and the next is placed in the 5496 read-ahead buffer. Succeeding READ (RDR) commands move a card image from the 5496 read-ahead buffer to the 3735 RDR buffer, and the 5496 then loads another card image into its read-ahead buffer. The last card image in the file remains in the 5496 read-ahead buffer until the 3735 operator selects a new FDP. Thus, if successive FDPs are to read successive card files at a single loading of the 5496, each card file should be followed by a /* card and a blank card, and the FDPs should be designed to test for, and dispose of, such blank cards. If this technique is not practical for certain applications, the operator should be instructed to reset the 3735 by turning the power off, then back on, or by turning the Terminal/Manual switch on the 5496 to Manual, then back to Terminal.

READ (IDR)

Causes an Operator Identification Card read operation to be performed by the ID reader attachment. The previous contents of the IDR buffer are replaced and the default value of the lower-numbered character position from which source data is obtained from the ID reader buffer is reset to one. When this command is executed in error correct mode or playback mode, it requires that an ID card be inserted into the ID reader.

READ (CCR)

This command performs the same function as READ (IDR), except that in error correct mode or playback mode no read is performed and the SOURCE = CCR operand gets the previously-read CCR data field from the disk.

The following commands (READ, WRITE, PURGE, and GETKEY) are supported only when the File Storage capability is present in the system.

READ [(file qualifier)]

The file qualifiers for the READ command may be one of the following: FILE, KEY, SAVE, and KEYNOTE. The READ command causes the terminal control program (TCP) to search the 3735 disk storage file and either place a record in the input/output buffer (IOB) with the NRF indicator set off, or set the NRF indicator on if no record is found for a READ (KEY) or READ (KEYNOTE) or set NRF on if the file save pointer points to the group separator character for a READ (SAVE) or READ (FILE). If the qualifier KEY is coded, a key followed by a record sepatator (SINK = IOB with DELIMIT) must reside in the I/O buffer before the issue of the Read command. The record read will be the one matching the key found in the I/O buffer.

If the qualifier is KEYNOTE, the READ functions like the READ (KEY) command except that the position of the next consecutive record is noted and saved by a file save pointer.

If the qualifier is SAVE, the record read is the one pointed to by the file save pointer set by some previous READ or WRITE command that noted the next consecutive record. The file save pointer is not updated.

If the qualifier is FILE or the qualifier is omitted but FILE is implied by other FILE or IOB commands in the same command group, the read operation is performed like the READ (SAVE) command except that the position of the next consecutive record is noted as is done by the KEYNOTE qualifier.

Before KEY or KEYNOTE operations, the I/O buffer must be loaded with a key followed by a record separator X'14'. For an explanation of the format in the buffer,

refer to the SINK = IOB DELIMIT operand description. Before SAVE or FILE operations, a KEYNOTE or FILE operation should be performed to set the file save pointer.

Note: Use of the qualifiers KEY or SAVE does not modify the file save pointer. Therefore, commands using these qualifiers may be intermixed with those using the KEYNOTE and FILE qualifiers without loss of the consecutive pointer. Also, the file save pointer is not initialized at the start of an FDP. Hence, if SAVE or FILE qualifies are used, they should be preceded by a command specifying KEYNOTE. Refer also to the description of the GETKEY command below.

WRITE[(file qualifier)]

The file qualifiers for the WRITE command may be one of the following: FILE, KEY, SAVE, KEYNOTE, and KEYLAST. The WRITE command causes the terminal control program (TCP) to write the contents of the input/output buffer (IOB) to the 3735 disk storage file and if the file qualifier was KEY, KEYNOTE, or KEYLAST either set the NRF indicator on if no record is overlayed, or overlay a record in the I/O buffer (IOB) with the NFR indicator set off. The write command also resets the default starting position to one for I/O buffer source and sink specifications. If the qualifier KEY is coded, a key followed by a record separator and data followed by a record separator (SINK = IOB with DELIMIT) must reside in the I/O buffer.

If the qualifier is KEYNOTE, the WRITE functions like the WRITE (KEY) command except that the position of the next consecutive record is noted and saved by the same file save pointer as used by READ (KEYNOTE) and READ (FILE).

If the qualifier is SAVE, the record is written over the one pointed to by the file save pointer set by some previous READ or WRITE command that noted the next consecutive record. The file save pointer is not updated. If the file save pointer points to the group seperator when the write is executed than the record is added to the file and the NRF indicator is set on, otherwise the NRF indicator is set off.

If the qualifier is FILE or the qualifier is omitted but FILE is implied by other FILE or IOB commands in the same command group, the write operation is performed like the WRITE (SAVE) command except that the position of the next consecutive record is noted as is done by the KEYNOTE qualifier.

The qualifier KEYLAST functions like the KEY qualifier except that it directs the 3735 to write the end-of-group indicator after writing the current record. This effectively purges any records that follow. For an explanation of the file format, refer to the description of the SINK = IOB operand specification.

Before KEY, KEYNOTE, or KEYLAST operations, the I/O buffer should be loaded with a key followed by a record separator X'14' and data followed by a record separator. For an explanation of the format of the buffer, refer to the SINK = IOB, DELIMIT operand description.

Note 1: Use of the qualifier KEY or SAVE does not modify the save pointer. Therefore, commands using these qualifiers may be intermixed with those using the KEYNOTE and FILE qualifiers without loss of the consecutive pointer. Also, the save pointer is not initialized at the start of the FDP. Hence, if SAVE or FILE qualifiers are used, they should be preceded by a command specifying KEYNOTE. Refer also to the description of the GETKEY command below.

Note 2: When updating the 3735 disk storage file, you should make sure that a new record with the same key as an existing record is no longer or shorter than the existing record in the file. If you should replace a record in the file that is X bytes long with one that is greater than X bytes long, the next record following the original X bytes can no longer be found because part or all of its key has been destroyed. This also occurs in the case of a short record.

Note 3: Records are added to the file at the end following all existing records. Thus, during the processing of a file, any records that have been added do not appear in key sequence within the file.

PURGE[(FILE)]

Causes the terminal control program (TCP) to clear the 3735 disk storage file. The TCP does this by writing a group separator character in the first data byte in the file. For an explanation of the file format, refer to the description of the SINK = IOB operand specification.

GETKEY ('key')

Functions like a READ (KEYNOTE) command expect that the specified key is automatically placed into the IOB by macro generated code. The key specified must be the same length as the key of the desired record on the file and must have trailing blanks or leading zeros if they are needed. This command may be used to position the file for subsequent read operations by a READ[(FILE)] command to process a file or portion of a file consecutively from the starting key.

PUNCH

Causes a card-punch operation to be performed on the 5496 card punch. The contents of the card image buffer are not modified, but the default value for the lower-numbered character position into which field data is put in the card-image buffer is reset to 1. This command may also be used for card output from positions 97-192 of RPB. The PUNCH command is executed in all operating modes.

CLEAR ({ PCH | STG | INQ | LPB | RPB | IDR | CCR })

Causes the 3735 to set the contents of the card-image punch buffer (PCH), the storage buffer (STG), the inquiry buffer (INQ), the line printer buffer (LPB), the read/punch buffer (RPB), the Identification Reader buffer (IDR), or the Credit Card Reader buffer (CCR) to all blanks (SP characters). The default value for the lower-numbered source or sink character position is reset to 1. The qualifier (LPB) can be omitted when CLEAR is coded in conjunction with SKIP (d), SKIPTO (d), or PRINT, since the meaning can be inferred from the context. The CLEAR (PCH) and CLEAR (RPB) specifications are mutually exclusive. For IDR and CCR, the only sequences permitted are (CLEAR (bufname)) , (READ (bufname)) and (CLEAR (bufname) ,READ). The CLEAR (STG) and CLEAR (INQ) commands are not executed in error correct mode or playback mode; all other CLEAR commands are executed in all operating modes.

PRINT

Directs the 3735 to print the current contents of the line printer buffer (LPB) on the 3286 printer. Printing begins with the first character position and continues through the highest-numbered character position modified since the last CLEAR (LPB) operation (up to a 132-character maximum). Unmodified higher positions are not put to the 3286, and appear on the print line as blanks. The contents of the line printer buffer are not modified, but the default value for the lower-numbered character position into which field data is put in the LPB is reset to 1. After each PRINT command, a SKIP (d) or SKIPTO (d) command should be coded. If none is, SKIP (1) is used by default. The PRINT command is executed in all operating modes.

Note: PRINT must be coded before CLEAR if both appear in the same command group. In this combined operation, the 3735 always prints first and then clears the line printer buffer (LPB). The effect of the prohibited sequence (CLEAR,PRINT) can be achieved by coding COMMAND = (CLEAR, SKIP (1)) or COMMAND = (SKIP (1) , CLEAR).

SKIP (d)

Specifies that d 3286 form lines are to be skipped before any other instructions are executed for the 3286. Values of d may be specified from 1 to 16383, but you must not code a value that forces skipping beyond the end of the form. The SKIP (d) command is executed in all operating modes.

Note: You must evaluate the effects of SKIP (d), and of the skipping implied by PRINT, to make sure that the 3286 does not attempt to print or skip beyond the end of the form.

SKIPTO (d)

Specifies that the 3286 form should be advanced to line d before any other instructions are executed for the 3286. Note that this line number is specified in relation to the beginning of the form, rather than in relation to the beginning of the current page (as it is for the Selectric printer). Values of d may be specified from 1 to 16383, but you must not code a value that forces skipping beyond the end of the form. In addition, you should not attempt any backward movement to a previously passed line. The SKIPTO (d) command is executed in all operating modes. The largest value of d coded in a SKIPTO (d) command within a given FDP is assumed to be the number of the last physical line on the 3286 form. Before coding an FDEND macro to end the form processing, you should code an FDCTRL macro that causes the 3286 to skip to the end of the current form (unless the FDP prints on the last line of the form). For example, if the last line of the form is line 66, you should code:

```
FDCTRL COMMAND = SKIPTO (66)
```

Note: In cases where an FDP segment can be reached by alternate paths that result in different numbers of output lines on the 3286, an FDCTRL COMMAND = SKIPTO (d) should be coded in that segment to reestablish a specific line position on the 3286 form. This situation can occur when the 3286 is operated within a cycle. The use of SKIPTO (d) is not allowed within a cycle.

SEND

Causes the 3735 to attempt to send the current contents of the inquiry buffer to the central computer with which it is connected. When SEND is issued at the 3735, the Terminal Control Program (TCP) inserts a three-byte message header (NUL I NUL) in the first three positions of the inquiry buffer.

On a multipoint (leased) line, no terminal operator intervention is needed. On a dial (switched) line, the 3735 suspends processing of the FDP until the terminal operator either makes a connection with a central computer or cancels the processing of the current record (that is, the current copy of the form). Once the operator has made the connection, no further manual intervention is needed until after a response to the inquiry is received at the 3735.

Once a communication link is established, the 3735 sends the message and turns off the TIMEOUT indicator. When the 3735 receives a response, or if a three-minute timeout occurs, FDP processing resumes where it was stopped by the SEND command. Additional inquiry messages may be sent to the central computer (and responses received by the 3735) until the switched-line connection is broken by issuing a DISC command.

If no response is received within three minutes, the TIMEOUT indicator is turned on. This indicator can be tested by coding an FDCTRL IF = TIMEOUT instruction. The SEND command is not executed during error correct mode or playback mode.

DISC

Causes the 3735 to break the switched-line connection with the central computer. This command is not normally issued until the central computer has sent its responses to all inquiry messages initiated by a given FDP. The DISC command is not executed during error correct mode or playback mode.

Note: You need not issue a DISC command for 3735 terminals that communicate over leased (nonswitched) lines, since the line connection is never broken. In this environment, the DISC command is treated as a no-operation (NO-OP) command.

STOP

Causes the 3735 to stop processing the record (form) in progress and to advance to the next copy of the form, if any. Any data entered up to this point is saved by the 3735 control program.

CANCEL

Causes the 3735 to discard the record (form) in progress and to advance to the next copy of the form, if any. Any data entered up to this point is ignored, and is not saved by the 3735 control program.

Note: When single-copy hand-fed forms are being used, one effect of the STOP or CANCEL command is to eject the current copy of the form from the Selectric printer. COMMAND = STOP and COMMAND = CANCEL are mutually exclusive. When either one is specified, it should be the last command coded in an FDCTRL macro. Neither one is permitted if a GOTO operand is coded.

A single COMMAND operand can be coded to perform several functions. Each group of commands must apply to the same 3735 device, buffer, or function, and a command group must be enclosed in parentheses if it consists of more than one command. For example, if you want to read an input data card from the 5496, punch an output data card on the 5496, and send an inquiry message, all you need to code (once the proper buffers are filled by other macro statements) is:

```
COMMAND = (READ (RDR) ,PUNCH , SEND)
```

Or, if you want to print an output line on the 3286 printer, clear the line printer buffer for further output, then read a card from the 5496, you can code:

```
COMMAND = ( (PRINT,CLEAR), READ (RDR) )
```

GOTO = target

The GOTO operand alters the terminal's normal processing sequence by causing control to be transferred to the macro statement named by target. The transfer of control (branch) is delayed until the functions of all other FDCTRL operands have been performed, if any were coded.

The transfer of control may not be specified to imply reverse tabulation, reverse line feed, or reverse form feed, since these functions are not available in the 3735. (However, a backward branch to a macro statement in which SAVELOC = YES or SAVELOC = d was coded is permitted if the restrictions described above are not violated.) In addition, the target macro statement must not be within the range of repetition of a CYCLE operand unless the FDCTRL statement containing the GOTO operand is itself within the range of repetition of the same CYCLE operand. If the GOTO is within a CYCLE and the target is the macro that began the cycle, then the cycle continues without altering the cycle count. If the GOTO is within a CYCLE and the target begins a different cycle, the new cycle is initiated. You can never code a GOTO that branches into a cycle from outside the cycle.

Note: An unconditional GOTO should not be specified in an FDCTRL macro statement that is a CYCLE limit. In such a macro statement, therefore, the IF operand must be coded.

The GOTO operand, when combined with the logical testing capability of the IF operand, provides you with the ability to decide what kind of processing will be performed when your FDP is used at a 3735. An example of such a conditional branch, using IF and GOTO, is part of the discussion of the IF operand following Figure 18 (Part 1).

In addition, the GOTO operand may be used to perform unconditional branches by coding:

FDCTRL GOTO = macname

where macname is the name of the macro statement to which control is being passed.

CYCLE = ([d] [, limit] [,target])

The CYCLE operand specifies a cycle count (d), delimits a group of sequential macro statements that are to be processed repeatedly as a unit (limit), and specifies where sequential processing is to resume after the cyclic processing ends (target). The coding of this operand is explained in the discussion of the FDLINE macro instruction.

SAVELOC = { NO / YES / d }

The SAVELOC operand directs the FD macros to save the location of the current macro statement within the form description, for the purpose of resolving backward references to the statement.

For example, suppose that you want to scan through a card file that consists of both header and detail items, and that you want to process only the header cards. If a header card is identified by the digit 1 in the first card column, the section of code that scans the file might be as follows:

```
SCAN      FDCTRL   COMMAND = READ,SAVELOC = 1
          FDCTRL   IF = EOF (RDR), GOTO = ENDFILE
          FDFIELD  SOURCE = (RDR), 1, 1) , IND = (1, EQ, 1) ,SINK = NULL
          FDCTRL   IF = (NOT , IND (1) ) , GOTO = SCAN
PROCESS   [ process the header card as desired. ]
```

The coding of the SAVELOC operand is explained in the discussion of the FDPAGE macro instruction. Note that this coding technique does not limit the size of the file (as CYCLE would do).

If you want to. . .	Code. . .	Unless you want to use the default value of. . .
Repeat the execution of a group of macro statements. . .	CYCLE = (d,limit,target), where: • d is a number from 1 to 16383 that specifies the maximum number of time the cycle is to operate. • limit is the name entry of the last macro in the repeated group. • target is the name entry of the macro to be executed when the cycle is finished.	No repeated execution of macro statements. When CYCLE is coded, at least one of the suboperands should be coded. The default values are described following Figure 12 (Part 1).
and. . . Define a summary block to be executed at the end of the cycle. . .	SKIP (d) as the linenum operand of a target FDLINE macro statement.	No summary block. See the discussion following Figure 12 (Part 2) for more details
Save the location of the macro so that you can make backward references to it. . .	SAVELOC = YES, if you want to save the location for the entire form, or SAVELOC = d, where d is the number of backward references you will make to this statement.	SAVELOC = NO, which does not save the statement location. *Note:* If SAVELOC is coded YES or d, a name entry is required in the FDCTRL statement.

Figure 18. Coding the FDCTRL Macro Instruction (Part 4 of 4)

FDLOAD Macro Instruction

The FDLOAD macro instruction allows you to write an FDP that can use CPU-generated data to load or update the 3735 disk storage file when the File Storage capability is present. (File storage updating during normal document preparation is handled by the appropriate WRITE (file qualifier) commands.) The only macro statements that may appear in a file load FDP are FDFORM, FDLOAD, and FDEND. Conversely, such a specially-designed FDP may not be used with any other data except that received from the CPU. Figure 18A shows the format of the FDLOAD macro instruction.

Name	Operation	Operands
[symbol]	FDLOAD	KEYLEN = d ,DATALEN = d[,ENDCHAR = 'string']

Figure 18A. Format of the FDLOAD Macro Instruction

The FDLOAD macro describes records that are transmitted from the CPU to the 3735 to be written in the user's data area of the 3735 disk. FDPs consisting of FDLOAD macros must be executed at the terminal in playback mode. If an FDP is not executed in playback mode, an error message is generated and the FDP is canceled. Each FDLOAD macro coded in an FDP generates a loop of code that reads CPU data records and writes them on the disk until a record with a key identical to that specified by an ENDCHAR operand is detected. When this key is found, the next FDLOAD-generated loop gains control for processing if more records follow. If no more records follow, the FDP ends.

symbol

The name entry (symbol) specifies the name of the macro statement, if coded. It may be used to indicate the file name, but is not used by the macros.

KEYLEN = d

The KEYLEN operand specifies the length of the key field for each CPU-generated file storage record. The value specified may range from 1 to 15. If FDLOAD is coded, this operand must be coded.

DATALEN = d

The DATALEN operand specifies the length of the data field for each CPU-generated file storage record. The specified value (d) must be a number from 1 to 233 such that the sum of the key length (KEYLEN) and data length (DATALEN) values does not exceed 234. If FDLOAD is coded, this operand must be coded.

ENDCHAR = 'string'

The ENDCHAR operand indicates the key of the last CPU-generated record in the record group. When the FDP executes with appropriately prepared CPU data, each record, except the last for each FDLOAD macro, is written on the 3735 disk storage file. The key field of the last record in the group must be the same as the character string in the ENDCHAR operand, or the implicit default. If the ENDCHAR operand is not coded, the default is a key field of all asterisks (*). If the ENDCHAR operand is coded, the operand must specify a character string (within apostrophes) that is the same as the key field of the final (dummy) record. The last record for each FDLOAD macro should always be a dummy record because the last record is not written on the 3735 disk storage file. For an ENDCHAR string that is longer than the keys specified by the KEYLEN operand, the string is truncated beginning from the right. If the string is short, it is padded to the right with blanks until its length is equal to that in the KEYLEN operand.

Note: The MODE operand of FDFORM must specify MODE = LOAD when an FDP containing FDLOAD statements is assembled.

The data used to update the 3735 disk file storage consists of a three-digit FDP identifier field, followed by some file storage record groups. The FDP identifier field identifies the form ID (FID) of the FDP that is to be used to load the file. Each file storage record group corresponds to, and must be defined by, an FDLOAD macro with

KEYLEN, DATALEN, and ENDCHAR operands coded in the corresponding FDP. Each file storage record group consists of at least one record, and each record consists of a key field followed by a data field. The length of the key field and data field are those specified in the mandatory KEYLEN and DATALEN operands of the corresponding FDLOAD macro. Refer to Appendix L for further explanation of using the FDLOAD macro to load or update a file from CPU data.

Delimiting Form Description Macro Instruction

The delimiting macro FDEND marks the close of each form description. Only one FDEND macro statement may appear in an FDP.

FDEND Macro Instruction

The FDEND macro instruction marks the end of a group of macro statements that describe a single form. You must code FDEND only once, at the end of the macro statements for each FDP, since it indicates to the Assembler that there are no more source statements for this FDP. It also completes code generation and prepares the FD macros to process a possible following FDP. Figure 19 shows the format of the FDEND macro instruction.

Name	Operation	Operands
[symbol]	FDEND	[,] (This macro has no operands.)

Figure 19. Format of the FDEND Macro Instruction

symbol

The name entry (symbol) specifies the name of the macro statement, if coded. If this macro is an explicit CYCLE target or a GOTO target, a name entry is required.

You can code the FDEND macro simply as:

```
FDEND
```

The FDEND macro statement has no operands. It may be the target statement named by a GOTO operand (see the discussion of the FDCTRL macro) or by a CYCLE operand, but it may not be the limit statement named by a CYCLE operand. If a comment is placed in an FDEND statement, a comma should be placed in the operand field to indicate that no operands appear.

Diagnostic Form Description Macro Instruction

The diagnostic macro instruction may be used to aid in debugging FDPs. It may be coded at any point in an FDP without effecting form structure or function.

FDSYNTAX Macro Instruction

The FDSYNTAX macro instruction, which has no operands, may be coded before or between any of the other FD macros. Any macro statements following the first FDSYNTAX macro are not expanded; however, their operands are checked for syntactic integrity by the Assembler. A subsequent FDSYNTAX macro statement causes the macros that follow it to be expanded normally. The syntax checking performed consists only of those checks made by the Assembler on macro source statements, such as checks for unbalanced parentheses, invalid continuation, and undefined keyword parameters. The format of the FDSYNTAX macro is the same as that of the delimiting macro FDEND.

Segments and Paths in a Form Description Program

The output listing from an FDP assembly contains messages that divide the FDP into paths and segments. This information about paths and segments can help the forms encoder to verify the correctness of the FDP and to trace the sequence of actions should the FDP produce unexpected results when used at the 3735.

A new path is created by (1) the start of the FDP, (2) by the coding of a SAVELOC operand outside a CYCLE, or (3) by the joining of two paths. The segments of each path are numbered from 1 upward. Each segment consists of a block of code that is always executed as a unit. At the end of the segment, control is passed either to another path or to a higher-numbered segment in the same path. The first segment of a path is always executed if the path is entered. The remaining segments are executed conditionally or unconditionally, depending on the flow of control through the path. A path terminates (1) with the start of another path, (2) with the end of the FDP, or (3) if an unconditional STOP or CANCEL command is encountered.

The segments that make up a path are created by (1) the start of a path, (2) a branch instruction (for example, a GOTO), (3) the creation of a CYCLE, (4) the coding of a SAVELOC within a CYCLE, (5) the macro statement following a CYCLE limit macro, or (6) the joining of one or more segments in the path.

At the end of each path, the actions performed on the indicators, buffers, devices, and counters that were used in the path are summarized in detail through MNOTE messages. Warnings describing possible erroneous use of these storage areas and devices are listed when the potential error condition occurs (for example, moving data out of a buffer without earlier reference to it in the path), or at the end of the path (for example, moving data into a buffer and failing to make further use of the data before the path ends). Messages describing the transfer of control between paths and segments are listed when the branches are resolved.

Before going on to the discussion of assembly considerations, look at the sample form shown in Figure 20. The check marks identify fields that should be stored for transmission to the central computer. Figure 21 shows one way that the FD macros could be coded to process this form at a 3735. Note that when a default is used, a comment entry specifies what the default value should be. This sample program uses cyclic processing, but does not require other types of conditional processing. Another sample program, found in Appendix C, illustrates conditional branches using the FDCTRL IF and GOTO operands.

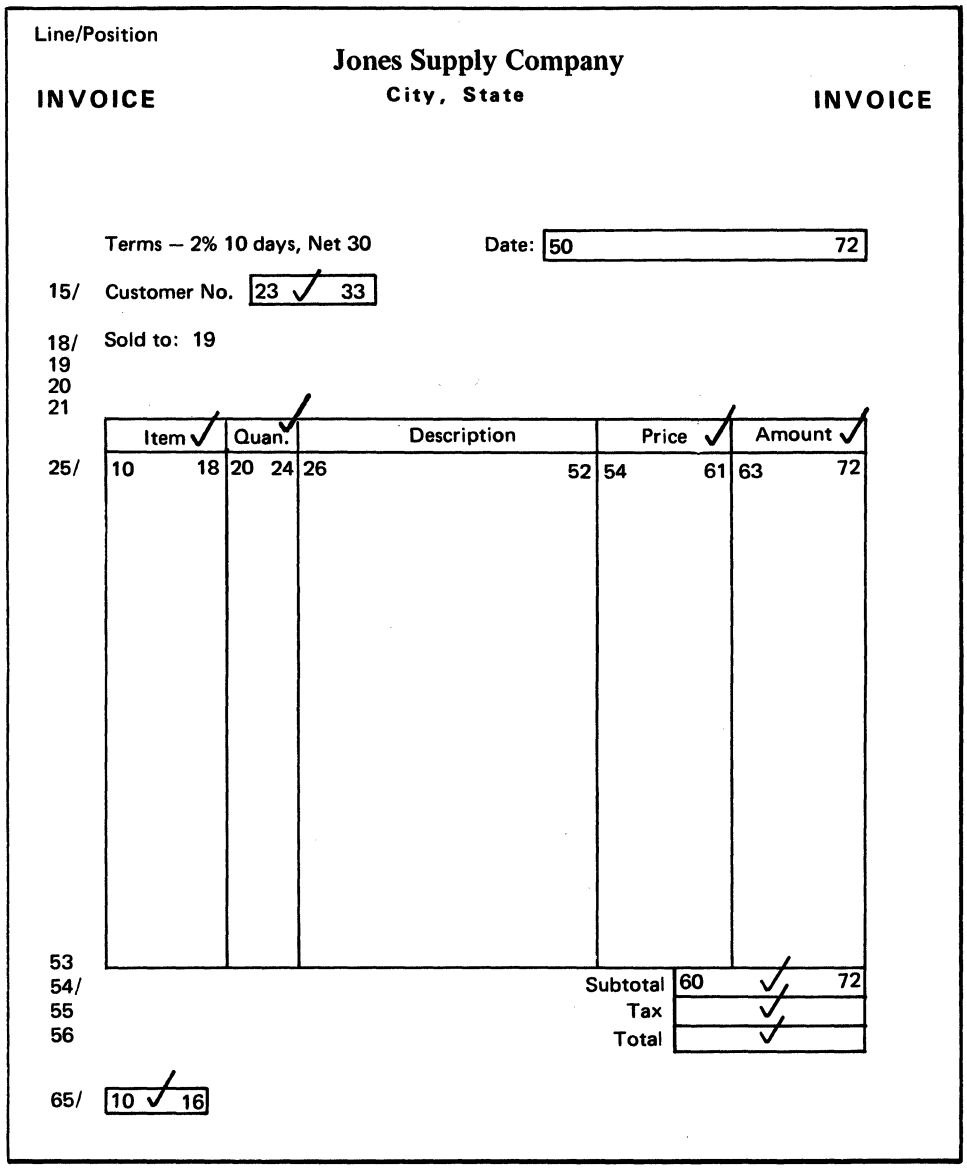


Figure 20. Jones Supply Company Invoice

```
FORM001  FDFORM   FID='001',              FORM IDENTIFIER IS 001          X
               PACKING=DELIMIT,          REDUCES TRANSMITTED DATA        X
               MRGSTOP=9,                SET MARGIN STOP                 X
               MESSAGE='JONES SUPPLY CO. - FORM 1895 - MRGSTOP=9 - VERSX
               ION 1 - 03/31/72',                                        X
               HTAB=53                   TAB OUT OF DESCRIPTION FIELD
*
         FDCTRL   CTR=((2,CLR),(3,CLR)) CLEAR COUNTERS 2 AND 3
*
         FDPAGE   1,HEIGHT=66,           ONE STANDARD HEIGHT PAGE        X
               VMRG=(15,65),             TEST VERTICAL MARGINS AT 15 & 65X
               WIDTH=85,                 STANDARD WIDTH LINES            X
               HMRG=(10,72),             TEST HORIZ. MARGINS AT 10 AND 72X
               SOURCE=(KBD,OPTIONAL),    INPUT FROM KEYBOARD             X
               SINK=PRT                  OUTPUT TO SELECTRIC
*
         FDLINE   ,                      DEFAULTS TO LINE 15
CUSTNO   FDFIELD  23,33,                 SPACE FOR CUSTOMER NUMBER       X
               SINK=(,TMT)               PRINT AND TRANSMIT OUTPUT
*
DATE     FDFIELD  50,72                  SPACE FOR DATE -- NO TRANSMIT
*
SOLDTO   FDLINE   18                     4 LINES FOR CUST. NAME & ADDR.
NAME     FDFIELD  19,72                  SPACE FOR CUSTOMER NAME
ADDR1    FDLINE   ,                      DEFAULTS TO LINE 19
         FDFIELD  19,72                  SPACE FOR ADDRESS LINE
ADDR2    FDLINE   ,                      DEFAULTS TO LINE 20
         FDFIELD  19,72                  SPACE FOR ADDRESS LINE
ADDR3    FDLINE   ,                      DEFAULTS TO LINE 21
         FDFIELD  19,72                  SPACE FOR ADDRESS LINE
*
*        BEGIN BODY OF FORM -- USES CYCLIC PROCESSING
*
         FDLINE   25,                    BEGIN ON LINE 25                X
               SINK=(,TMT),              TRANSMIT DATA FOR EACH LINE     X
               CYCLE=(28,GROSS,TOTALS)   CYCLE UP TO 28 LINES
*
ITEM     FDFIELD  10,18,                 SPACE FOR ITEM NUMBER           X
               SOURCE=(,,NUMPAD),        FROM NUMERIC KEYS ON KEYBOARD   X
               JUSTIFY=R                 RIGHT-JUSTIFY SELECTRIC SINK
*
QUAN      FDFIELD  20,24,                SPACE FOR QUANTITY ORDERED      X
               SOURCE=(,,NUMPAD),        FROM NUMERIC KEYS ON KEYBOARD   X
               SINK=(,,CTR(1)),          TO SELECTRIC, COMPUTER, & CTR(1)X
               JUSTIFY=R                 RIGHT-JUSTIFY SELECTRIC SINK
*

DESC     FDFIELD  26,51,                 SPACE FOR DESCRIPTION OF ITEM   X
               SINK=(,NULL)              PRINT ONLY -- NO TRANSMIT
```

Figure 21. A Form Description Program for the Jones Supply Co. Invoice (Part 1 of 2)

```
PRICE    FDFIELD  53,61,                  SPACE FOR PRICE EACH            X
               CTR=(1,MPY,FIELD),        MULTIPLY BY QUAN ORDERED        X
               PICTURE='$*,**9.99'       PERMITS AMOUNTS TO $9,999.99
*
GROSS    FDFIELD  63,72,                  FOR PRICE TIMES QUAN EXTENSION  X
               SOURCE=CTR(1),            GET AMOUNT FROM COUNTER         X
               CTR=(2,ADD,FIELD),        ACCUMULATE GROSS AMOUNTS IN CTR X
               PICTURE='$**,**9.99'      PERMITS AMOUNTS TO $99,999.99
*
*        THIS IS THE END OF CYCLIC PROCESSING
*
TOTALS   FDLINE   54,                     BEGIN TOTALS CALCULATIONS       X
               SINK=(,TMT)               TRANSMIT DATA FOR LINE
ENDCYCLE FDFIELD  ,                       FOR END CYCLE INDICATION        X
               SOURCE='***',             END CYCLE INDICATOR             X
               SINK=(NULL,TMT)           NO PRINT - TRANSMIT ONLY
         FDFIELD  60,72,                  SPACE FOR SUBTOTAL              X
               SOURCE=CTR(2),            GET ACCUMULATIONS FROM COUNTER 2X
               PICTURE='$*,***,**9.99'   PERMITS AMOUNTS TO $9,999,999.99
*
*        THIS FDCTRL MACRO COMPUTES 4% TAX AND ADDS IT TO THE SUBTOTAL
TAXCOMP  FDCTRL   CTR=((3,ADD,CTR(2),MPY,4,DVR,100),(2,ADD,CTR(3)))
*
TAX      FDLINE   55,                     FOR DISPLAYING TAX ON SELECTRIC
               SINK=(,TMT)               TRANSMIT DATA FOR LINE
         FDFIELD  60,72,                  SPACE FOR TAX AMOUNT            X
               SOURCE=CTR(3),            GET TAX AMOUNT FROM COUNTER 3   X
               PICTURE='$*,***,**9.99'   PERMITS AMOUNTS TO $9,999,999.99
*
GRANDTOT FDLINE   56,                     LINE FOR GRAND TOTAL            X
               SINK=(,TMT)               TRANSMIT DATA FOR LINE
         FDFIELD  60,72,                  SPACE FOR GRAND TOTAL           X
               SOURCE=CTR(2),            GET TOTAL FROM COUNTER 2        X
               BATCH=1,                  FLAG FOR BATCH TOTALS           X
               PICTURE='$*,***,**9.99'   PERMITS AMOUNTS TO $9,999,999.99
*
THANKS   FDLINE   60                      LINE FOR MESSAGE ON INVOICE
         FDFIELD  10,72,                  USE ENTIRE LINE                 X
               SOURCE='THANK YOU FOR YOUR ORDER -- PLEASE CALL AGAIN', X
               JUSTIFY=C                 FOR SELECTRIC ONLY
*
NUMBERS  FDLINE   65                      LINE FOR FID/RSN DATA
         FDFIELD  10,12,SOURCE=FID        FID NUMBER HERE
         FDFIELD  13,13,SOURCE='/'        SLASH GOES HERE
         FDFIELD  14,16,SOURCE=RSN,       RECORD SEQUENCE NUMBER HERE     X
               SINK=(,TMT)               TRANSMIT DATA FOR FIELD
*
         FDEND     ,                      END OF INVOICE
```

Figure 21. A Form Description Program for the Jones Supply Co. Invoice (Part 2 of 2)

Assembling the Form Description Macro Instruction

To assemble the Form Description (FD) macro instructions, you need an operating system (either OS or DOS) with these facilities:

- An Assembler.
- A macro library containing the FD macro definitions.
- Enough auxiliary storage to contain the results of Assembler processing.

You also need a description of the job control statements that are required to use the Assembler and related facilities, including the auxiliary storage media that are available for retaining the Assembler object-module output (when you do not want to retain it in card form).

When you examine the assembly listing, you must verify the correctness of the results and eliminate all observed errors before you submit the Assembler output to the FD utility. The general flow of the assembly process is shown in Figure 22.

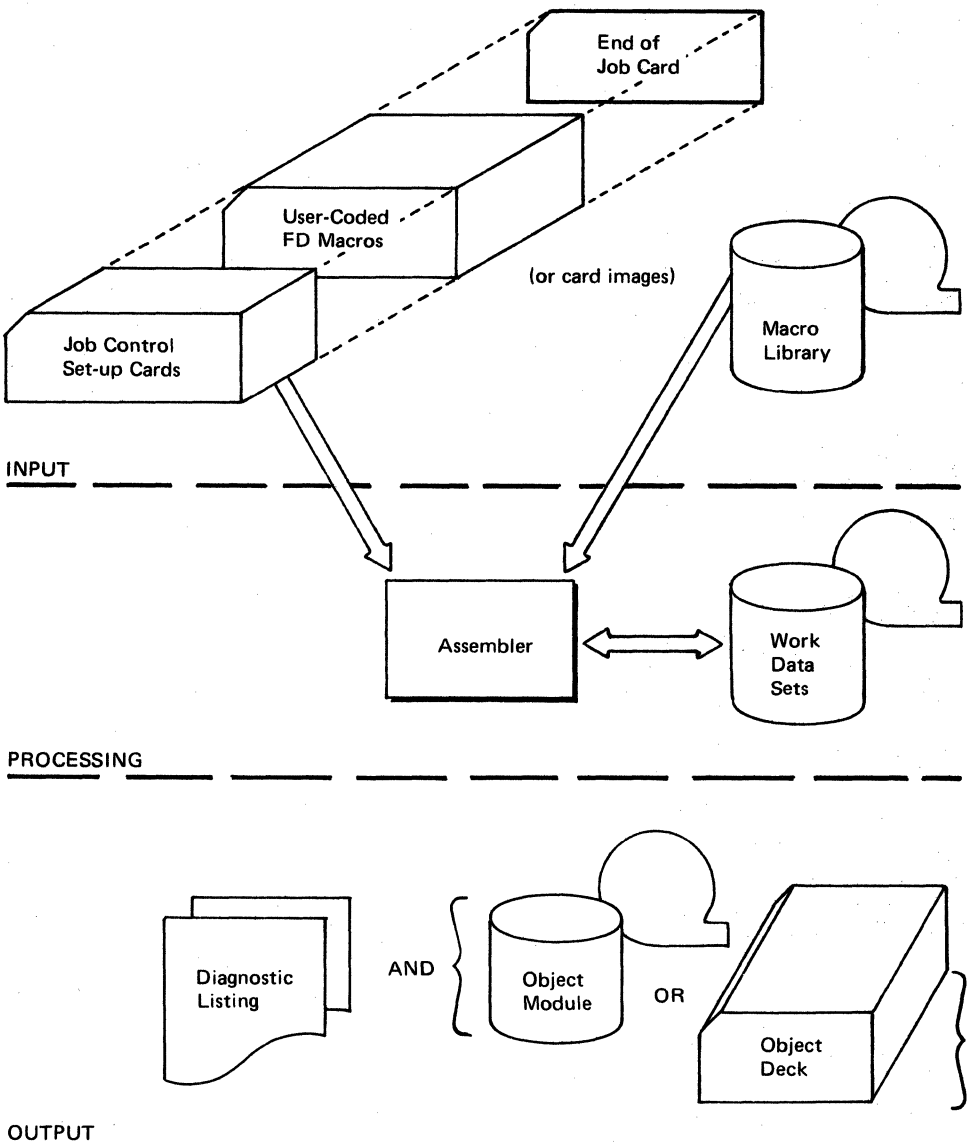


Figure 22. Flow of Control and Data Through the Assembler

The code generated during the FDP assembly consists primarily of unpacked hexadecimal strings that represent FDP instructions and data. The unpacking portion of the code generation processing creates FDP strings as follows:

1. The character representation is translated to 3735 internal code. For example, the number 123 is translated to X'313233'.
2. The hexadecimal string is then padded by alternating the digits 4 and 7 to create the unpacked FDP strings. For example, the coded string X'313233' appears in the Assembler listing, after unpacking, as X'437143724373'.

Appendix D describes the various MNOTE messages that may be issued during the FD macro assembly. The Assembler also issues other kinds of messages; these are described in the Assembler Language publications referred to in the OS and DOS discussions that follow.

Although assembly procedures for the Operating System and Disk Operating System are similar, the following sections discuss each system separately, so that the OS user need not be concerned with DOS requirements, and *vice versa.*

Operating System (OS) Assembly Considerations

In most installations, you can use an OS cataloged procedure to assemble your source statements. The installation's system programmer is responsible for defining the necessary Job Control Language (JCL) statements so that the cataloged procedure specifies the proper macro library and proper output data sets.

A general example of such a cataloged procedure is:

```
//jobname JOB  (other required parameters)
//stepname     EXEC ASMFC
//ASM.SYSIN    DD   *
                .
                .
        (source program statements)
                .
                .
/*                    (end-of-data-set delimiter)
//                    (end-of-job statement)
```

The job control statements generated by the invocation of this cataloged procedure in a typical installation might be:

```
XXASM          EXEC PGM=IEUASM,REGION=50K
XXSYSLIB       DD   DSNAME=SYS1.MACLIB,DISP=SHR
XXSYSUT1       DD   DSNAME=&SYSUT1,UNIT=SYSSQ,SPACE=(1700,(400,50)),
XX                  SEP=(SYSLIB)
XXSYSUT2       DD   DSNAME=&SYSUT2,UNIT=SYSSQ,SPACE=(1700,(400,50))
XXSYSUT3       DD   DSNAME=&SYSUT3,SPACE=(1700,(400,50)),
XX                  UNIT=(SYSSQ,SEP=(SYSUT2,SYSUT1,SYSLIB))
XXSYSPRINT     DD   SYSOUT=A
XXSYSPUNCH     DD   SYSOUT=B
```

Note: The SYSPUNCH statement names the data set that is to contain the object module produced by the Assembler. SYSOUT = B directs the output to the card punch. If you do not want card output, you should specify that the output be placed in some other data set.

The object modules that result from the assembly begin with a CSECT named IDF1000, and continue through IDF1001, IDF1002, and so forth, as required. You should carefully check all MNOTE messages issued during the FD macro assembly. Many of these messages are informational; they simply describe the values that were coded in the FD macros. You should examine these messages to make sure that the values you coded were properly interpreted. Other messages indicate possible errors and definite errors. You should examine these messages to determine the cause of each error, then correct the indicated errors (if necessary) before assembling the FDP again. Once you have an error-free assembly, you can proceed to the next stage in preparing your programs for use at the 3735 terminal, executing the FD utility.

For further information concerning the OS Assembler and its processing, refer to the *OS Assembler Language* publication, Order No. GC28-6514, and the *OS Assembler (F) Programmer's Guide,* Order No. GC26-3756. The latter publication describes the various messages (other than MNOTE indications) that may be issued during Assembler processing.

Disk Operating System (DOS) Assembly Considerations

The Disk Operating System (DOS) job control statements required to execute the Assembler are, to some extent, configuration-dependent. The installation system programmer is responsible for determining the device addresses that you should use in your job control assignment (ASSGN) statements to indicate the input/output devices required by the Assembler. The job control statements required are shown below in the order in which they must be coded, where "cuu" denotes that the actual device address must be coded in the job control statements (for example, "00E", "192", or "280"). Many DOS installations have standard device-address assignments for particular devices. Check with your computing center for details.

```
// JOB jobname (other optional parameters)
// ASSGN SYSIPT,X'cuu'    (source program input)
// ASSGN SYSLST,X'cuu'    (program listing)
// ASSGN SYS001,X'cuu'
// ASSGN SYS002,X'cuu'    (work files)
// ASSGN SYS003,X'cuu'
// ASSGN SYSPCH,X'cuu'    (required when DECK option is specified)
// OPTION DECK,...        (optional -- indicates Assembler functions)
// EXEC ASSEMBLY          (required)
                .
                .
          (source program statements)
                .
                .
/*                        (end-of-data-set delimiter)
/&                        (end-of-job statement)
```

The object modules that result from the assembly begin with a CSECT named IJLF1000, and continue through IJLF1001, IJLF1002, and so forth, as required. You should carefully check all MNOTE messages issued during the FD macro assembly. Many of these messages are informational; they simply describe the values that were coded in the FD macros. You should examine these messages to make sure that the values you coded were properly interpreted. Other messages indicate possible errors and definite errors. You should examine these messages to determine the cause of each error, then correct the indicated errors (if necessary) before assembling the FDP again. Once you have an error-free assembly, you can proceed to the next stage in preparing your programs for use at the 3735 terminal, executing the FD utility.

For further information concerning the DOS Assembler and its processing, refer to the *Disk and Tape Operating Systems Assembler Language* publication, Order No. GC24-3414.

Before the FDPs generated by the Form Description (FD) macro assembly can be used at the 3735 terminal, they must be processed by the Form Description (FD) utility. When the utility processing is completed, the resulting FDPs are in a format that the 3735 terminal control program (TCP) can interpret. You can then use your own teleprocessing application programs to transmit the FDPs to the 3735. Figure 23 shows the general flow of control and data through the FD utility.

The following paragraphs describe FD utility processing. Although the utility functions are similar for both OS and DOS, many operations and techniques are not. The discussion is organized so that a user of either system needs to read only those sections that pertain to his specific system. Thus the OS user need not concern himself with DOS considerations, and *vice versa.*

While performing its processing functions, the FD utility generates various informational and diagnostic messages. These messages are fully described in Appendix E (for OS systems) and Appendix F (for DOS systems).

What the Form Description Utility Does

The FD utility is divided into three distinct processing steps: *control, link-edit,* and *storage.*

- The *control* step obtains the card-image output of the FD macro assembly from an input data set (either a card reader or some other sequential input device), and generates the control statements that the link-edit step needs to produce the program that is executed in the storage step.
- The *link-edit* step obtains the output of the control step and combines it with an IBM-supplied module that contains the executable code for the storage step.
- The *storage* step places the FDP blocks in a user-specified data set. When the storage step has completed normal processing, you can transmit the FDPs to the 3735 terminal.

Certain errors detected by the FD utility during execution may cause processing to terminate abnormally, depending on the severity of the error. The program response to errors arising from improper input (cards missing or out of sequence) is to write a message in the diagnostic listing produced by the control step and terminate processing. The input data following the item in error is not processed.

Program response to errors such as insufficient space in a data set is to terminate processing and note the condition in the diagnostic listing produced by the storage step.

Using the OS Form Description Utility

The OS Form Description utility is executed as a sequence of job steps scheduled through OS job control. Before the utility can be executed, it must be made known to the Operating System either by placing it in the system library (SYS1.LINKLIB) or by placing it in a private library and using an appropriate JOBLIB or STEPLIB DD statement.

You can execute the OS FD utility by invoking an OS cataloged procedure. The installation's system programmer is responsible for defining JCL statements so that the cataloged procedure specifies the proper input and output data sets (CTRL . SYSIN and STG . SYSLIB). The cataloged procedure UTIL3735 may be invoked by the following JCL:

```
//jobname JOB (other required and optional parameters)
//STEP    EXEC UTIL3735
//CTRL.SYSIN   DD (* if card input, dsname if not card input)
/*                 (needed only for card input)
//STG.SYSLIB   DD (dsname of file to hold FDPs)
//
```

The CTRL . SYSIN data set should contain the object modules created by the Assembler. These modules may be in 80-column cards in the input stream, or in card images on some auxiliary storage device. The STG . SYSLIB data set should provide enough space to hold the output of the storage step. It should be defined as a partitioned data set (PDS). Each member placed in this PDS by the storage step is a separate FDP.

The job control statements generated by the invocation of this cataloged procedure are as follows:

```
XXCTRL     EXEC PGM=IDFCT
XXSYSPRINT DD   SYSOUT=A,DCB=(RECFM=FB,LRECL=120,BLKSIZE=3600)
XXSYSUT1   DD   DSNAME=&SYSLIN,UNIT=SYSDA,DISP=(NEW,PASS),
XX              SPACE=(TRK,(7,3))
XXLKED     EXEC PGM=IEWL,PARM='LIST,OVLY,XREF,MAP,DC',REGION=96K,
XX              COND=(0,LT,CTRL)
XXSYSLIN   DD   DSNAME=&SYSLIN,DISP=(OLD,DELETE)
XXSYSUT1   DD   UNIT=SYSDA,SPACE=(1024,(50,20,1))
XXSYSPRINT DD   SYSOUT=A
XXSYSDD    DD   DSNAME=SYS1.LINKLIB,DISP=SHR
XXSYSLMOD  DD   DSNAME=&GOSET(GO),UNIT=SYSDA,DISP=(MOD,PASS),
XX              SPACE=(1024,(50,20,2))
XXSTG      EXEC PGM=*.LKED.SYSLMOD,COND=((0,LT,LKED),(0,LT,CTRL))
XXSTEPLIB  DD   DSNAME=*.LKED.SYSDD,DISP=SHR
XXSYSPRINT DD   SYSOUT=A,DCB=(RECFM=FB,LRECL=120,BLKSIZE=3600)
```

Much of the DCB information for the STG . SYSLIB data set is provided by a DCB macro instruction in the module IDFST. This DCB information is described later under "OS Storage Step Operations".

If no condition codes are specified in EXEC statements, the job is terminated if any job step returns a nonzero completion code. You can, if you wish, use the REPLACE option in the PARM field of the EXEC statement for the STG job step to replace any existing FDP in your data set that has the same name as any of the FDPs you are currently processing with the utility. Coding PARM . STG = 'REPLACE' requests that all FDPs with duplicate names be replaced with new ones. Coding PARM . STG = 'REPLACE = (name1, name 2, . . . , name 20)' requests that up to 20 specified names be replaced if they are found to be duplicates.

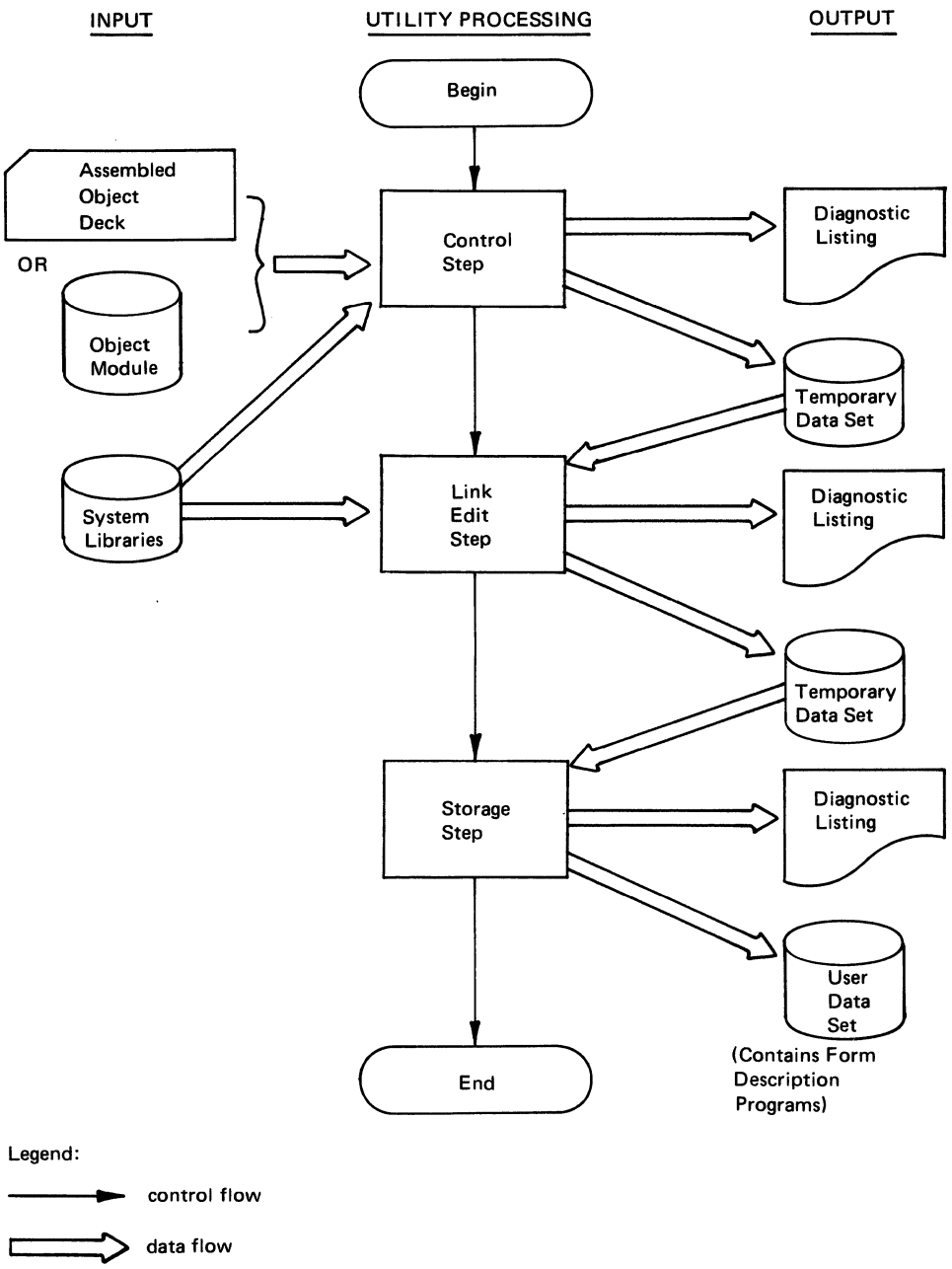


Figure 23. Flow of Control and Data Through the Form Description Utility

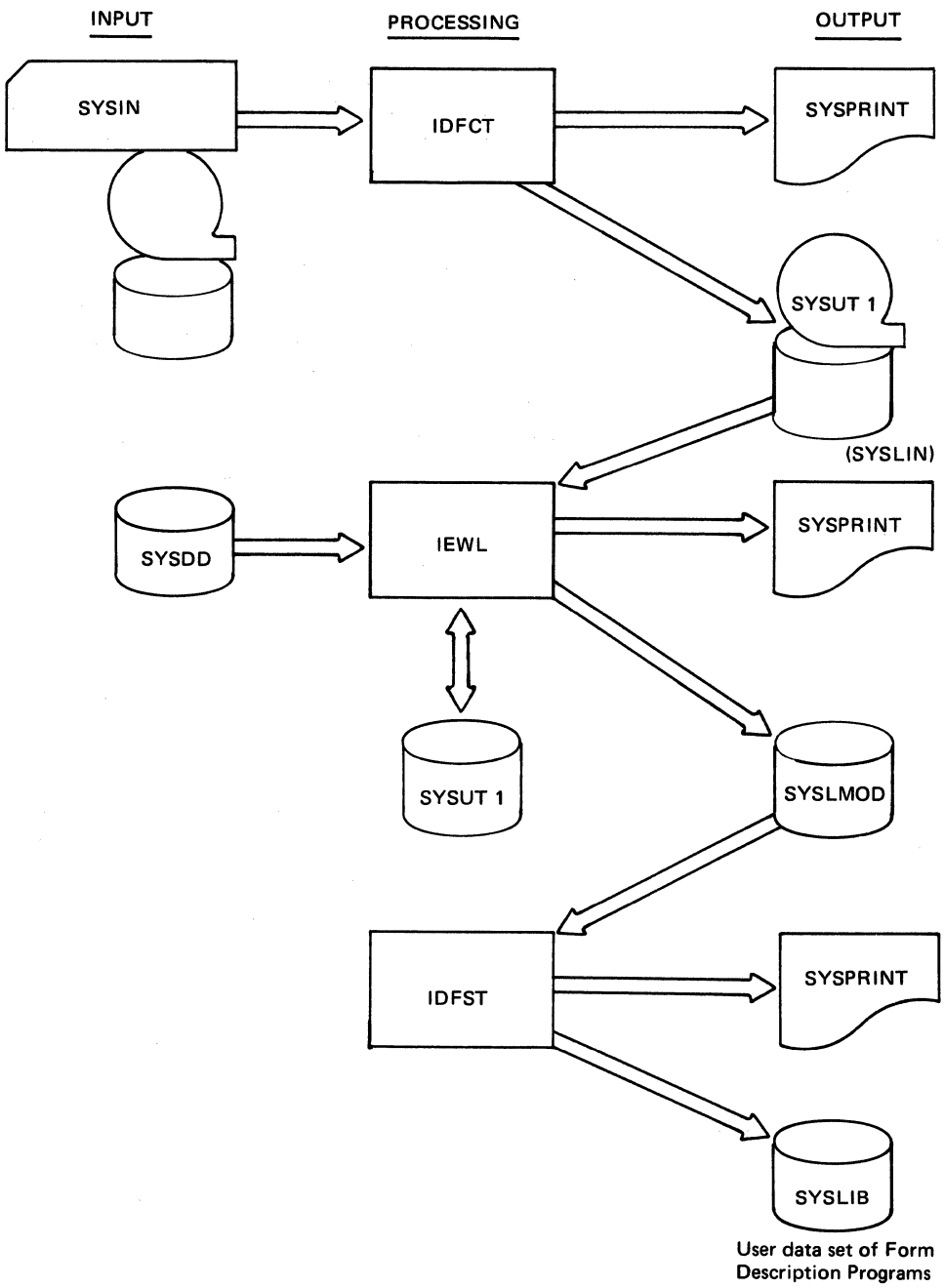


Figure 24. Data Flow Through the OS Form Description Utility

If an FDP in your data set has the same name as a new FDP to be added, and the utility has not been instructed to replace the old FDP, it attempts to store the new FDP with a temporary name from the series IDFTEMP0, IDFTEMP1, . . . , IDFTEMP9. If these temporary names have been exhausted, the new FDP is not stored. The action taken is noted in the diagnostic listing produced by the storage step.

OS Control Step Operations

The OS control step (CTRL) uses three data sets during its execution:

- SYSIN, which contains the output data from the FD macro assembly.
- SYSUT1, which contains the output data from this step – one or more object modules. each followed by its associated Linkage Editor control statements.
- SYSPRINT, which provides the programmer with a diagnostic listing describing the results of step processing.

The SYSIN data set contains one or more object modules with a variable number of CSECTs. Each CSECT is composed of six 486-byte blocks. (The last CSECT may have from one to six such blocks.) Each block in a CSECT contains an 8-byte name, a 2-byte key, and a 476-byte data sector that contains the macro-generated object code. The 8-byte name is the name of the FDP as specified in the name field of the FDFORM macro instruction. This is the name by which the FDP is stored in the user's data set. The 2-byte key begins with X'0000' in the first block for each FDP, and increases by one for each succeeding block of the FDP.

The OS control step examines these input CSECTs, card image by card image, and creates output modules that have Linkage Editor control statements inserted at the end of each module. The output is placed in the SYSUT1 data set in the format shown in Figure 25. The Linkage Editor control statements are generated when the OS control step examines the External Symbol Dictionary (ESD) card images in the SYSIN data set. All card images placed in the SYSUT1 data set have been checked for correct sequence number and deck identification, and all ESD card images have been checked for proper content.

The SYSPRINT data set contains diagnostic information of interest to the programmer. It is normally directed to the system printer, and consists of one message line for each error detected by the control step. The messages that may appear in the listing are described in Appendix E.

OS Link-Edit Step Operations

The OS Link-edit step (LKED) uses five data sets during its execution:

- SYSLIN, which contains the output from the OS control step (the same data set as CTRL . SYSUT1). This is the primary input to the Linkage Editor.
- SYSDD, which is a data set containing the IBM-supplied load module that is combined with the primary input data set (SYSLIN) to form the program that is executed in the OS storage step.
- SYSUT1, which is a work file used by the Linkage Editor.
- SYSLMOD, which contains the output load module to be executed in the OS storage step.
- SYSPRINT, which provides the programmer with a diagnostic listing describing the results of step processing.

The SYSLIN data set should contain the sequence of records shown in Figure 25. The control statements generated by the control step describe the Linkage Editor processing required to create the overlay program that is executed in the storage step.

The SYSDD data set contains the IBM-supplied module IDFST, which is the executable code for the OS storage step. This module stores the new FDPs in the user's library of FDPs and provides a diagnostic listing of processing results. When the IDFST module is link-edited with the primary input data set (SYSLIN), it becomes the root segment in an overlay program produced by the Linkage Editor.

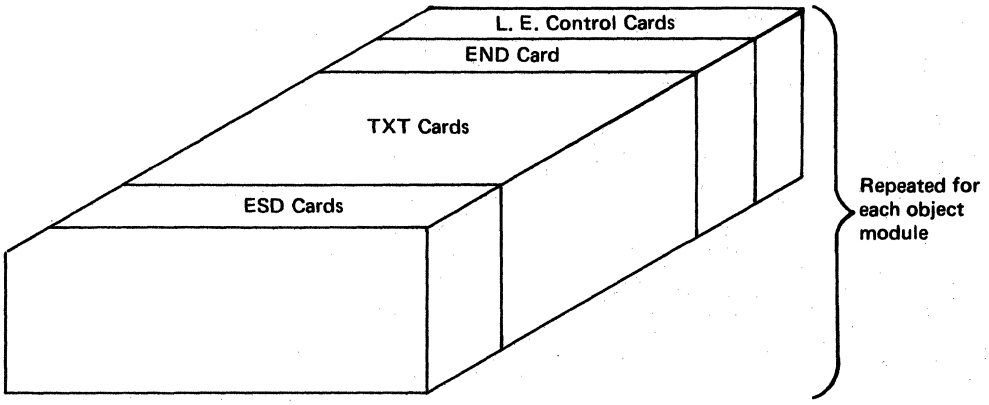


Figure 25. OS Control Step Output Format

The SYSUT1 data set is a work file that is used to contain intermediate results of Linkage Editor processing.

The SYSLMOD data set contains the executable overlay program produced by the Linkage Editor.

The SYSPRINT data set contains the diagnostic messages produced during Linkage Editor processing. The Linkage Editor messages that may be directed to this data set are described in Appendix E.

Note that this step of the OS FD utility can use the smallest Linkage Editor available in the system: that is, 15K bytes (Level E) or 44K bytes (Level F).

Further information describing the operation of the Linkage Editor can be found in the *OS Linkage Editor and Loader* publication, Order No. GC28-6538.

OS Storage Step Operations

The OS storage step (STG) uses two data sets during its execution:

- SYSLIB, which defines the user library (partitioned data set) in which the new FDPs are to be stored.
- SYSPRINT, which provides the programmer with a diagnostic listing describing the results of step processing.

The program executed in this step is created by the Linkage Editor in the previous step and placed in the SYSLMOD data set. Its function is to place the new FDPs in the SYSLIB user library, so that the user can later transmit any or all of them to the 3735 terminal. Only the 476-byte data sectors in each CSECT are placed in the user library, and each FDP stored is identified by the 8-byte name that was specified in the name field of the FDFORM macro statement for that FDP. This name becomes a member name in the SYSLIB partitioned data set.

The output data control block (DCB) describes the characteristics of the SYSLIB data set as follows:

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

The application program that retrieves the FDPs from this data set should refer to a particular FDP by its form name (the name entry that was coded in the FDFORM macro) via the Basic Partitioned Access Method (BPAM) macros, and should simply extract and transmit all of the 476-byte data blocks associated with that form name (member name). The application program can be written to obtain the member names for the data set directory, or from some input medium (such as the SYSIN input stream or PARM fields on an EXEC statement).

The SYSPRINT data set contains diagnostic information of interest to the programmer. It is normally directed to the system printer, and may consist of one or more message lines for each execution of the storage step. The messages that may appear in the listing are described in Appendix E.

When the FD utility has completed normal processing, the FDPs are ready to be sent to a 3735.

Using The DOS Form Description Utility

The DOS Form Description utility is executed as a sequence of job steps scheduled through DOS job control. Before the utility can be executed, it must be made known to the system by cataloging it in a relocatable library and a core-image library.

Before the DOS FD utility is executed, you must ensure that the core image library (or private core image library) contains enough unused space to catalog all the program blocks that will be created by the Link Edit step. The required space can be determined by assuming that each CSECT used as input to the Control step occupies one block in the core image library. Then add 25 blocks for the program root phase. For example, if your FDPs total 10 CSECTs, then 35 blocks of core image library space will be needed. Since the FD utility operates as a "link", load, and go" sequence of job steps, this space is freed after the Storage step is finished.

The job control statements that you use to execute the utility should be similar to those shown below, modified to suit your installation requirements. (These job control statements assume that you are using 9-track tape work files.) To create the ISAM file:

```
// JOB  jobname (other JOB statement parameters)
// ASSGN  SYSIPT,X'cuu'[1]
// ASSGN  SYSPCH,X'cuu'[2]
// ASSGN  SYSLST,X'cuu'
// EXEC  IJLFCT
// CLOSE  SYSPCH,X'cuu'[3]
// PAUSE  RELOAD OUTPUT TAPE
// RESET  SYSIPT
// RESET  SYSPCH
// ASSGN  SYSIPT,X'cuu'[4]
// OPTION LINK
   INCLUDE[5]
// EXEC  LNKEDT
// PAUSE  VALIDATE OUTPUT
// RESET  SYSIPT
// ASSGN  SYS000,X'cuu'[6]
// DLBL  IJFDLIB,...,ISC[7] [8]
// EXTENT SYS000,serialnumber,4,1,...[8]
// EXTENT SYS000,serialnumber,1,2,...[8]
// EXTENT SYS000,serialnumber,2,3,...[8]
// EXEC
// OPTION=LOAD
// DEVICE={2311}[9]
          {2314}
```

```
/*
// PAUSE  CHECK SYSLST FOR MSG 4F24I
// DLBL   IJFDLIB,...,ISE[7] [8]
// EXTENT SYS000,serialnumber,4,1,...[8]
// EXTENT SYS000,serialnumber,1,2,...[8]
// EXTENT SYS000,serialnumber,2,3,...[8]
// EXEC
// OPTION=LOADFST
// DEVICE={2311}[9]
          {2314}
/*
/&
```

Notes:

1 Supplies the object modules created by the Assembler.

2 Separate unlabeled work tape drive.

3 Restores normal SYSPCH assignment.

4 Same device as prior SYSPCH.

5 IJLFST, IJLFLOAD, and IJLFUPDT must be in user's relocatable library. When 'ATTN cuu' prints on the system console, respond with an EOB.

6 User's indexed sequential data set.

7 IJFDLIB is a fixed file name.

8 Optional if STDLABEL or if PARSTD was used to store label previously.

9 Device must be the same DASD device for LOAD and LOADFST options.

To update or add to the ISAM file:

```
// JOB  jobname
// ASSGN  SYSIPT,X'cuu'[1]
// ASSGN  SYSPCH,X'cuu'[2]
// ASSGN  SYSLST,X'cuu'
// EXEC  IJLFCT
// CLOSE  SYSPCH,X'cuu'[3]
// PAUSE  RELOAD OUTPUT TAPE
// RESET  SYSIPT
// RESET  SYSPCH
// ASSGN  SYSIPT,X'cuu'[4]
// OPTION LINK
   INCLUDE[5]
// EXEC  LNKEDT
// PAUSE  VALIDATE OUTPUT
// RESET  SYSIPT
// ASSGN  SYS000,X'cuu'[6]
// DLBL  IJFDLIB,...,ISE[7] [8]
// EXTENT SYS000,serialnumber,4,1,...[8]
// EXTENT SYS000,serialnumber,1,2,...[8]
// EXTENT SYS000,serialnumber,2,3,...[8]
// EXEC
// OPTION=UPDATE
// DEVICE={2311}
          {2314}
// RPLACE or // RPLACE=modname[9]
/*
/&
```

Notes:

1 Supplies the object modules created by the Assembler.
2 Separate unlabeled work tape drive.
3 Restores normal SYSPCH assignment.
4 Same device as prior SYSPCH.
5 IJLFST, IJLFLOAD, and IJLFUPDT must be in user's library. When 'ATTN cuu' prints on the system console, respond with an EOB.
6 User's indexed sequential data set.
7 IJFDLIB is a fixed file name.
8 Optional if STDLABEL or if PARSTD was used to store label previously.
9 // RPLACE replaces all duplicate modules; // RPLACE = modname replaces only the module named.

If the DOS FD utility processes an FDP with the same name as one that already exists in the user's indexed sequential data set, the utility attempts to store the new FDP with a temporary key (from the series IJLFTM00 to IJLFTM09) unless the RPLACE statement is used to indicate some other action. The RPLACE statement may be coded in two ways:

// RPLACE = modname or // RPLACE

In the first case, the FDP identified by modname is replaced. Up to 20 // RPLACE = modname statements are permitted. In the second case, all duplicate FDPs found are to be replaced with new ones.

Figure 26 shows the data flow through the DOS FD utility.

DOS Control Step Operations

The DOS control step uses three data sets during its execution:

- SYSIPT, which contains the output data from the FD macro assembly.
- SYSPCH, which contains the output data from this step – Linkage Editor control statements for each object module, followed by the associated module.
- SYSLST, which provides the programmer with a diagnostic listing describing the results of step processing.

The SYSIPT data set contains one or more object modules with a variable number of CSECTs. Each CSECT is composed of three 486-byte blocks. (The last CSECT may have from one to three such blocks.) Each block in a CSECT contains an 8-byte name, a 2-byte key, and a 476-byte data sector that contains the macro-generated object code. The 8-byte name is the name of the FDP as specified in the name field of the FDFORM macro instruction. This is the name by which the FDP is stored in the user's indexed sequential data set. The 2-byte key begins with X'0000' in the first block for each FDP, and increases by one for each succeeding block of the FDP.

The DOS control step examines these input CSECTs, card image by card image, and creates output modules that have Linkage Editor control statements inserted at the beginning of each module. The output is placed in the SYSPCH data set in the format shown in Figure 27. The Linkage Editor control statements are generated when the DOS control step examines the External Symbol Dictionary (ESD) card images in the SYSIPT data set. All card images placed in the SYSPCH data set have been checked for correct sequence number and deck identification, and all ESD card images have been checked for proper content.

The SYSLST data set contains diagnostic information of interest to the programmer. The messages that may appear in the listing are described in Appendix F.

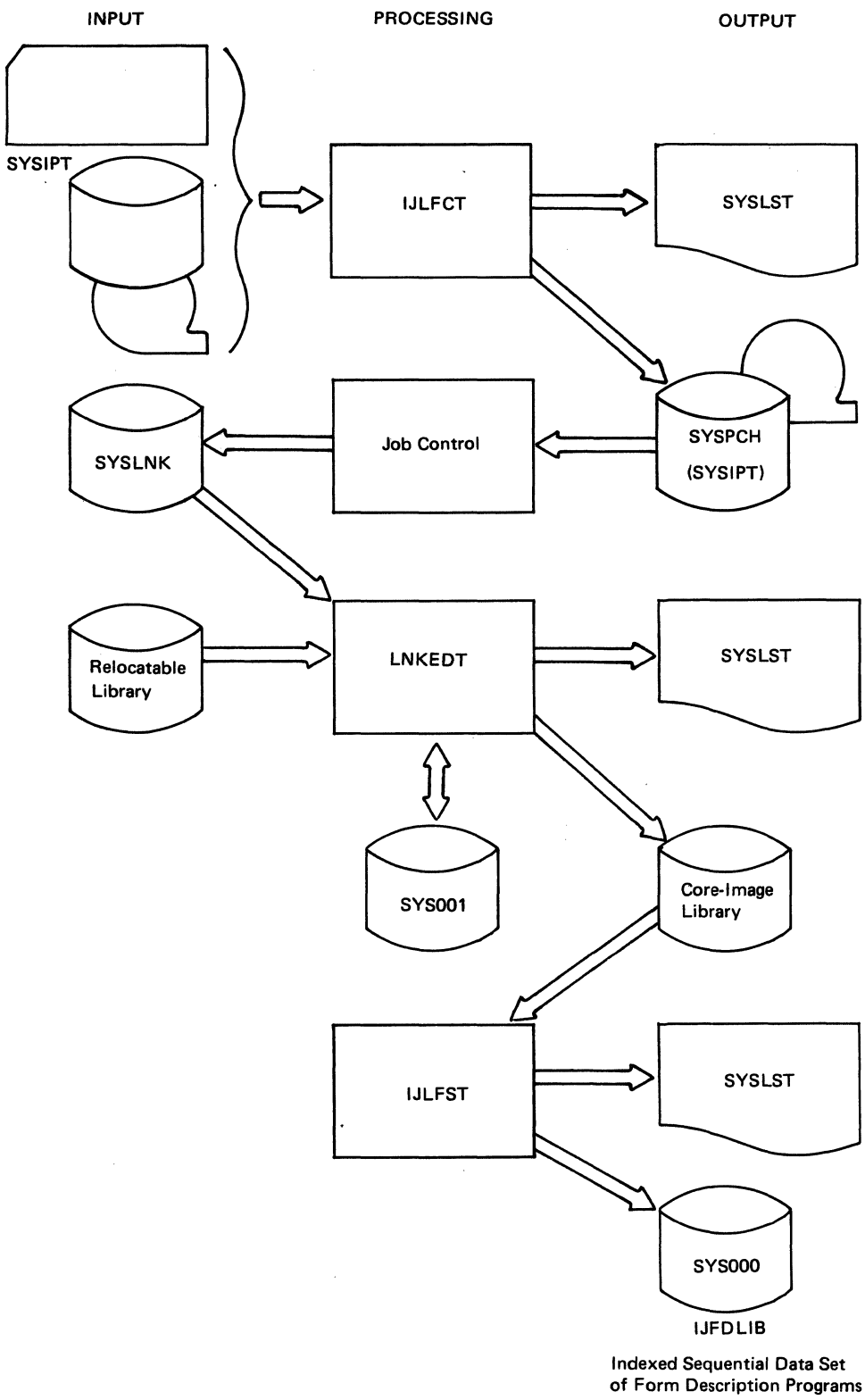


Figure 26. Data Flow Through the DOS Form Description Utility

DOS Link-Edit Step Operations

The DOS link-edit step uses five data sets during its execution:

- SYSLNK, into which Job Control has placed the output from the DOS control step (the same data set as the control step data set SYSPCH). This is the primary input to the Linkage Editor.
- Three modules included from a relocatable library. These IBM-supplied relocatable modules (IJLFST, IJLFLOAD, and IJLFUPDT) are combined with the primary input data set (SYSLNK) to form the program that is executed in the DOS storage step.
- SYSLST, which provides the programmer with a diagnostic listing describing the results of step processing.
- SYS001, which provides a work file extent to be used by the Linkage Editor.
- Core image library space, which is used to contain the overlay program segments as they are created.

The SYSLNK data set should contain the sequence of records shown in Figure 27. Linkage Editor control statements generated by the control step describe the Linkage Editor processing required to create the overlay program that is executed in the storage step.

The relocatable library contains the IBM-supplied modules IJLFST, IJLFLOAD, and IJLFUPDT, which are the executable code modules for the DOS storage step. These modules store the new FDPs in the user's indexed sequential data set and provide a diagnostic listing of processing results. The IJLFST, IJLFLOAD, and IJLFUPDT modules are link-edited with the primary input data set (SYSLNK).

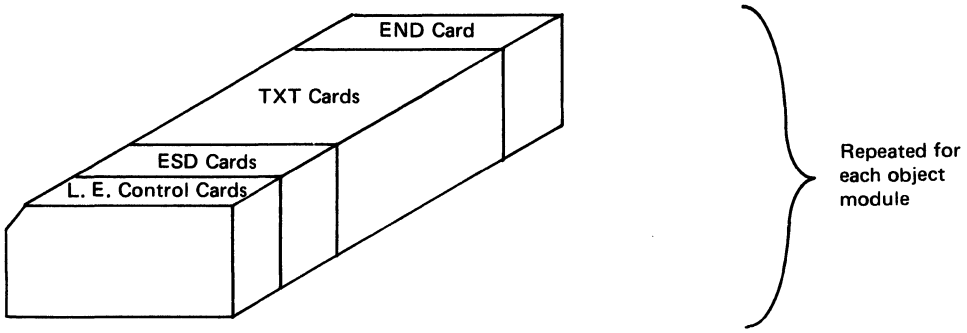


Figure 27. DOS Control Step Output Format

The SYSLST data set contains diagnostic information of interest to the programmer. The Linkage Editor messages that may appear in the listing are described in Appendix F.

The SYS001 data set is a work file that is used to contain intermediate results of Linkage Editor processing.

Core image library space is needed to contain the overlay program phases as they are created. Normally, these phases are not cataloged.

Further information describing the operation of the Linkage Editor can be found in the *DOS System Control and System Service Programs* publication, Order No. GC24-5036.

DOS Storage Step Operations

The DOS storage step uses four data sets during its execution:

- SYS000, which defines the extent for the user's indexed sequential (ISAM) data set (IJFDLIB) in which the new FDPs are to be stored.
- SYSLST, which provides the programmer with a diagnostic listing describing the results of step processing.
- SYSIPT, which contains the storage-step control statements.
- Core image library space containing program overlay segments.

The DOS storage step can be executed in two ways:

1. To create the ISAM data set of FDPs.
2. To update the ISAM data set of FDPs.

The complete 486-byte CSECT entries (including the 10-byte key field) are placed in the data set, but only the 476-byte data areas in each CSECT entry consist of data suitable for transmission to the 3735. The first 10 bytes of each CSECT entry are used as the indexed sequential key.

When creating the ISAM data set, the DTFIS macro operands describe the characteristics of the IJFDLIB file as follows:

DSKEXTNT = 3	DEVICE = 2311*
IOROUT = LOAD	ERREXT = YES
KEYLEN = 10	HINDEX = 2311*
NRECDS = 1	IOAREAL = ISAMAREA
RECFORM = FIXUNB	MODNAME = LOADMOD
RECSIZE = 476	WORKL = WORKAREA
CYLOFL = 8	

When updating the ISAM data set, the DTFIS macro operands describe the characteristics of the IJFDLIB file as follows:

DSKEXTNT = 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

*Where 2311 appears, a job control statement that specifies DEVICE = 2314 can be used to override this.

The application program that retrieves the FDPs from the IJFDLIB file should refer to a particular FDP by searching for the 10-byte key fields that were used to store each section of the FDP, then simply extract the block associated with any particular indexed sequential key using ISAM macro instructions. An example of a DOS BTAM program designed to perform this function, then send the FDPs to a 3735, is found in Appendix G.

The SYSLST data set contains diagnostic information of interest to the programmer. The messages that may appear in the listing are described in Appendix F.

The core image library contains the overlay program segments created by the Linkage Editor.

When the utility has completed normal processing, the FDPs are ready to be sent to a 3735.

This section describes some of the environmental and implementation factors that must be evaluated in designing a teleprocessing system that uses 3735 terminals as remote devices. These considerations are grouped under several headings, as follows:

- Telecommunications Access Methods, which describes the different access methods that can support 3735 operations.
- Communication Procedures, which discusses the communication discipline used with the 3735.
- Application Programs, which examines several implementation factors that must be considered in designing programs to process the data captured at the 3735.
- System Generation Considerations and Storage Estimates, which define the system generation and storage requirements for installing the programs that support 3735 operations.

Telecommunications Access Methods

The 3735 can communicate with System/360 (except Model 20 or Model 67 in Time-Sharing mode) or System/370 (Models 135, 145, 155, 165, or 195) central processing units using any of three telecommunications access methods:

1. The Telecommunications Access Method (TCAM) operating under OS.
2. The Basic Telecommunications Access Method (BTAM) operating under OS.
3. The Basic Telecommunications Access Method (BTAM) operating under DOS.

These access methods control the transmission of information between remote 3735 terminals and a central computer in much the same way as other access methods control. the transfer of information between local input/output devices and the computer. An application programmer can design, write, and test his programs as though he is using a local input/output device, but he performs input/output and message handling operations by using the macro instructions provided by the telecommunications access method used in his installation.

The 3735 terminal uses Binary Synchronous Communications (BSC) procedures to control the flow of information between a terminal and a central system. All data in BSC is transmitted as a serial stream of binary digits (zero and one bits). Synchronous communications means that the active receiving station on a communications channel operates in step with the transmitting station through the recognition of a specific bit pattern (sync pattern) at the beginning of each transmission.

The basic 3735 terminal configuration provides for communication over point-to-point switched (dial-up) common-carrier facilities at speeds of 1200 or 2000 bits per second (bps), or at 2400 bps with an IBM 3872 Modem (or equivalent). (World Trade customers may specify a 600-bps transmission speed.) Communication over multipoint nonswitched (leased) lines at 1200, 2000, or 2400 bps can be provided by the addition of special features to the basic 3735 terminal.

Further information about BSC concepts can be found in the Systems Reference Library publication *General Information–Binary Synchronous Communications,* Order No. GA27-3004. Specific details concerning BSC operations and restrictions in a particular access method are found in the appropriate access method publications, which are referred to in the paragraphs that discuss each access method and listed in the bibliography at the back of this book.

Whichever access method you use, be aware that the FDPs are assembled in the internal code of the 3735. They must not be translated before being sent to the terminal, regardless of the transmission code or access method used.

OS TCAM

Once each 3735 terminal in the teleprocessing network has been identified in the TCAM Message Control Program (MCP), you can use TCAM facilities to transmit properly assembled and structured FDPs to any 3735 terminal in the network, and to receive messages and data from the 3735 terminals. Almost all TCAM facilities provided for other BSC terminals are available to the 3735, including error detection and recovery procedures. The only restriction placed on the 3735 in a TCAM system is that it cannot be used as a primary or secondary operator control terminal.

For a general understanding of the facilities that TCAM can provide, see the *OS TCAM Concepts and Facilities* publication, Order No. GC30-2022. Specific information about designing and building a TCAM Message Control Program and TCAM application programs is found in the *OS TCAM Programmer's Guide and Reference Manual,* Order No. GC30-2024.

OS and DOS BTAM

BTAM controls terminal input/output operations initiated by READ and WRITE macro instructions issued in a teleprocessing application program. When each 3735 terminal in the teleprocessing system has been identified in the proper BTAM control block and terminal list specifications, you can use the BSC facilities provided by BTAM to transmit properly assembled and structured FDPs to any terminal in the network, and to receive messages and data from the 3735 terminals.

An example of a DOS BTAM program that retrieves FDPs from an indexed sequential file and sends them to a 3735 is found in Appendix G. An example of a similar OS BTAM program is found in Appendix H.

Detailed information on the facilities that OS BTAM provides is found in the *OS BTAM* Systems Reference Library publication, Order No. GC30-2004. Similar information for the DOS BTAM user is found in the *DOS BTAM* Systems Reference Library publication, Order No. GC30-5001.

Transmission Codes

The transmission codes used with the 3735 are EBCDIC and ASCII. (A modified version of ASCII is used as the internal code of the 3735.) EBCDIC characters are transmitted low-order bit first in the following sequence: 7, 6, 5, 4, 3, 2, 1, 0. No parity is transmitted; checking is provided by a 16-bit cyclic accumulation. The EBCDIC codes used by the 3735 are shown in Figure 28.

ASCII characters are transmitted low-order bit first as follows: 1, 2, 3, 4, 5, 6, 7, parity. Checking is performed by a character parity check and an 8-bit longitudinal redundancy accumulation. The ASCII codes used by the 3735 are shown in Figure 29.

Timeouts

Timeouts of different lengths are provided to control message traffic on a line. All of these timeouts except one operate as part of the binary synchronous communication (BSC) procedures. In addition to these BSC timeouts, the 3735 provides an additional three-minute timeout that is used during inquiry operations to prevent the 3735 from waiting indefinitely for a response from the CPU. When the 3735 has waited for a response to an inquiry for three minutes, it times out, sets an indicator, and returns control to the FDP that issued the inquiry request. The FDP can then check the status of the indicator and proceed accordingly.

Transmission checking, format checking, block counting, and I/O and buffer check procedures are also provided to maintain orderly message traffic on the communication line.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL				SP	&	–						{	}		0
1							/		a	j			A	J		1
2									b	k	s		B	K	S	2
3									c	l	t		C	L	T	3
4									d	m	u		D	M	U	4
5	HT	NL	LF						e	n	v		E	N	V	5
6		BS							f	o	w		F	O	W	6
7									g	p	x		G	P	X	7
8									h	q	y		H	Q	Y	8
9									i	r	z		I	R	Z	9
A					¢	!		:								
B					.	$	,	#								
C		IFS			<	*	%	@								
D	CR				(	)	_	'			[	]				
E		IRS			+	;	>	=		±						
F					|	¬	?	"								

Figure 28. 3735 Character Code Chart - EBCDIC

Communication Procedures

Communicate mode is entered when the 3735 operator presses the ENTER or TAB key following the compression of operator-created data. The 3735 keyboard locks and the COMM indicator turns on. The 3735 remains in communicate mode until communication is terminated by a Power Down message from the CPU or by the operator.

Except during inquiry operations, the following message sequence must be used:

1. The 3735 must transmit its data (if any) to the CPU.
2. After receiving all 3735 data, the CPU may transmit FDPs. If FDPs are transmitted, the entire set required to be resident at the terminal must be sent.
3. Following any FDP transmission, the CPU can transmit other data. The FDPs and data received by the 3735 overlay, on disk, the FDPs and data that were on the disk when communication began.

3735-to-CPU Transmission

On a switched line, if the operator dials the CPU from the 3735, the initial line control sequence is as follows:

3735	*CPU*
ID ENQ	
	ID ACK 0
Data Block	
.	ACK 1
.	.
.	.
	.
EOT	

Rows	Columns	0	1	2	3	4	5	6	7
	$b_7b_6b_5$	000	001	010	011	100	101	110	111
	$b_4b_3b_2b_1$								
0	0 0 0 0	NUL		SP	0	@	P		
1	0 0 0 1			!	1	A	Q		
2	0 0 1 0			"	2	B	R		
3	0 0 1 1			#	3	C	S		
4	0 1 0 0			$	4	D	T		
5	0 1 0 1			%	5	E	U		
6	0 1 1 0			&	6	F	V		
7	0 1 1 1			'	7	G	W		
8	1 0 0 0	BS		(	8	H	X		
9	1 0 0 1	HT		)	9	I	Y		
A	1 0 1 0	LF		*	:	J	Z		
B	1 0 1 1			+	;	K	[		
C	1 1 0 0		FS	,	<	L	\		|
D	1 1 0 1	CR		-	=	M	]		}
E	1 1 1 0		RS	.	>	N	^		
F	1 1 1 1			/	?	O			

Figure 29. 3735 Character Code Chart - ASCII

If the 3735 has no data to send, it sends an EOT instead of the first data block.
If the CPU calls the 3735, the initial line control sequence is as follows:

CPU	*3735*
ID ENQ	
	ID ACK 0
EOT	
	ENQ
ACK 0	
	Data Block
ACK 1	.
.	.
.	.
.	
	EOT

If the 3735 has no data to send, it sends an EOT instead of the first data block.

All data blocks except the last consist of 476 bytes of data preceded by STX and followed by ETB. The last block can be from 0 to 475 bytes of data preceded by STX and followed by ETX. (Zero bytes of data result when the available data exactly fills the previous disk sector.) The 3735 does not use the SOH or ITB characters in communicate mode. Only the characters shown in Figure 28 (EBCDIC) or Figure 29 (ASCII) are sent by the 3735 as data characters. Note, however, that the 3735 does not transmit the carriage control characters (NL, HT, BS, LF, and CR). The 3735 retransmits a data block acknowledged with a NAK character for an indefinite number of times. The 3735 never transmits a TTD. It will, however, receive WACKs (and respond with ENQs) for an in-

definite number of times. If it receives an RVI, the 3735 sends a DISC sequence and aborts transmission. If the CPU sends an EOT in response to a data block, the 3735 issues a DISC sequence and aborts the transmission.

The 3735 data transmission to the CPU is complete when an EOT is sent to the CPU. The CPU should respond with DISC if no data is to be sent to the 3735. If this is the case, the 3735 drops the communication line, reinitializes itself for further communications, and waits for another connection with a CPU. The same CPU or different CPUs can continue to "read" the 3735 data until data is successfully sent from the CPU to the 3735, or until an attempt to send data to the 3735 has been aborted.

On a multipoint line, if the CPU polls the 3735 and the terminal has data to transmit, the 3735 responds with the first data block. If it has no data to transmit, the 3735 responds with EOT. If the CPU selects the 3735 and the terminal has no data to transmit, the 3735 replies with ACK 0. If the CPU selects the 3735 and the terminal does have data to transmit, the 3735 replies with RVI. The CPU should then send EOT POLL to "read" the data.

The data block formats are the same for multipoint networks as they are for switched networks. The 3735 retransmits a data block acknowledged with a NAK character for an indefinite number of times. The 3735 never transmits a TTD. It will, however, receive WACKs (and respond with ENQs) for an indefinite number of times. If the CPU sends an EOT in response to a block, the 3735 aborts the transmission. If it receives an RVI, the 3735 issues an EOT, waits for a subsequent polling sequence, and then continues transmission with the block following the one that was acknowledged with the RVI character.

Sending Abort Conditions

The 3735 aborts the sending phase of communication if any of the following conditions occur:

- The CPU ID is invalid (switched lines).
- An EOT is received as the response to a data block, or as the response to an ENQ (switched or multipoint lines).
- A DISC sequence is received before the 3735 sends the EOT for end of data (switched lines).
- An RVI is received in response to a data block (switched lines).
- The 3735 transmits 15 ENQs to solicit a response from the CPU (switched or multipoint lines).

In all of the above cases, the 3735 aborts transmission and reinitializes itself for further communication. If the CPU establishes communication and requests the data again, transmission starts from the beginning.

Another abort condition can arise when the 3735 cannot read a data block from its internal disk. The 3735 TCP attempts to read the disk sector 30 times. If the read is still unsuccessful after 30 attempts, the following block is sent instead of the error block:

```
S N   N     E
T U S U 06  T
X L   L     X
```

A DISC (switched lines) or EOT (multipoint lines) sequence is sent following the acknowledgment of the block. The 3735 TCP drops out of communicate mode, initiates a data listing function, and cancels the records in error. (The 3735 cannot continue transmission following the error block. Since the block cannot be read, the data chaining information is destroyed and the location of the next block is unknown.) A listing of the error records is then printed on the Selectric ® printer. The TCP then recompresses the remaining records on the disk and reenters communicate mode. When the CPU establishes communication again, transmission starts from the beginning.

CPU-to-3735 Transmission

The 3735 can receive several different types of messages. Each message type is handled differently in terms of the ground rules that must be followed and the area of the disk surface that is devoted to servicing and storing that message type. No FDPs or data can

be transmitted from the CPU to the 3735 unless the 3735 data records (the results of normal document creation activity) have been transmitted to the CPU first. The message type identifier formats are defined for these message types:

- FDP messages.
- ID List messages.
- Selectric messages.
- Terminate communicate mode messages.
- Power Down messages.
- Text messages.

These message type identifiers are described in the following paragraphs. Message blocks must begin with an STX character and end with an ETB or ETX character. Normally, ETX is used only for the last block of a message.

Form Description Program Message Format

FDP data blocks must be sent to the 3735 before any other data is sent. The FDP message blocks must be preceded by the following preliminary message:

```
S N   N                                E
T U F U (up to 473 optional characters) T
X L   L                                B
```

Note: Up to 473 characters may be transmitted between the second NUL and the ETB. These optional fill characters are ignored by the 3735 except for accumulation of CRC or LRC, and may be used for any purpose, such as TCAM header characters. The 476-byte block is stored on the disk and occupies one sector until transmission is completed and the FDPs are packed and cataloged. The disk sector is then used for packed FDPs.

Each FDP message block must be formatted as follows:

```
S Unpacked      E
T FD Program    T
X Data Block    B
```

Unpacked FDP data bytes are paired, and have the following format:

```
Even byte   0100 0XXX
Odd byte    0111 YYYY
```

When packed at the 3735, the data appears as PXXX YYYY, where P represents a generated odd-parity bit.

The last FDP message block must be followed by:

```
S N   N                                E
T U E U (up to 473 optional characters) T
X L   L                                B
```

The FDPs received are stored on the disk, with each FDP block occupying one disk sector. (The NUL E NUL block is not stored on the disk.) The FDPs are counted as they are received, and if the space in the initial FDP directory is not sufficient to catalog all the FDPs received, additional disk sectors are reserved for directory expansion. Each disk sector can catalog 58 FDPs, and up to 17 sectors may be taken from the customer data area for directory expansions. The initial FDP directory sector created during 3735 system generation is included in the terminal control program area.

Note: When you transmit FDPs to a 3735 terminal with ASCII code, do not send the sector flags (3X'FFFF'), which are the last six bytes of each 476-byte FDP block. Either transmit only the first 470 bytes of each record (framed by STX and ETB characters) or change the X'FF' bytes to X'7F' and then transmit the entire 476-byte block. These last six bytes, if sent, are stripped off at the terminal.

ID List Message Format

The format of a CPU ID list message sent to the 3735 is:

```
S N   N        E
T U L U ID List T
X L   L        B
```

The ID list can consist of up to 236 characters. Each ID must be delimited by a file separator character (X'1C'), and the entire list by a record separator character (X'1E'). The IDs can consist of from one to 15 characters each. The number of IDs in the list is limited only by the number which, with their delimiters, will fit in the 236 bytes. The list can be padded with any valid text characters following the record separator to expand the block size to as many as 476 characters.

When the 3735 receives the new ID list, it stores the ID list temporarily until the transmission is successfully completed. The 3735 replaces its old ID list with the new one only when a power down or terminate communicate mode message has been received, or when the operator uses the OPER key to terminate communicate mode.

Selectric Message Format

The format of a Selectric message sent to the 3735 is:

```
S N   N            E
T U M U Message Text T
X L   L            B
```

The message text can consist of up to 233 characters. The 3735 performs a new line (NL) function before starting to print the message. Other carriage control characters may be embedded in the message data to provide further Selectric ® printer motion control, if desired. (The carriage control characters permitted are those described in the FDFORM MESSAGE operand discussion. Note, however, that no repetition factor is permitted with embedded carriage control codes, since this type of message is not generated by the FD macros.) The last message character must be followed by a record separator (X'1E'). More text characters may follow the record separator to expand the block size to as many as 476 bytes (including the NUL M NUL header and the record separator, but not including the framing STX and ETB or ETX). After the transmission has been completed, the 3735 TCP prints the message on the Selectric ® printer (just before powering down or returning to local mode). If more than one Selectric message is sent, only the last one received is printed.

Terminate Communicate Mode Message Format

The format of a terminate communicate mode message sent to the 3735 is:

```
S N   N                              E
T U T U (up to 473 optional characters) T
X L   L                              B
```

This message instructs the 3735 TCP both to pack and catalog all received FDPs and data, and to return to local mode following the next EOT or DISC sequence received from the CPU.

Note: Up to 473 characters may be transmitted between the second NUL and the ETB. These optional fill characters are ignored by the 3735 except for the accumulation of the CRC or LRC.

Power Down Message Format

The format of a power down message sent to the 3735 is:

```
S N   N                              E
T U P U (up to 473 optional characters) T
X L   L                              B
```

This message instructs the 3735 TCP to pack and catalog all received FDPs and data, and to power down the 3735 following the next EOT or DISC sequence received from the CPU.

Note: Up to 473 characters may be transmitted between the second NUL and the ETB. These optional fill characters are ignored by the 3735 except for accumulation of the CRC or LRC.

Text Message Format

Text transmitted from the CPU to the 3735 requires no special preliminary message. If FDPs and text messages are sent, the FDPs must be sent first. The ID list, Selectric, or power down messages can be sent before FDPs, after FDPs, or after text. Text transmissions to the 3735 must be in blocks of 476 characters or less (excluding the framing STX and ETB or ETX characters). If the last block is short, the ETB or ETX character must immediately follow the record separator (X'1E') delimiting the record or the block must be padded with null characters (X'00').

One extra disk sector is used by the 3735 terminal control program (TCP) to indicate that data is being received. This disk sector is released when the transmission has ended and data compression begins. Only one such extra sector is used, even if multiple transmissions occur (for example, if more than one CPU calls, or the same CPU calls to add to data already received). The 3735 maintains a count of the data records received. If more than 58 records are received, an additional disk sector is reserved for directory expansion. Up to 17 additional sectors can be taken from the customer data area for directory expansions (for each 58 records). Each data record received from the CPU is sequentially numbered starting with 000.

The first three characters of each form record transmitted to the 3735 must contain the ID of the FDP that is to be used for processing the record. Each form record must end with a record separator. Fields must contain characters for each position, or must be terminated by a delimiter (IFS for EBCDIC, FS for ASCII; both are equivalent to hexadecimal '1C'). Further information concerning text message formats is found in the "Application Programs" section.

Transmission Blocks

Message blocks transmitted to the 3735 must not be longer than 476 characters (excluding the framing STX. . .ETB or ETX). Block length may be less than 476 characters; however, during reception each block is stored as a sector of information on the magnetic disk. Therefore blocks of less than 476 characters reduce the amount of message data that can be accomodated by the 3735 disk storage device. Following reception of a message, the terminal control program searches the received records, removes any excess record separator characters, and packs the blocks into contiguous blocks. This storage compaction is performed only after the reception of the message is completed.

Receive Abort Conditions

The 3735 aborts the data reception phase of communication if any of the following conditions arise:

- The CPU attempts to transmit data on a switched line before the 3735 operator-created data has been sent to the CPU.
- An illegal character is detected in a block received from the CPU.
- The CPU sends a block of more then 476 data characters with a valid BCC or LRC check. This causes a dynamic buffer overflow condition.
- The CPU attempts to send more FDPs and data than the 3735 disk can hold. This causes a disk full condition.
- The CPU sends more than 1000 FDPs or more than 1000 data records. Both the FDP directory and the data directory can catalog only 1000 records each. This causes a directory full condition.
- The CPU sends an undefined header block of NUL X NUL (where the X represents a character other than F,E,L,M,T,P, or I).

If any of the above conditions arise, the 3735 sends a status message to the CPU. The line control sequences for a switched network are as follows:

CPU	*3735*
Error Block	
	EOT
EOT	
	ENQ
ACK 0	
	Status Message
ACK 1	
	DISC

On a multipoint line, the 3735 waits to be polled following the EOT abort, then begins transmission of the status message. If the CPU selects instead of polls, the 3735 responds with NAK five times, then aborts communication. The status message format is:

```
S N    N       E
T U S  U  bb   T
X L    L       X
```

The status bytes (bb) have the following meanings:

bb	*Meaning*
00	CPU attempted to send before receiving (and 3735 data has not been read at least once)
01	Illegal character in block
02	Dynamic buffer overflow
03	Disk full
04	Directory full
05	Undefined header (NUL X NUL)

If any of these abort conditions arises, or if a 20-second BSC timeout condition arises, the 3735 ignores all data received from the CPU. CPU transmission must be reinitiated from the beginning.

Inquiry Operations

The user creates the inquiry message to be sent to the CPU with his FDP. Before executing the SEND command, the 3735 TCP sets a NUL I NUL header sequence in the first three bytes of the inquiry buffer. When the message is sent, the TCP furnishes the STX and ETX to delimit the entire message and turns off the TIMEOUT indicator. On a switched network, the initial SEND command causes the 3735 COMM indicator to blink, thus letting the operator know that a call should be placed to the CPU. The line control for handling this inquiry sequence is a follows:

```
3735                          CPU

ID ENQ
                              ID ACK 0
S N    N            E
T U I  U  message   T
X L    L            X
                              ACK 1
EOT
```

Following the transmission of the EOT, the 3735 waits up to three minutes for a response from the CPU. No line activity need take place during that interval. When the CPU has the response ready, communication must continue as follows:

```
3735          CPU

              ENQ
ACK 0
              S N    N            E
              T U I  U  message   T
              X L    L            X
ACK 1
              EOT
```

The inquiry block from the 3735 to the CPU is always 236 bytes long (not including the framing STX and ETX). The CPU response can be of variable length, but no greater than 236 bytes (including the NUL I NUL header). The CPU response must have the indicated NUL I NUL header. Following reception of the response, the 3735 places the data in the inquiry buffer (including the NUL I NUL header) and returns control to the FDP. No other line activity takes place until another message is sent or until the FDP

issues a DISC command. The CPU should remain in receive mode waiting for another ENQ. If another SEND command is issued, the line control is as follows:

3735	*CPU*
ENQ	
	ACK 0
Inquiry	
	ACK 1
EOT	

This procedure is repeated for each inquiry operation. It is the user's responsibility at the CPU to limit the amount of time between inquiries, if desired. Subsequent inquiry operations may be controlled by operator action, and thus the amount of time between the CPU's EOT and the 3735's ENQ may be dependent on the operator.

The FDP DISC command causes a disconnect sequence to be sent to the CPU.

On a multipoint line, the 3735 waits to be polled before sending the inquiry, and also waits up to three minutes to be selected for the response reception.

The inquiry response from the CPU must consist of fixed-length fields without any field or record separators. Care should be taken when creating a message at the 3735, since the CPU response is placed in the inquiry buffer. If the message data is less than 233 bytes long, the ETX received from the CPU is also placed in the inquiry buffer, and the user must make sure that this character is removed before sending another inquiry (if any). An FDCTRL COMMAND = CLEAR (INQ) can be used for this purpose.

If the inquiry operation cannot be successfully completed, the TIMEOUT indicator is turned on and control is returned to the FDP. This indicator is set if the three-minute timeout is exceeded or if any abort conditions are encountered in the inquiry operation. The FDP can then test this indicator by using an FDCTRL IF = TIMEOUT instruction, and take appropriate action if necessary.

Application Programs

Previous sections have described how a forms encoder can write FDPs that can be used to capture data at a 3735 terminal. The application programmer must also consider how the 3735 formats the data that is sent to the central system. The following discussions provide information on how data from the CPU must be formatted for proper handling by the 3735, and how the 3735 formats the data for transmission to the CPU.

The output of the FD utility is 476-byte blocks containing macro-generated bit strings that form the data portion of FDP messages. These FDPs reside in a user data set. The user must create teleprocessing application programs to select and transmit particular FDPs to a 3735, just as he must create programs to process the data captured at the 3735 terminals.

In transmitting the FDPs to the individual terminals, not all programs in the user data set need be transmitted to all 3735 terminals. The user can be selective, sending only certain programs from the data set to certain terminals. However, when programs are sent to a 3735, all of the FDPs that are to reside at that terminal must be transmitted continuously. Additional programs cannot be sent to a 3735 without overlaying those already there. The FDPs already at the terminal must be retransmitted along with any new ones to make sure that all the desired programs are present at the 3735.

Data received from the CPU must be played back under control of the FDP specified in the first three bytes of the record. Counter operations, editing, and so forth can be performed when playing back CPU-generated data records.

The field formats must be defined to correspond *exactly* with the FDP specifications for each field. Each field must contain characters for each position, or must be terminated by a delimiter. For example, if the 3735 is an ASCII model, the field is 5 characters long,

and the data is the characters ABC, the hexadecimal data string (after translation to ASCII) must appear as:

'4142431C' [Delimiter is X'1C' .]

or

'4142432020' [No delimiter - all field positions are filled.]

This format must be used whether or not the FDP indicates that the field is to be transmitted.

If the batch specification is used, two blank bytes must be placed directly in front of the affected data so that the 3735 can insert the batch indicator and batch number.

If a field definition specifies a buffered data source, the data must be included in the CPU-generated record. In addition, no data is stored on disk or in any buffer (except the PCH or LPB buffers) as a result of a data sink specification.

The only conditional execution flag that may be included in the data is one that terminates cyclic processing. To terminate a cycle, you must include, in the data, some graphic character (for example, an asterisk) that indicates the end of cyclic data. The FDP can then be coded to test for this character during playback of the CPU data. (Note that this technique may make your playback FDP unsuitable for creation of form records at the 3735.)

CPU data cannot be modified by the operator. The record must be terminated with the record separator (X'1E').

Switched Network Considerations

The CPU must read the 3735 data before it can transmit any data block (text, FDPs, ID List, etc.) to the terminal. Even if the 3735 has data to send, it accepts CPU data if the CPU has read the 3735 data at least once. Following the reception of the EOT from the 3735, the line control must be as follows:

3735	*CPU*
	ENQ
ACK 0	
	Data Block
ACK 1	.
.	.
.	.
.	
	EOT
DISC	

If no data is to be sent to the 3735, the CPU should send DISC instead of ENQ. When EOT is sent, the 3735 replies with DISC.

Data blocks can be terminated with either ETB or ETX. (Normally, ETX is used only for the last block.) The 3735 receives data blocks and stores them on the disk until it receives an EOT or DISC sequence. The 3735 then drops the line and tests for the reception of a power down or terminate communicate mode message, or for depression of the OPER key. If none of these conditions has occurred, the 3735 remains in communicate mode and can accept more data if another call is placed. The data received is packed, compressed, and cataloged if a power down or terminate communicate mode message was received before EOT or DISC (or the OPER key was pressed). The 3735 powers down or exits to local mode after the data is cataloged.

The 3735 accepts TTDs (and responds with NAKs) for an indefinite number of times. The 3735 treats a received SOH as an STX. In addition, the ITB character can be sent to the 3735. The 3735 sends WACK sequences when it cannot write a received data block on its disk within a two-second period. The 3735 never sends an RVI on a switched network.

Multipoint Network Considerations

Following reception of an EOT from the 3735, the line control must be as follows:

3735	*CPU*
Last Block	
	ACK 0 or ACK 1
EOT	
	(EOT) SELECT
ACK 0	
	Data Block
ACK 1	.
.	.
.	.
.	
	EOT

Except for RVI, the data formats and procedures are the same in multipoint networks as in switched. The 3735 accepts TTDs (and responds with NAKs) for an indefinite number of times. The 3735 treats a received SOH as an STX. In addition, the ITB character can be sent to the 3735. The 3735 sends WACK sequences when it cannot write a received data block on its disk within a two-second period. The 3735 sends an RVI only if the CPU selects the 3735 when the terminal has data to send. When not in communicate mode, the 3735 sends EOT in response to a poll, and NAK in response to selection.

Relating Application Programs to Form Data

In addition to specifying the way in which a document is processed, the FDPs also govern what data is stored for transmission to the central computer. The application program that processes this data must consider not only the data, but also its message format.

Bypassed Fields

During document creation, some fields of a form may be bypassed. The bypass may be caused by information entered by the 3735 operator; for example, the FDP may specify that a credit entry causes bypassing of a debit field. Or the bypass may be initiated by the terminal operator when all repetitions of a group of fields designated for cycling are not used. Or the operator may elect to pass an optional field because the information is not required for the transaction being processed.

No automatic indication of fields bypassed is provided for transmission to the CPU. Thus, the application programmer and forms encoder must consider how the fields are to be identified. The FDPs can be written so that all fields are filled, either with input data or with blanks. The application program can then identify each field by its position within the record.

On the other hand, when the FDP design allows fields to be bypassed, you can reduce transmitted message length by identifying transmitted fields with unique character strings. This allows your application program to recognize what fields have been bypassed and to identify the data that has been transmitted to it.

Lines and fields bypassed when an end cycle function is performed are treated in the same way as described above; no automatic indication is provided. Thus, unique identifiers may be required to show where cyclically-created data stops.

Fields in which the operator does not enter data are not considered bypassed fields. These fields are transmitted with or without blanks or delimiters as specified by the PACKING operand of the FDFORM macro instruction.

Form Records

A record separator character (RS for ASCII machines, IRS for EBCDIC machines–both X'1E') is inserted automatically at the end of each form record transmitted from the 3735 to the CPU. Each record received by the 3735 must also end with the record separator.

The three-byte FDP identifier is supplied automatically in the first three positions of records sent to the CPU. The data sent to the CPU consists of fixed-length fields, with the complete record terminated by the record separator (X'1E'), or else variable-length fields with each field terminated by the file separator (X'1C') or some other delimiter inserted by the user's FDP. These data formats are controlled by the coding of the FDFORM PACKING operand.

Note: When the 3735 sends its data records to the CPU, it sends the FDP number (as specified by FID = ddd in the FDFORM macro) for every FDP that was used at the 3735 since the last time data was sent to the CPU. Thus, regardless of whether or not an FDP stored any data for transmission to the CPU, its FID number is stored on the 3735 disk, and the FID number is sent to the CPU when communication takes place. Your application program should recognize and handle these "null" FDP records.

For example, consider the appearance of a record created by the sample program shown in Figure 20 on an ASCII 3735 with this input data:

Customer number: 012345

Record sequence number: 007

Item	*Quan*	*Price each*
JONES6789	2	10.00
JONESABCD	1	5.99

The record sent to the CPU would appear as follows:

FDP # Customer # Item Quantity Price

'303031303132333435 1C 4A4F4E4553 6373839 1C 20202020 321C 313030301C'

Gross * Item Quantity Price Gross *

'30323030301C 4A4F4E4553 414243441C 20202020311C 3539391C 303539391C'

End Cycle Totals* Tax* Grand Total * RSN#

'2A2A2A1C 30323539391C 303130341C 30323730331C 3030371C 1E'

On-Line Processing

An FDP can be used for a simple inquiry consisting of a request transaction sent to the central computer, followed by a completely formatted reply transmitted back. Or the FDP can allow keying of abbreviated inquiries, with the terminal adding transaction codes and other system information to complete the transaction. The reply can contain variable data to be printed out, with appropriate headings and editing added by the FDP. This approach makes the inquiry operation simpler, less prone to error, and less demanding of line time, since the FDP can be designed to perform the required editing functions.

Application programs designed to handle on-line processing requests from a 3735 need not be active in the CPU at all times. All you need to provide is an inquiry analysis routine that can determine what service is requested, then load and pass control to the application program designed to provide that service. Since the 3735 can perform most of its operations off-line, without requiring service by the central computer, this approach to inquiry handling makes economical use of the computer's main storage without significantly increasing inquiry response time.

*All transmitted fields originating from a counter which are not edited (by PICTURE, FILL, etc.) are transmitted as 10-digit fields; thus the fields indicated would be filled with zeros (X'30') to increase their size to 10 digits. For the sake of brevity, these left zeros have not been shown.

During inquiry operations over switched lines, the 3735 FDP may not issue a disconnect (DISC) command until it has handled all of its inquiry requests. Thus, if the 3735 does not send another inquiry to the CPU immediately after receiving a response to a previous inquiry, the application program at the CPU should take action to ensure that the communication line remains open for some period of time. The specific time period used should be selected jointly by the application programmer and the forms encoder so that a balance is struck between allowing a 3735 to monopolize use of the line, and obliging the 3735 operator to dial the CPU repeatedly while using the FDP.

Batch Processing

The 3735 terminal can operate off-line all day in many environments. In some situations, however, the volume of data to be accumulated at the terminal may exceed its storage capacity. When such cases arise, the terminal needs to communicate with the central computer so that it can send the collected data to the application programs that process it. From the standpoint of the central computer, the processing to be performed is what would normally be done at some later time during the day, except that the 3735 has requested service ahead of schedule. Application programs should be designed to handle such situations, as well as the normal once-a-day batch processing of 3735 data. If the central system has limited processing resources, and cannot be set up to process this much data on demand, the received 3735 data can easily be placed on an auxiliary storage device for later processing.

In the normal batch processing situation, a separate application program may be required to process the data collected by each FDP. The design of each application program depends on the specific format and type of data transmitted from the 3735. Thus close cooperation is required between the forms encoder who writes the FD macro statements and the application programmer who writes the programs that process the form data.

Combined Operation

The use of the 3735 for an order entry application illustrates the interaction of off-line data recording and on-line inquiry. In this application, the 3735 is programmed for order entry. Input data formats and validity checks are programmed to guide the operator through the order procedure. When an item number and quantity are entered, the 3735 operator can send an inquiry to the central computer to see if a sufficient quantity of this item is in stock to meet the order. If it is, the price can be returned to the terminal for a price-times-quantity extension. If not, the operator can be notified to select an alternate quantity or item, or to indicate whether a "back ordered" status is acceptable.

The advantage of this approach is that it provides most of the functions of on-line order entry without the additional CPU, file, multiplexer, and line costs of the on-line approach. A single leased line with a simple inquiry program to an abbreviated inventory file is all that is required to support several terminals.

System Generation Considerations

System generation procedures for OS and DOS configurations should include a telecommunications access method, the FD macros, and the FD utility. For further system generation considerations and requirements, refer to the following publications:

OS - OS System Generation GC28-6554
DOS - DOS System Generation GC24-5033

Storage Estimates

Since the FD macro instructions do not generate code that is executed in the central computer, the generated FDPs do not require any CPU main storage. However, auxiliary storage must be provided for the data set containing the collection of FDPs that may be sent to 3735 terminals. The storage requirements for the programs that support FDP creation activities are described in the following sections. Normally, the required programs are an Assembler, a Linkage Editor, the FD utility, and a teleprocessing access method. User application programs that handle transmission between the CPU and 3735 terminals also require some storage, and should be considered in designing the total system configuration.

3735 Disk Storage Considerations

An accurate estimate of storage usage on the 3735 disk cannot be obtained until the user has actually coded and assembled his FDPs, since the FDPs will be of different lengths. An MNOTE message (IDF147) describes the 3735 disk storage required for the assembled FDP. Once an FDP has been coded, you can estimate the storage required for data records that it creates at the 3735. The number of bytes stored on the disk for each field is equal to the maximum field size plus one. (The additional byte is for the file separator character, X'1C'.) Additional control bytes are stored as follows:

- The FDP number (3 bytes) is stored at the beginning of each record.
- A one-byte flag is stored with each field that is *not* to be sent to the CPU.
- A one-byte record separator character (X'1E') is stored at the end of each created record.
- If the FDFIELD BATCH operand was coded for a field, a one-byte batch flag (plus one byte for the batch number—two bytes in all) is stored for each BATCH operand encountered.
- If the CYCLE operand was coded, a one-byte end-cycle flag is included for each CYCLE operand encountered. When estimating storage requirements for cyclic processing, assume that the CYCLE will execute the full number of times specified by the cycle count.
- For each processing decision at which a branch can be taken, a one-byte branch-taken or branch-not-taken flag is stored in the record (unless the branch occurs in an FDFIELD IND operand, in which case no flag is stored).
- When a field that is to be sent to the CPU is edited by an FDFIELD PICTURE operand, all the edited characters resulting from the 'picturespec' definition are included in the record, in addition to the raw data.
- When a field that is to be sent to the CPU is right-justified with left blanks or left zeros, or centered, the edited output is stored in addition to the raw input data.

When creating form records at a 3735, almost all source data is stored on the 3735 disk. Note, however, that data from the sources identified by SOURCE = FID, SOURCE = RSN, SOURCE = 'string', and SOURCE = CTR (d) is stored on the disk only when SINK = TMT is in effect. Data is stored on the disk for all other source specifications, whether or not the data is to be sent to the CPU. You should remember that the basic disk capacity is 62,832 bytes (expandable, by special features, to 146,608 bytes), and that the disk directory can expand to hold entries for a maximum of 1000 records for FDPs, and 1000 records for data. As each FDP or data record is stored, the respective disk directory is expanded at the rate of 58 directory entries per disk sector, thus reducing the available storage on the disk by 476 bytes per directory sector.

The File Storage feature requires an additional eight sectors of customer data storage space to accomodate the expanded terminal control program (TCP).

When FDPs are being stored on the 3735 disk, only 234 bytes of FDP data are stored in each sector (plus 6 bytes of chaining information). Thus, FDPs require approximately twice as many disk sectors to store the same number of bytes as do data records, which are stored with 476 bytes in each disk sector.

OS Storage Considerations

The FD macro instruction definitions require no main storage, but they must be included in a macro library on some auxiliary storage device. These macros can be used to generate an FDP with the smallest Assembler available in the system. The FD utility requires main storage for execution, but needs only that required for the minimum Linkage Editor available in the system – 15K bytes (Level E) or 44K bytes (Level F). The other two utility steps require no more than 10 K bytes of main storage of execution.

OS BTAM and TCAM users must provide an OS MFT system with at least 128K bytes of main storage, or an OS MVT system with at least 256K bytes of main storage. Thus the minimum system configuration can support the 3735 terminal in a teleprocessing environment under OS is a System/360 Model 40 with at least 128K bytes of main storage (or an equivalent System/370).

DOS Storage Considerations

The FD macro instruction definitions require no main storage, but they must be included in a macro library on some auxiliary storage device. These macros can be used to generate an FDP with the smallest Assembler available in the system (Assembler D). The FD utility requires 12K bytes of main storage for execution.

DOS BTAM, however, requires a system with at least 32K bytes of main storage; thus the minimum system configuration that can support the 3735 terminal in a teleprocessing environment under DOS is a System/360 Model 22 with at least 32K bytes of main storage (or an equivalent System/370).

THIS PAGE INTENTIONALLY LEFT BLANK

A self-check digit is the units position of a self-checking number. The self-check digit is developed by calculations made on the base number (the original number without the self-check digit). The calculated digit is then included with the base number to create the self-checking number. When a self-checking number is entered into the 3735, the calculations originally performed are repeated, and the generated digit is compared to the self-check digit in the number that was entered. If the numbers are not the same, a self-check error is indicated on the 3735.

When the SELFCHK operand specifies GENONLY, then the self-check digit is computed on all the digits in the number that is entered. In this case, only the base number is present when the self-checking procedure begins, so no comparison is made when the self-check digit is computed. Instead, the generated digit is placed at the end of the base number. Any further operations with this number (such as COMPARE or SINK) must include an additional character space for the self-check digit.

The 3735 calculates the self-check digit with either the Modulo-10 or Modulo-11 algorithm, depending on which was specified in the SELFCHK operand. If you calculate your own self-checking digits, you must perform the same computations as the 3735 does, or else the self-check digits will not be the same. The Modulo-10 algorithm operates as follows:

1. Disregarding the self-check digit (if present), multiply the base number's units position, and every alternate position moving leftward, by two. For example:

6	1	2	4	8	1
x2		x2		x2	
12		4		16	

2. Add the digits of these products to the digits of the base number which were *not* multiplied by two:

   1+2+1+4+4+1+6 = 19

3. Subtract this total from the next higher number ending in zero:

```
 20
-19
 --
  1
```

4. The result of this subtraction is the self-check digit (in this example, 1). Thus, the complete self-checking number is 612481. The 3735 treats a self-check digit of 10 as a zero.

   The Modulo-11 algorithm operates as follows:

1. Disregarding the self-check digit (if present), multiply the units position of the base number by two, the tens position by three, the hundreds position by four, and so forth. Continue this procedure until you have multiplied by seven (if the number contains that many digits), then begin multiplying by two again. For example:

5	6	6	2	1	8	6	5
x2	x7	x6	x5	x4	x3	x2	
10	42	36	10	4	24	12	

2. Add the products to each other.

   10+42+36+10+4+24+12 = 138

3. Divide the sum of the products by 11;

```
   12
11)138
   11
   28
   22
    6
```

4. Subtract the remainder of this division from 11.

```
 11
-6
 5
```

5. The result of this subtraction is the self-check digit (in this case, 5). Thus, the complete self-checking number is 56621865. The 3735 treats a self-check digit of 11 as a zero.

   Note: A number that generates a self-check digit of 10 is illegal, and cannot be processed by an IBM 29 Card Punch that has the self-checking number feature. You should not use such numbers as self-checking numbers. (If requested to generate a self-check digit on such a number, the 3735 generates a zero.)

Appendix B. Form Description Macro Instruction Format Summary

Name	Operation	Operands
symbol	FDFORM	FID = 'ddd' [,PACKING = { NO / YES / DELIMIT }] [,DEVICES = (3735,K[D]] [,BUFFERS = ([RPB] [, (LPB[, { 132 / 126 / 120 / d }])])] [,OBJECT = { OS / DOS }] * [,MODE = { NONLOAD / LOAD }] [,MRGSTOP = { 0 / d }] [,MESSAGE = ({ cc[({ 1 / d })] / 'string' } [, { cc[({ 1 / d })] / 'string' }]...)] [,HTAB = (d[,d] . . .)]
[symbol]	FDPAGE	[pagenum] [,HEIGHT = { 66 / d }] [,VMRG = ([{ 1 / dt }] [, { height / db }])] [,SAVELOC = { NO / YES / d }]
[symbol]	FDLINE	[{ linenum / SKIP (d) }] [,WIDTH = { 85 / d }] [,HMRG = ([{ mrgstop + 1 / dl }] [, { width / dr }])] [,CYCLE = ([d] [,limit] [,target])] [,SAVELOC = { NO / YES / d }]

Name	Operation	Operands
[symbol]	FDFIELD	[{hmrgdl / prevdr + 1 / dl / DUMMY}] [{compIng / LNG (d) / dr}] [,CTR = ((d,op[, FIELD]) [, (d,op[, FIELD])] . . .)] [,IND = ((d,logexp) [, (d,logexp)] . . .)] [,CYCLE = ([d] [, limit] [, target])] [,SAVELOC = {NO / YES / d}] [,SOURCE = (origin[,qualifier] . . .)] [,KIND = {U / A / N / AN / K}] [,SELFCHK = {NO / ({10 / 11} [, GENONLY])}] [,COUNT = {([MIN ,] {1 / d1} ,[MAX,] {compmax / d2}) / d}] [,COMPARE = ([FIELD,] comparopr,comparand [, {AND / OR} ,[FIELD,] comparopr,comparand] . . .)] [, SINK = ([(destination[,qualifier] . . .)] [,[(destination[,qualifier] . . .)]] . . .)] [,JUSTIFY = ([justcode] [,[justcode]] . . .)] [,FILL = (['char'] [,['char']] . . .)] [,UL = ([{NO / YES}] [,[{NO / YES}]] . . .)] [,PICTURE = (['picturespec'] [,['picturespec ']] . . .)] [,BATCH = d]
[symbol]	FDCTRL	[d] [,IF = (logterm[, {AND / OR} ,logterm] . . .)] [,CTR = (({d / Xn} [,CLR] [,op,opnd] . . . [, ({d / Xn} [,CLR] [,op,opnd] . . .)] . . .)] [;IND = ((d, ON / OFF / INV) [, (d, X1,compar[,X2])] [, (d,X2,compar[,X1])] [,TOTAL = ((d,'fid' ,CTR (d)) [, (d, 'fid' ,CTR (d))] . . .)] [,COMMAND = ((cmndgrp) [, (cmndgrp)] . . .)] [,GOTO = target] [,CYCLE = ([d] [,limit] [,target])] [,SAVELOC = {NO / YES / d}]
[symbol]	FDLOAD	KEYLEN = d ,DATALEN = d[,ENDCHAR = 'string']
[symbol]	FDEND	[,] (This macro has no operands.)

*Default is to the system that assembled the FDP.

This sample program does not attempt to show all the permissible uses of the FD macros, nor even the most concise way to write an FDP. It is presented simply to illustrate some of the functions and coding techniques that can be used to describe the sample form presented in Appendix A of the *IBM 3735 Programmable Buffered Terminal Concept and Application* publication, Order No. GA27-3043.

```
F026     TITLE 'FID 026 -- GENERAL WHOLESALE COMPANY SALES·INVOICE'
FDFORM   FDFORM 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
*
         FDCTRL CTR=(*,CLR),              CLEAR ALL COUNTERS (*=ALL)     X
               COMMAND=READ(RDR)          READ A CARD FROM 5496
*
         FDCTRL IF=EOF(RDR),              WAS THERE A CARD AT 5496       X
               COMMAND=CANCEL             IF NOT, CANCEL FORM
*
         FDPAGE 1,                        THIS IS A ONE-PAGE FORM        X
               HEIGHT=66,                                                X
               VMRG=(12,58),                                             X
               WIDTH=85,                  LINE WIDTH IS CONSTANT         X
               HMRG=(12,80)               HOR MARGINS ARE CONSTANT
*
         FDLINE 12                        SOLD-TO NAME LINE
*
NAME1    FDFIELD 12,31,                   NAME POSITION                  X
               SOURCE=(RDR,1,20),         GET FROM CARD IMAGE            X
               SINK=PRT                   PUT TO SELECTRIC II
*
         FDLINE 13                        SOLD-TO STREET ADDRESS LINE
*
ADDRESS1 FDFIELD 12,31,                   STREET ADDRESS POSITION        X
               SOURCE=(RDR,21,40),        GET FROM CARD IMAGE            X
               SINK=PRT                   PUT TO SELECTRIC II
*
         FDLINE 14                        SOLD-TO CITY/STATE/ZIP LINE
*
ADDRESS2 FDFIELD 12,31,                   CITY/STATE POSITION            X
               SOURCE=(RDR,41,60),        GET FROM CARD IMAGE            X
               SINK=PRT                   PUT TO SELECTRIC II
*
ZIPCODE1 FDFIELD 34,38,                   ZIP POSITION                   X
               SOURCE=(RDR,61,65),        GET FROM CARD IMAGE            X
               KIND=N,                    MUST BE ALL NUMERIC CHARACTERS X
               SINK=PRT                   PUT TO SELECTRIC II
```

```
*
         FDFIELD ,                          CONTROL-CHARACTER INPUT          X
               SOURCE=(KBD,OPTIONAL),       GET FROM KEYBOARD                X
               COUNT=1,                     SINGLE CHARACTER                 X
               COMPARE=(FIELD,EQ,'K'),      IF ENTRY MADE, MUST BE 'K'       X
               IND=(1,FIELD,EQ,'K'),        FLAG ENTRY OF 'K'                X
               SINK=NULL                    NO OUTPUT
*
         FDCTRL IF=IND(1),GOTO=KSHIPTO      ON MEANS OPR. WILL KEY SHIP-TO
*
CSHIPTO  FDLINE 16                          SHIP-TO NAME LINE (CARD)
*
NAME2C   FDFIELD 12,31,                     NAME POSITION                    X
               SOURCE=(RDR,1,20),           GET FROM CARD IMAGE (AS BEFORE)X
               SINK=PRT                     PUT TO SELECTRIC II
*
         FDLINE 17                          SHIP-TO STREET ADDRESS LINE
*
ADDR3C   FDFIELD 12,31,                     STREET ADDRESS POSITION          X
               SOURCE=(RDR,21,40),          GET FROM CARD IMAGE (AS BEFORE)X
               SINK=PRT                     PUT TO SELECTRIC II
*
         FDLINE 18                          SHIP-TO CITY/STATE/ZIP LINE
*
ADDR4C   FDFIELD 12,31,                     CITY/STATE POSITION              X
               SOURCE=(RDR,41,60),          GET FROM CARD IMAGE (AS BEFORE)X
               SINK=PRT                     PUT TO SELECTRIC II
*
ZIP2C    FDFIELD 34,38,                     ZIP POSITION (KNOWN TO BE O.K.)X
               SOURCE=(RDR,61,65),          GET FROM CARD IMAGE (AS BEFORE)X
               SINK=PRT                     PUT TO SELECTRIC II
*
         FDCTRL GOTO=HEADLINE               BRANCH OVER OPERATOR ENTRY
*
KSHIPTO  FDLINE 16                          SHIP-TO NAME LINE (KEY)
*
NAME2    FDFIELD 12,31,                     NAME POSITION                    X
               SOURCE=KBD,                  GET FROM KEYBOARD                X
               SINK=PRT                     PUT TO SELECTRIC II
*
         FDLINE 17                          SHIP-TO STREET ADDRESS LINE
*
ADDRESS3 FDFIELD 12,31,                     STREET ADDRESS POSITION          X
               SOURCE=KBD,                  GET FROM KEYBOARD                X
               SINK=PRT                     PUT TO SELECTRIC II
*
         FDLINE 18                          SHIP-TO CITY/STATE/ZIP LINE
*
ADDRESS4 FDFIELD 12,31,                     CITY/STATE POSITION              X
               SOURCE=KBD,                  GET FROM KEYBOARD                X
               SINK=PRT                     PUT TO SELECTRIC II
*
ZIPCODE2 FDFIELD 34,38,                         ZIP POSITION                 X
               SOURCE=(KBD,CPTIONAL,AUTOEOF),   GET FROM KEYBOARD           X
               COUNT=5,                         MUST BE 5 CHARS (IF ANY)    X
               KIND=N,                          MUST BE ALL NUMERIC CHARSX
               SINK=PRT                         PUT TO SELECTRIC II
```

```
*
HEADLINE FDLINE 23                          HEADING LINE
*
DATEFLD  FDFIELD 12,19,                     DATE POSITION                    X
               SOURCE=(STG,1,6),            GET FROM STORAGE (PRESET)        X
               SINK=PRT,                    PUT TO SELECTRIC II              X
               PICTURE='Y9/99/99',          MM/DD/YY FORMAT                  X
               KIND=N                       NUMERIC DATA ONLY
*
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')  PICTURE SPECIFICATIONS
*
         FDCTRL CTR=(1,ADD,1)               INCREMENT INVOICE NUMBER
*
         FDFIELD ,                          RETAIN NEW NUMBER (NO POSITION) X
               SOURCE=CTR(1),               GET FROM COUNTER (LOW ORDER)    X
               SINK=(STG,7,13)              PUT TO STORAGE
*
CUSTNO   FDFIELD 30,34,                     CUSTOMER NUMBER POSITION         X
               SOURCE=(RDR,66,70),          GET FROM CARD IMAGE              X
               KIND=N,                      MUST BE ALL NUMERIC CHARACTERS   X
               SINK=(PRT,TMT)               PUT TO SEL. II AND TRANSMIT
*
TERRITRY FDFIELD 36,38,                     TERRITORY NUMBER POSITION        X
               KIND=N,                      MUST BE ALL NUMERIC CHARACTERS   X
               SOURCE=(RDR,71,73),          GET FROM CARD IMAGE              X
               SINK=PRT                     PUT TO SELECTRIC II
*
SALESMAN FDFIELD 40,53,                     SALESMAN SURNAME POSITION        X
               SOURCE=(RDR,74,83),          GET FROM CARD IMAGE              X
               KIND=A,                      MUST BE ALL ALPHABETIC CHARS     X
               SINK=PRT                     PUT TO SELECTRIC II
*
TERMS    FDFIELD 55,59,                     TERMS OF SALE POSITION           X
               SOURCE=(RDR,84,87),          GET FROM CARD IMAGE              X
               KIND=N,                      MUST BE ALL NUMERIC CHARACTERS   X
               SINK=PRT,                    PUT TO SELECTRIC II              X
               PICTURE='99/99'              INSERT SLANT IN CENTER
*
         FDFIELD ,                          OPERATOR SELECTION OF SHIP-VIA   X
               SOURCE=(KBD,AUTOEOF),        GET FROM KEYBOARD                X
               COUNT=1,                     SINGLE CHARACTER                 X
               COMPARE=(EQ,'A',OR,EQ,'D',OR,EQ,'P',OR,EQ,'R',OR,              X
               EQ,'T'),                     VALIDATE OPERATOR RESPONSE       X
               IND=((2,EQ,'D'),(3,EQ,'P'),(4,EQ,'R'),(5,EQ,'T')),            X
               SINK=TMT                     TELL CPU WHICH WAY
*
*              BRANCH TABLE TO CONTROL PRINTING OF SHIP-VIA FIELD
*
         FDCTRL IF=IND(2),GOTO=DELIVER      ON IF 'D' KEYED
*
         FDCTRL IF=IND(3),GCTO=PICKUP       ON IF 'P' KEYED
*
         FDCTRL IF=IND(4),GOTO=RREXP        ON IF 'R' KEYED
```

```
*
         FDCTRL IF=IND(5),GOTO=TRUCK      ON IF 'T' KEYED
*
*                                         NONE ON IF 'A' KEYED, FALL THRU
*
AIR      FDFIELD 61,80,                   SHIP-VIA POSITION                X
               SOURCE='AIR FREIGHT',      GET LITERAL STRING               X
               SINK=PRT,                  PUT TO SELECTRIC II              X
               JUSTIFY=C                  CENTERED
*
         FDCTRL GOTO=BODY                 PROCEED TO BODY OF FORM
*
DELIVER  FDFIELD 61,80,                   SHIP-VIA POSITION                X
               SOURCE='DELIVER',          GET LITERAL STRING               X
               SINK=PRT,                  PUT TO SELECTRIC II              X
               JUSTIFY=C                  CENTERED
*
         FDCTRL GOTO=BODY                 PROCEED TO BODY OF FORM
*
PICKUP   FDFIELD 61,80,                   SHIP-VIA POSITION                X
               SOURCE='PICK UP',          GET LITERAL STRING               X
               SINK=PRT,                  PUT TO SELECTRIC II              X
               JUSTIFY=C                  CENTERED
*
         FDCTRL GOTO=BODY                 PROCEED TO BODY OF FORM
*
RREXP    FDFIELD 61,80,                   SHIP-VIA POSITION                X
               SOURCE='RAILWAY EXPRESS',  GET LITERAL STRING               X
               SINK=PRT,                  PUT TO SELECTRIC II              X
               JUSTIFY=C                  CENTER
*
         FDCTRL GOTO=BODY                 PROCEED TO BODY OF FORM
*
TRUCK    FDFIELD 61,80,                   SHIP-VIA POSITION                X
               SOURCE='TRUCK',            GET LITERAL STRING               X
               SINK=PRT,                  PUT TO SELECTRIC II              X
               JUSTIFY=C                  CENTER
*
BODY     FDLINE 28,CYCLE=(28,AMOUNT,GROSS) PROVIDES FOR LINES 28-55
*
ORDERQTY FDFIELD 12,14,                        ORDER QUANTITY LOCATION     X
               SOURCE=KBD,                     GET FROM KEYBOARD           X
               KIND=N,                         MUST BE ALL NUMERIC CHARS   X
               COUNT=(1,3),                    1, 2, OR 3 CHARACTERS       X
               SINK=(PRT,CTR(2),CTR(3),TMT),   CTRS ARE QO SCR & BO SCR    X
               JUSTIFY=R,                      RIGHT JUSTIFY ON SEL. II    X
               FILL=' '                        WITH LEADING BLANKS
*
SHIPQTY  FDFIELD 16,18,                   SHIP QUANTITY LOCATION           X
               SOURCE=(KBD,OPTIONAL),     GET FROM KBD (OPERATOR OPTION)   X
               COUNT=(1,3),               1, 2, OR 3 CHARACTERS            X
               KIND=N,                    MUST BE ALL NUMERIC CHARACTERS   X
               CTR=(3,SUB),               SUBTRACT SHIP FROM QO            X
               SINK=(PRT,TMT),            PUT SHIP TO SEL. II & TRANSMIT   X
               JUSTIFY=R,                 RIGHT JUSTIFIED                  X
               FILL=' '                   WITH LEADING BLANKS
```

```
*
BACKORD  FDFIELD 20,22,                  BACKORDERED QUANTITY LOCATION  X
               SOURCE=CTR(3),            GET COMPUTED BO QUANTITY       X
               SINK=(PRT,TMT),           PUT TO SEL. II AND TRANSMIT    X
               PICTURE='ZZZ'             SUPPRESS LEADING ZEROS SEL. II
*
MEASURE  FDFIELD 24,25,                  MEASURING UNIT POSITION        X
               SOURCE=(KBD,AUTOEOF),     GET FROM KEYBOARD              X
               COUNT=2,                  EXACTLY TWO CHARACTERS         X
               COMPARE=(EQ,'EA',OR,EQ,'DZ',OR,EQ,'GR',OR,EQ,'LB'), UNITX
               SINK=PRT                  PUT TO SELECTRIC II
*
ITEMNO   FDFIELD 27,31,                  ITEM NUMBER POSITION           X
               SOURCE=(KBD,AUTOEOF),     GET FROM KEYBOARD              X
               COUNT=5,                  EXACTLY FIVE CHARACTERS        X
               KIND=N,                   MUST ALL BE NUMERIC            X
               SELFCHK=10,               MODULO 10 TEST AGAINST 5TH CHARX
               SINK=(PRT,TMT)            PUT TO SEL. II AND TRANSMIT
*
DESCRIPT FDFIELD 33,62,                  DESCRIPTION POSITION           X
               SOURCE=KBD,               GET KEYBOARD                   X
               COUNT=(2,30),             2-30 CHARACTERS (ANY)          X
               SINK=PRT                  PUT TO SELECTRIC II
*
PRICE    FDFIELD 64,70,                  PRICE POSITION                 X
               SOURCE=KBD,               GET KEYBOARD                   X
               COUNT=(2,5),              2-5 CHARACTERS                 X
               KIND=N,                   MUST ALL BE NUMERIC            X
               CTR=(2,MPY),              MULTIPLY QO BY PRICE IN CENTS  X
               SINK=(PRT,TMT),           PUT TO SEL. II AND TRANSMIT    X
               PICTURE='$***V.99'        SHOW * FILL UNDER $100
*
AMOUNT   FDFIELD 72,80,                  AMOUNT POSITION                X
               SOURCE=CTR(2),            GET COMPUTED AMOUNT IN CENTS   X
               CTR=(4,ADD),              ACCUMULATE AMOUNT IN CENTS     X
               SINK=(PRT,TMT),           PUT TO SEL. II AND TRANSMIT    X
               PICTURE='$*,***V.99'      SHOW * FILL UNDER $1,000
*
GROSS    FDLINE 58                       FOOTING LINE
*
GROSSAMT FDFIELD 33,43,                  GROSS AMOUNT POSITION          X
               SOURCE=CTR(4),            GET ACCUMULATED AMOUNT IN CENTSX
               SINK=(PRT,TMT),           PUT TO SEL. II AND TRANSMIT    X
               PICTURE=('$***,***V.99','$***,***V.99')  PICTURES
*
TAXPERCT FDFIELD 45,48,                  TAX RATE POSITION              X
               SOURCE=(RDR,88,90),       GET FROM CARD IMAGE IN 1/100THSX
               KIND=N,                   MUST BE ALL NUMERIC CHARACTERS X
               PICTURE='9.99',           SHOW AS PERCENT.HUNDREDTHS     X
               SINK=(PRT,TMT,CTR(5))     PUT TO SEL, TMT, AND CTR(5)
*
         FDCTRL CTR=(5,MPY,CTR(4),DVR,10000) TAX IN CENTS IN CTR(5)
*
TAXAMT   FDFIELD 50,58,                  TAX AMOUNT POSITION            X
               SOURCE=CTR(5),            GET FROM SCRATCH COUNTER       X
               CTR=(4,ADD),              ADD TAX AMOUNT TO GROSS AMOUNT X
```

```
                SINK=(PRT,TMT),          PUT TO SEL. II AND TRANSMIT    X
                PICTURE='$*,***V.99'     SHOW * FILL WHEN UNDER $1,000
*
SHIPCHG   FDFIELD 60,68,                 SHIPPING CHARGE POSITION       X
                SOURCE=(KBD,OPTIONAL),   GET FROM KBD (OPERATOR OPTION) X
                KIND=N,                  MUST ALL BE NUMERIC            X
                COUNT=(1,6),             1-6 CHARACTERS                 X
                CTR=(4,ADD),             ADD SHIP CHARGE TO GROSS+TAX   X
                SINK=(PRT,TMT),          PUT TO SEL. II AND TRANSMIT    X
                PICTURE='$*,***V.99'     SHOW * FILL WHEN UNDER $1,000
*
INVTOTAL  FDFIELD 70,80,                 INVOICE TOTAL POSITION         X
                SOURCE=CTR(4),           GET FROM ACCUMULATOR           X
                SINK=(PRT,TMT),          PUT TO SEL. II AND TRANSMIT    X
                PICTURE='$***,***V.99'   SHOW * FILL WHEN UNDER $100,000
*
          FDLINE 60
FIDNO     FDFIELD 12,14,                 FDP IDENTIFICATION             X
                SOURCE=FID,              RETRIEVE FID                   X
                SINK=PRT                 TO SELECTRIC ONLY
*
          FDFIELD 15,15,SOURCE='/',      EMIT SLASH TO SEPARATE         X
                SINK=PRT                 TO SELECTRIC ONLY
*
RECSEQNO  FDFIELD 16,18,                 RECORD SEQUENCE NUMBER         X
                SOURCE=RSN,              GET RECORD SEQUENCE NUMBER     X
                SINK=(PRT,TMT),          PUT TO SEL. II AND TRANSMIT    X
                PICTURE='999'            ONLY THE THREE DIGITS
*
          FDEND ,FID 026 -- GENERAL WHOLESALE COMPANY SALES INVOICE
```

Appendix D. Form Description Macro Instruction MNOTE Messages

The MNOTE messages issued by the Form Description macros during the assembly of a form description program (FDP) are of three types. The first type have severity codes of asterisk (*) and are informative in nature. They describe FDP parameters such as page height, line margins, and so forth. These MNOTEs should be analyzed to ensure that the FDP was assembled with the desired parameters. Also included in this type are FDP logic MNOTEs that describe the beginning and end of paths and segments. These MNOTEs may be used to trace the logical flow of execution within the FDP if unusual or incorrect results are observed at the terminal.

The second type of MNOTEs have severity codes of zero (0) and are warnings that some unusual condition or parameter has been found. These MNOTEs should be carefully analyzed to ensure that the condition or parameter is actually what is desired. Used in conjunction with the path and segment MNOTEs, these warnings may be used to find oversights and possible logic errors in the FDP. If the action indicated is desired, the FDP does not need to be reassembled.

The third type of MNOTEs have severity codes of eight (8) and indicate that a severe error has been encountered and further code generation for the FDP is suppressed. Although actual assembly is ended, syntax checking of operands continues as though no error had been found. These MNOTEs report invalid codings of operands, parameters not within allowable ranges, and so forth. If an MNOTE with severity code of eight is issued during assembly of an FDP, the FDP is flagged as invalid, and if used as input to the Form Description utility, the FDP will be rejected. All errors associated with severity eight MNOTEs must be corrected and the FDP assembled again before a valid FDP can be generated.

There is one macro generated message that may appear during execution of a file load FDP:

FDP: nnn NOT PLAYBACK MODE FILE LOAD FDP CANCELED

This message is issued at the 3735 terminal when a file load FDP is not executed in playback mode; nnn indicates the form ID of the FDP.

Note: The DOS FD macros replace the prefix IDF with IJLF for each MNOTE message.

*,IDF101 FORM NAME IS name

Explanation: This message reports the name entry that was coded on the FDFORM macro for this FDP. The name is the identifier by which the FDP will be stored in your data set when the FDP is processed by the FD utility.

System Action: None. Assembly continues.

Programmer Response: Make sure that the name described is desired and correct.

*,IDF102 FORM ID IS ddd

Explanation: This message reports the value coded in the FDFORM FID operand for this FDP. The number (ddd) is the FDP identifier that the 3735 operator uses to request the FDP at the terminal.

System Action: None. Assembly continues.

Programmer Response: Make sure that the FID value described is desired and correct.

*,IDF103 TABS SET AT COLUMNS

Explanation: This message reports the setting of tabs at the specified columns.

System Action: None. Assembly continues.

Programmer Response: Make sure that the tabs described are desired and correct.

*,IDF104 PAGE pp INCLUDES LINE n1
*, THROUGH n2 WITHIN THE FORM

Explanation: This message reports the cumulative starting (n1) and ending (n2) line of each page (pp) in the FDP, relative to the entire form, so that you can verify proper page positioning.

System Action: None. Assembly continues.

Programmer Response: Make sure that the lines described are desired and correct.

*,IDF105 FDEND NOT NEEDED

Explanation: A superfluous FDEND macro has been found and is ignored. This error may occur if the FDFORM macro was missing or incorrect.

System Action: None. Assembly continues.

Programmer Response: Remove the extra FDEND if you have to assemble the program again because of other errors.

*,IDF106 CTR (d) USED AS ACCUMULATOR

Explanation: The specified counter was used to accumulate data in FDFIELD or FDCTRL counter operations.

System Action: None. Assembly continues.

Programmer Response: Make sure that the counter described is desired and correct.

*,IDF107 CTR (d) USED AS GENERATOR

Explanation: The specified counter was used as a data source in FDCTRL counter operations.

System Action: None. Assembly continues.

Programmer Response: Make sure that the counter described is desired and correct.

*,IDF108 STARTING PATH p

Explanation: This message reports the beginning of the specified logical path.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF109 STARTING SEGMENT s

Explanation: This message reports the beginning of the specified logical segment.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF110 END OF SEGMENT s

Explanation: This message reports the end of the specified logical segment.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF111 END OF PATH p

Explanation: This message reports the end of the specified logical path.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF112 INDICATORS USED IN PATH p

Explanation: This message heads a list of the indicators used in the specified path. Each indicator identified in the list was set, reset, or tested in this path.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF113 IND (d)

Explanation: This message repeats as often as necessary to report all the indicators used in the path specified in message IDF112.

System Action: None. Assembly continues.

Programmer Response: Make sure that the indicators described are desired and correct.

*,IDF114 COUNTERS USED IN PATH p

Explanation: This message heads a list of the counters used in the specified path. Each counter identified in the list was used as a data SOURCE or SINK in FDFIELD operations, or was used for arithmetic computations in FDCTRL counter operations.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF115 CTR (d)

Explanation: This message repeats as often as necessary to report all the counters used in the path specified in message IDF114.

System Action: None. Assembly continues.

Programmer Response: Make sure that the counters described are desired and correct.

*,IDF116 BUFFERS USED IN PATH p

Explanation: This message heads a list of the buffers used in the specified path. Each buffer identified in the list was used as a data SOURCE or SINK in this path.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF118 THIS SEGMENT ENTERED FROM SEGMENT s

Explanation: This message repeats as often as necessary to report all entries to this segment from other segments. This information is useful when analyzing FDP logic if unusual or incorrect operation occurs at the 3735.

System Action: None. Assembly continues.

Programmer Response: Make sure that the entry described is desired and correct.

*,IDF119 THIS PATH ENTERED
*, FROM SEGMENT s OF PATH p

Explanation: This message repeats as often as necessary to report all entries to this path from previous paths and segments. This information is useful when analyzing FDP logic if unusual or incorrect FDP operation occurs at the 3735.

System Action: None. Assembly continues.

Programmer Response: Make sure that the entry described is desired and correct.

*,IDF120 macro1 LEVEL ATTRIBUTES
*, CHANGED FROM macro2

Explanation: This message heads a list of parameters that were changed when the transition was made from macro1 to macro2.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF122 param IS dd

Explanation: This message describes the values of WIDTH, HEIGHT, and MRGSTOP.

System Action: None. Assembly continues.

Programmer Response: Make sure that the values described are desired and correct.

*,IDF123 margin MARGIN IS dd

Explanation: This message describes the values of the LEFT, RIGHT, TOP, and BOTTOM margins.

System Action: None. Assembly continues.

Programmer Response: Make sure that the margins described are desired and correct.

*,IDF124 KIND IS kind

Explanation: This message describes the KIND attribute associated with the current SOURCE.

System Action: None. Assembly continues.

Programmer Response: Make sure that the attribute described is desired and correct.

*,IDF125 SINK d IS type [,d] [DELIMIT]

Explanation: This message describes the assignment of each SINK (type). For buffered sinks, the message also reports the starting position (either absolute or index counter) in the buffer where data will be stored (d). For the I/O buffer, DELIMIT indicates that a record separator is to be added to the characters being placed in the buffer.

System Action: None. Assembly continues.

Programmer Response: Make sure that the sink described is desired and correct.

*,IDF126 param FOR SINK d IS value

Explanation: This message describes the FILL, JUSTIFY, and UNDERLINE, options for each SINK.

System Action: None. Assembly continues.

Programmer Response: Make sure that the options described are desired and correct.

*,IDF127 SELF-CHECK OPTION IS option

Explanation: This message describes the self-check option associated with the current SOURCE.

System Action: None. Assembly continues.

Programmer Response: Make sure that the option described is desired and correct.

*,IDF128 SOURCE IS type [,option] . . .

Explanation: This message describes the current SOURCE (type) and its options, if any (option). For buffers, the option specifies the starting position in the buffer where data is located.

System Action: None. Assembly continues.

Programmer Response: Make sure that the value described is desired and correct.

*,IDF129 IND d operation

Explanation: This message reports that the specified indicator (d) was SET, TESTED, or INVERTED.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action described is desired and correct. (Note that SET means to set ON or to set OFF.)

*,IDF130 buffer

Explanation: This message repeats as often as necessary to report all the buffers used in the path specified in message IDF116.

System Action: None. Assembly continues.

Programmer Response: Make sure that the buffers described are desired and correct.

*,IDF132 AT END OF CYCLE PRINT ELEMENT WAS
*, POSITIONED ON LINE dd OF FORM

Explanation: This message reports the line on which the Selectric ® print element was positioned when the current CYCLE ended. This information is useful for verifying that the position was as intended.

System Action: None. Assembly continues.

Programmer Response: Make sure that the value described is desired and correct.

*,IDF133 NO TERMINATING ERRORS FOUND IN THIS FDP

Explanation: This message indicates that this FDP assembled without an FD macro-detected error. If no Assembler errors are noted, the FDP may be used as input to the Form Description utility program.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF134 SINK d OUTPUT COUNT IS digits

Explanation: This message describes the count of output characters (digits) for the current sink (d). The count is useful for verifying that the results are as intended.

System Action: None. Assembly continues.

Programmer Response: Make sure that the count described is desired and correct.

*,IDF135 PICTURE WAS USED FOR FORMATTING
*, OUTPUT OF SINK d

Explanation: This message indicates that the PICTURE specification that was used to format the indicated sink (d).

System Action: None. Assembly continues.

Programmer Response: Make sure that the action described is desired and correct.

*,IDF136 THIS SEGMENT BRANCHES TO SEGMENT ss OF PATH pp

Explanation: This message describes the logic flow as FDP control leaves this code segment, as defined by the values of ss and pp. This information is useful when you are trying to locate logic errors in your FDP.

System Action: None. Assembly continues.

Programmer Response: Make sure that the branch described is desired and correct.

*,IDF137 type FEATURE INDICATOR TESTED

Explanation: This message describes the feature indicator that was tested (5496, 3286, IDR, or CCR).

System Action: None. Assembly continues.

Programmer Response: Make sure that the indicator tested is desired and correct.

*,IDF138 PACKING OPTION IS option

Explanation: This message describes the packing option selected.

System Action: None. Assembly continues.

Programmer Response: Make sure that the option described is desired and correct.

*,IDF139 LINE NUMBER IS dd

Explanation: This message identifies the current line number within the form.

System Action: None. Assembly continues.

Programmer Response: Make sure that the line number described is desired and correct.

*,IDF140 ind SPECIAL INDICATOR SET OR TESTED

Explanation: The specified indicator (ind) was set or tested in the current path.

System Action: None. Assembly continues.

Programmer Response: Make sure that the indicator set or tested is desired and correct. (Note that SET means to set ON or to set OFF.)

*,IDF141 POSITION LIMITS FOR LPB ARE 1 AND n

Explanation: This message describes the limits that were specified for the line printer buffer (LPB) in the BUFFERS operand of the FDFORM macro instruction.

System Action: None. Assembly continues.

Programmer Response: Make sure that the limits described are desired and correct.

*,IDF142 SOURCE/SINK OPTION FOR 5496 IS RPB

Explanation: This message confirms that RPB was coded in the BUFFERS operand of the FDFORM macro instruction.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action described is desired and correct.

*,IDF143 SELECTRIC II PRINT REGION BEGINS AT
*, COLUMN a, ENDS AT COLUMN b

Explanation: The printing region for the Selectric ® printer begins at column a and ends at column b. When field processing is complete, the print element is positioned in column b+1.

System Action: None. Assembly continues.

Programmer Response: Make sure that the limits described are desired and correct.

*,IDF144 TMT DATA FORMAT IS [ZERO OR] [a TO] b CHARACTERS
*, [DELIMITED BY SEPARATOR]

Explanation: This message describes the number of characters that will be transmitted to the CPU. The options appear according to the way in which the field, source, data count, and PACKING operand were specified.

System Action: None. Assembly continues.

Programmer Response: Make sure that the values described are desired and correct.

*,IDF145 SOURCE CHARACTER COUNT IS [ZERO OR] [a TO] b

Explanation: This message describes the number of characters that are expected from the current SOURCE. The options appear according to the way the SOURCE and COUNT operands were specified.

System Action: None. Assembly continues.

Programmer Response: Make sure that the values described are desired and correct.

*,IDF146 FORM DESCRIPTION PROGRAM SPECIFIED SELECTRIC II FORM
*, HAVING nn LINES [AND 3286 FORM HAVING mm LINES]

Explanation: This message describes the total number of Selectric ® printer lines and 3286 printer lines (if any) in this form.

System Action: None. Assembly continues.

Programmer Response: Make sure that the values described are desired and correct.

*,IDF147 SUMMARY OF FDP-GENERATED DATA

Explanation: This message begins a description of the number of host operating system (OS or DOS) disk storage blocks required for this FDP, and the required 3735 disk storage (in sectors).

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF148 OBJECT OUTPUT IS sys FORMAT

Explanation: This message reports the object output format as either CS or DOS for input to the appropriate FD utility.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF149 INDEX COUNTERS USED IN PATH p

Explanation: This message is the heading for the reporting of the index counters used in the previous path p.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF150 ctr] AND X2]

Explanation: This message reports the index counters, X1 and/or X2, used in the previous path and may aid in verifying the correct analysis of counter coding.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF151 KEYLEN IS nn

Explanation: This message indicates the valid key length for the file records described by an FDLOAD macro.

System Action: None. Assembly continued.

Programmer Response: None.

*,IDF152 DATALEN IS nn

Explanation: This message indicates the valid data length for the file records described by an FDLOAD macro.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF153 INDEX COUNTER Xn USED AS A GENERATOR
*, IN CTR SUBOPERAND n

Explanation: The specified index counter (X1 or X2) has been used as a source of numeric information in the CTR operand of an FDCTRL macro.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF154 formname IS A FILE LOAD FDP

Explanation: The displayed form name identifies the current FDP as a file load FDP as specified by a MODE = LOAD operand of an FDFORM macro.

System Action: None. Assembly continues.

Programmer Response: None.

*,IDF155 INDEX COUNTERS X1 AND X2 COMPARED

Explanation: This message reports that the index counters X1 and X2 have been compared in the FDCTRL IND operand. If the compare condition is set, the specified indicator will be set on.

System Action: None. Assembly continues.

Programmer Respones: None.

0,IDF400 FDFORM MUST START FORM

Explanation: The first macro encountered either at the beginning of the assembly or after a previous FDEND was not FDFORM. The current macro is ignored.

System Action: None. Assembly continues.

Programmer Response: Start each FDP with an FDFORM macro.

0,IDF401 ELEMENT n OF HTAB OPERAND INVALID

Explanation: The specified element of the HTAB operand is non-decimal. The non-decimal element is ignored.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF402 CTR (d) MAY NOT HAVE BEEN USED AS
0, OUTPUT SINCE PRIOR INPUT

Explanation: The preceding operation in this path on the specified counter was not output, but the current operation is input. If two consecutive input operations are being performed, the current contents of the counter may be destroyed inadvertently.

System Action: None. Assembly continues.

Programmer Response: Make sure that the counter is used for output in a previous path, that its contents are of no value at this point in processing, or that it is being used as an accumulator.

0,IDF403 CTR (d) MAY NOT HAVE BEEN PROPERLY LOADED
0, BEFORE CURRENT OUTPUT

Explanation: The specified counter is used for output in this path, but the preceding operation on it in this path was not input. The contents of the counter may be erroneous.

System Action: None. Assembly continues.

Programmer Response: Make sure that the counter is loaded in this path or in a previous path.

0,IDF404 IND d MAY NOT HAVE BEEN TESTED SINCE SET

Explanation: The specified indicator is set in this path, but it may not have been tested since it was last set. The indicator status may be destroyed.

System Action: None. Assembly continues.

Programmer Response: Make sure that the indicator is tested in a previous path or that its status is of no value at this point in processing.

0,IDF405 IND d MAY NOT HAVE BEEN SET BEFORE TEST

Explanation: The specified indicator is tested in this path, but the preceding operation in this path did not set it. The indicator status may be erroneous.

System Action: None. Assembly continues.

Programmer Response: Make sure that the indicator is set in this path or a previous path.

0,IDF406 CTR (d) MAY NOT HAVE BEEN CLEARED
0, BEFORE FIRST INPUT

Explanation: The specified counter is used in this path as an accumulator, but it may not have been cleared previously.

System Action: None. Assembly continues.

Programmer Response: Make sure that the counter is properly cleared before beginning the accumulation.

0,IDF407 MESSAGE USED VERTICAL SPACING

Explanation: Vertical spacing has been specified in the operator message, and form positioning at the 3735 has changed as a result.

System Action: None. Assembly continues.

Programmer Response: Make sure that the 3735 operator will make any necessary adjustments to the form position before processing the form.

0,IDF408 MESSAGE USED HORIZONTAL TABS

Explanation: Horizontal tabs were specified in the operator message, but the 3735 tab setting routine has not been performed yet.

System Action: None. Assembly continues.

Programmer Response: Make sure that the proper tabs will be set at the 3735 before this operator message is printed, or remove the horizontal tab specifications.

0,IDF409 CHAINING IN EFFECT, op OPERAND IGNORED

Explanation: Chaining of a preceding operand is in effect. The operand specified (op) is ignored.

System Action: None. Assembly continues.

Programmer Response: Locate and correct the error before assembling the program a again.

0,IDF410 op IGNORED FOR DUMMY FIELD/LOAD MODE/SOURCE X1/X2

Explanation: The specified operand (op) is ignored because the current field is coded as a DUMMY field or because the MODE and SOURCE specifications dictate that operands are not required.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF411 SUBOPERANDS AFTER SUBOPERAND n OF
0, op OPERAND IGNORED

Explanation: The specified operand (op) has too many suboperands coded. Those coded after the indicated suboperand (n) are ignored.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF412 EXCESS CHARACTERS OF op
0, SUBOPERAND n IGNORED

Explanation: Excess characters have been found in the specified suboperand (n). The excess characters are ignored.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF413 POSSIBLE DUPLICATION OF EARLIER parm1
0, IN THIS parm 2

Explanation: This message reports the possible duplication of a PAGE within the FORM, or of a LINE within the PAGE. This situation may occur when two or more pages or lines have the same page or line number. This may not be an error if, for example, a

line is defined with the same line number twice, but only one of the duplicate lines is executed in a single execution of the FDP.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action described is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF414 { IND/CTR } d MAY NOT HAVE BEEN UNCONDITIONALLY
0, SET IN FIRST OPERATION

Explanation: The specified indicator or counter is being tested or used for output in this path, but it is possible that no setting has been performed.

System Action: None. Assembly continues.

Programmer Response: Make sure that the indicator or counter was set in a previous path or perform the setting in this path.

0,IDF415 UNPRINTABLE CHARACTER IN CHARACTER STRING

Explanation: A character that cannot be printed on any 3735 has been found in a character string. The unprintable characters are the logical NOT symbol (¬), the logical OR symbol (|), and the cent symbol (¢).

System Action: None. Assembly continues.

Programmer Response: Make sure that the character in question is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF416 CHARACTER NOT PRINTABLE ON ASCII 3735
0, FOUND IN CHARACTER STRING

Explanation: A character that cannot be printed on an ASCII 3735 has been found in a character string. (The unprintable character is the plus-minus.)

System Action: None. Assembly continues.

Programmer Response: Make sure that the character in question is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF417 CHARACTER NOT PRINTABLE ON EBCDIC 3735
0, FOUND IN CHARACTER STRING

Explanation: A character that cannot be printed on an EBCDIC 3735 has been found in a character string. (The unprintable characters are the circumflex and the reverse slash.)

System Action: None. Assembly continues.

Programmer Response: Make sure that the character in question is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF418 DEAD CODE, CYCLE IGNORED

Explanation: A CYCLE has been found in code that cannot be executed. The CYCLE is ignored. This situation may occur when code following an unconditional GOTO does not have a name entry on some macro that can be used as an entry point.

System Action: None. Assembly continues.

Programmer Response: Either remove the dead code or provide some entry to it before assembling the program again.

0,IDF419 CYCLE WITHIN A CYCLE OR SUMMARY BLOCK IGNORED

Explanation: A CYCLE has been detected within another CYCLE or within a summary block. Cycles may not be nested or coded within a summary block. (A summary block is considered to be an extension of a CYCLE.) The second CYCLE is ignored.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF420 EXCESS CHARACTERS OF FIELD LNG (d) IGNORED

Explanation: Excess characters have been found in the length specification for the current field. The excess characters are ignored.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF421 buffer BUFFER MAY NOT HAVE BEEN USED AS
0, OUTPUT SINCE PRIOR INPUT

Explanation: The specified buffer is used for input in this path, but it may not have been used for output since it was previously loaded. The buffer contents may be destroyed.

System Action: None. Assembly continues.

Programmer Response: Make sure that the buffer was used for output in a previous path or that the contents are of no value at this point in processing.

0,IDF423 COUNT PREDETERMINED, COUNT OPERAND IGNORED

Explanation: For sources of FID, RSN, CTR, or buffers with length specified, the count is set by the source type. The COUNT operand coded is ignored.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF424 SELFCHK IGNORED FOR EMITTED SOURCE

Explanation: The SELFCHK operand is ignored for emitted data (SOURCE = 'string').

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF425 SOURCE = RDR, SINK = PCH, AND 5496 COMMANDS
0, MAY BE INVALID ON A KATAKANA 3735

Explanation: The FDP has specified that Katakana code is supported and 5496 support is excluded.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IFD426 COMPARE IGNORED FOR SOURCE FID OR EMITTED

Explanation: Comparisons with SOURCE = FID or SOURCE = 'string' data are not permitted. The COMPARE operand is ignored for such data sources.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF427 KIND SET TO NUMERIC BY COMPARAND

Explanation: The comparand of a COMPARE operand is numeric and has forced the KIND option for the source to numeric. The KIND option that was coded is ignored.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF428 CTR IGNORED FOR EMITTED SOURCE

Explanation: Counter operations are not permitted with emitted source data (SOURCE = 'string'). The counter operation specified is ignored.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again. If operations involving constants and counters are desired, use the CTR operand of the FDCTRL macro.

0,IDF429 KIND SET TO NUMERIC BY COUNTER OPERAND

Explanation: A counter operation has caused the KIND option to be set to numeric. The KIND option that was coded is ignored.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF430 IND OPERAND IGNORED WITH SOURCE FID OR EMITTED

Explanation: Indicator operations are not permitted with SOURCE = FID or SOURCE = 'string' data. The indicator operations are ignored.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again. If operations involving explicit setting of indicators are desired, use the IND operand of the FDCTRL macro.

0,IDF431 COMMAND GROUP n, APPARENT RETROGRADE SKIPTO

Explanation: A SKIPTO command in the specified command group (n) has referred to the same 3286 line or a lower-numbered line. Backwards motion is not permitted. This may not be an error if only one of the SKIPTO commands is executed in a single execution of the FDP.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action described is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF432 BRANCH WITHIN SUMMARY BLOCK IGNORED

Explanation: A branch within a summary block is not permitted. The branch is ignored.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF433 BRANCH INTO CYCLE OR SUMMARY BLOCK IGNORED

Explanation: A branch into a CYCLE or summary block from outside the CYCLE or summary block is not permitted. A branch within a CYCLE or out of a CYCLE or summary block is permitted.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF434 MAX COUNT CONSIDERED EXACT FOR
0, NON-KEYBOARD SOURCES

Explanation: A maximum count specified for any SOURCE other than the Selectric ® keyboard is assumed to be the exact count. Maximum counts are applicable to SOURCE = KBD only.

System Action: None. Assembly continues.

Programmer Response: Make sure that the count described is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF435 MIN COUNT IGNORED FOR NON-KEYBOARD SOURCE

Explanation: A minimum count may be specified only for SOURCE = KBD. The minimum count is ignored for all other SOURCE types.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF436 KIND OPTION FORCED TO NUMERIC WITH BATCH OR SELFCHK

Explanation: The coding of a BATCH or SELFCHK operand implies a numeric SOURCE. KIND = N is assumed, and any coding of KIND is ignored.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF437 KIND OPTION FORCED TO NUMERIC WITH
0, UNCONDITIONALLY NUMERIC SOURCE

Explanation: A numeric SOURCE implies numeric KIND. KIND = N is assumed, and any coding of KIND is ignored.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF438 KIND OPERAND IGNORED WITH EMITTED SOURCE

Explanation: Emitted source data (SOURCE = 'string') cannot have any KIND attribute associated with it. The KIND operand is ignored.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF439 NUMERIC KIND OPTION FORCED BY CTR (d) SINK

Explanation: A specification of SINK = CTR (d) implies numeric KIND. KIND = N is assumed, and any coding of KIND is ignored.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF440 FILL OPERAND IGNORED WITH EMITTED SOURCE

Explanation: Filling of emitted source data (SOURCE = 'string') is not permitted. The FILL operand is ignored.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF441 FILL OPTION IGNORED FOR SINK d

Explanation: This type of the specified sink does not allow fill options. The FILL specification for this sink is ignored. The sinks that do not allow a FILL specification are SINK = NULL, SINK = CTR (d), and any sink that is described by a PICTURE operand.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF442 NUMERIC KIND OPTION FORCED BY ZERO FILL

Explanation: Zero FILL implies numeric KIND. KIND = N is assumed, and any coding of KIND is ignored.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF443 JUSTIFY OPTION IGNORED FOR SINK d

Explanation: The specified sink is one that cannot be justified. The JUSTIFY specification for this sink is ignored. The sinks that do not allow a JUSTIFY specification are SINK = NULL, SINK = CTR (d), and any buffered sink that is described by a PICTURE operand.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF444 UNDERLINE OPTION IGNORED FOR SINK n

Explanation: The specified sink is one that cannot be underlined. The UL specification for this sink is ignored. Only PRT sinks may be underlined.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF445 KIND SET TO NUMERIC BY IND OPERAND

Explanation: An indicator operation has forced the KIND specification for the current source to numeric; any encoding of KIND is ignored.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF446 PICTURE OPERAND IGNORED, SINK d NULL OR CTR

Explanation: NULL and CTR sinks may not be associated with a PICTURE. The PICTURE associated with the sink is ignored.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF447 POSSIBLE OVERLAY OF SINK d

Explanation: The current starting position for the specified sink is less than a previous ending position; data may be overlaid.

System Action: None. Assembly continues.

Programmer Response: This may not be an error if the sink is being processed non-sequentially. Make sure that the action taken is desirable and correct. If it is not, correct the error before assembling the program again.

0,IDF448 FIRST OPERATION ON buffer BUFFER MAY NOT HAVE
0, BEEN UNCONDITIONAL CLEAR OR INPUT

Explanation: The specified buffer is used in this path, but its contents may not be valid.

System Action: None. Assembly continues.

Programmer Response: Make sure that the buffer was cleared or loaded in a previous path, or load it in this path.

0,IDF449 buffer BUFFER MAY HAVE BEEN CLEARED
0, OR INPUT WITHOUT PRIOR OUTPUT

Explanation: The specified buffer is cleared or used for input in this path, but it may not have been used for output.

System Action: None. Assembly continues.

Programmer Response: Make sure that the buffer was used for output in a previous path, or that its contents are of no value at this point in processing.

0,IDF450 buffer BUFFER MAY HAVE BEEN OUTPUT
0, WITHOUT PRIOR INPUT

Explanation: The specified buffer is used for output in this path, but it may not have been properly loaded.

System Action: None. Assembly continues.

Programmer Response: Make sure that the buffer was loaded in some previous path, or load it in this path.

0,IDF451 FIRST OPERATION AFFECTING ind INDICATOR
0, WAS NOT UNCONDITIONAL CLEAR OR SEND

Explanation: The specified indicator is tested in this path, but it may not have been set previously.

System Action: None. Assembly continues.

Programmer Response: Make sure that a CLEAR or SEND command is issued before the test on this indicator.

0,IDF452 ind INDICATOR MAY HAVE BEEN
0, CLEARED WITHOUT PRIOR TEST

Explanation: The specified indicator (ind) is used in this path, but its prior status may not have been tested. Its status may have been destroyed.

System Action: None. Assembly continues.

Programmer Response: Make sure that the indicator was tested in a previous path or that its status is of no value at this point in processing.

0,IDF453 SAVELOC IN SUMMARY BLOCK IGNORED

Explanation: A SAVELOC operand may not be coded in a summary block. The SAVELOC is ignored.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF454 NAME OMITTED, SAVELOC IGNORED

Explanation: A SAVELOC operand was coded in the current macro, but the name entry (symbol) was not coded for the macro. The SAVELOC is ignored.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again (a name entry must be coded with SAVELOC).

0,IDF455 SAVELOC COUNT NOT BETWEEN d1 AND d2
0, ASSUME SAVELOC = YES

Explanation: The value coded for SAVELOC was not within the permitted range defined by d1 and d2. A value of SAVELOC = YES is assumed.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF456 command COMMAND MAY HAVE BEEN ISSUED
0, WITHOUT PRIOR TEST OF ind INDICATOR

Explanation: The expected logical sequence of commands and indicator tests may not have been followed possibly causing an execution time error.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action described is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF457 { EOF (RDR)/TIMEOUT } INDICATOR MAY HAVE BEEN TESTED
0, WITHOUT PRIOR { READ/SEND } COMMAND

Explanation: The action described may have occurred. The no-record-found indicator (NRF) should be preceded by a READ or WRITE command. The index indicator (NDX) should be preceded by an index counter operation in an FDCTRL CTR operand. The null indicator (NULL) should be preceded by a source operation.

System Action: None. Assembly continues.

Programmer Response: Make sure that the action described is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF458 KIND SET TO NUMERIC BY PICTURE OPERAND

Explanation: Coding a PICTURE operand implies numeric KIND. KIND = N is assumed, and any coding of KIND is ignored.

System Action: None, Assembly continues.

Programmer Response: Make sure that the action taken is desired and correct. If it is not, correct the error before assembling the program again.

0,IDF459 MACRO name CONTAINS n USED SAVELOC REFERENCES

Explanation: The named macro was coded with SAVELOC = d, but only n references were made to it. Such coding is inefficient and wastes macro resources.

System Action: None. Assembly continues.

Programmer Response: Correct the SAVELOC encoding, if desired. The correction is not mandatory unless message IDF737 is issued as well.

0,IDF460 OBJECT OPERAND INVALID, sys ASSUMED

Explanation: Either the OBJECT operand of an FDFORM macro is coded but is not DOS or OS, or in a multiple FDP assembly the OBJECT parameters specified for different FDPs are not the same.

System Action: The object deck is prepared for the specified system (OS or DOS). Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF461 CHAINING IN EFFECT, FDCTRL POSITION IGNORED

Explanation: The positioning parameter of an FDCTRL macro has been coded, but an operand is being chained. The position parameter is ignored.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF462 FDCTRL POSITION INVALID, DEFAULT TAKEN

Explanation: The position parameter of an FDCTRL macro is not between 1 and 130 or is not a decimal number. The default position is the column at which the preceding macro left the print element positioned.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF463 SELFCHK GENERATE AND CTR OPERATIONS
0, ARE PERFORMED BEFORE IND BRANCHING

Explanation: Either a SELFCHK generate or a CTR operation (CTR operand or SINK = CTR specification) has been coded in an IND operand that specifies a branch. These operations are independent of any branch to be taken and are performed before the analysis of the IND specification.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF464 macro IGNORED IN LOAD MODE

Explanation: The displayed macro has been coded in an FDP that has MODE = LOAD specified on the FDFORM macro. This FDP can contain only FDFORM, FDLOAD, and FDEND macros.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF465 INDEX COUNTER ctr MAY HAVE BEEN USED
0, WITHOUT PRIOR VALUE ASSIGNED

Explanation: This message reports that an index counter has been used as the starting position for a source or sink operation and may not have a known value.

System Action: None. Assembly continues.

Programmer Response: Analyze the logic of the FDP to verify that a value was assigned to the index counter before the counter was used.

0,IDF466 KEY AND/OR DELIMITER MAY NOT HAVE
0, BEEN PLACED INTO IOB BEFORE CURRENT OPERATION

Explanation: The current operation is a file command that may be using the KEY, KEYNOTE, or KEYLAST file qualifier without there being either a previous sink operation with DELIMIT or a GETKEY operation that placed the key and a delimiter into the I/O buffer.

System Action: None. Assembly continues.

Programmer Response: Verify that the key and delimiter are placed into the I/O buffer before such operations occur.

0,IDF467 FILE SAVE PTR MAY HAVE BEEN ALTERED
0, WITHOUT PRIOR USE

Explanation: The file save pointer to the next consecutive file record may be altered unintentionally because the preceding operation updated the pointer while the current operation sets it again.

System Action: None. Assembly continues.

Programmer Response: Verify that the file save pointer is not changed due to non-sequential macro processing or is of no value.

0,IDF468 FILE SAVE PTR MAY HAVE BEEN USED
0, WITHOUT PRIOR SET

Explanation: The value of the file save pointer may be erroneous because the current operation begins at the record indicated by the pointer even through the preceding operation was not a set or update to the pointer.

System Action: None. Assembly continues.

Programmer Response: Verify that the file save pointer was actually set or correct the error and reassemble the program.

0,IDF469 EXCESS SUBOPERANDS OF KEYLEN IGNORED

Explanation: The key length parameter of an FDLOAD macro has been coded with superfluous suboperands that are ignored.

System Action: None. Assembly continues.

Programmer Response: None.

0,IDF470 EXCESS SUBOPERANDS OF DATALEN IGNORED

Explanation: The data length parameter of an FDLOAD macro has been coded with superfluous supoperands that are ignored.

System Action: None. Assembly continues.

Programmer Response: None.

0,IDF471 EXCESS SUBOPERANDS OF ENDCHAR IGNORED

Explanation: The end character parameter of an FDLOAD macro has been coded with superfluous suboperands that are ignored.

System Action: None. Assembly continues.

Programmer Response: None.

0,IDF472 ENDCHAR TOO LONG, TRUNCATED TO nn CHARACTERS

Explanation: The characters specified in the ENDCHAR operand of the FDLOAD macro are longer than the KEYLEN operand specified. The end character parameter is truncated to the length specified in the key length parameter.

System Action: None. Assembly continues.

Programmer Response: None.

0,IDF473 ENDCHAR TOO SHORT, PADDED TO nn BY mm BLANKS

Explanation: The characters specified in the ENDCHAR operand of the FDLOAD macro do not meet the length requirement specified in the KEYLEN operand. The end characters are padded with the number of blanks needed to equal the key length specified.

System Action: None. Assembly continues.

Programmer Response: None.

0,IDF474 DEFAULT ENDCHAR IS nn ASTERISKS

Explanation: The ENDCHAR operand on the FDLOAD macro has been omitted. When the ENDCHAR operand is omitted, the end characters are asterisks, the same number as specified in the KEYLEN operand.

System Action: None. Assembly continues.

Programmer Response: None.

0,IDF475 FDLOAD IGNORED FOR NON LOAD FDP

Explanation: This message indicates that an FDLOAD macro has been specified in an FDP without a companion FDFORM macro specifying MODE = LOAD.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF476 SOURCE X1/X2 FORCES SINK OF PRT

Explanation: A source of either of the index counters, X1 or X2, has been coded without the mandatory SINK = PRT specification in the FDFIELD macro.

System Action: None. Assembly continues.

Programmer Response: Correct the error before assembling the program again.

0,IDF477 INDEX COUNTER Xn MAY NOT HAVE BEEN LOADED OR CLEARED
0, PRIOR TO CURRENT OPERATION

Explanation: The specified index counter has been used as a generator in FDCTRL CTR operations and may not have been cleared or loaded before this use.

System Action: None. Assembly continues.

Programmer Response: Verify that the index counter was properly loaded or cleared; or correct the error and reassemble the program.

8,IDF700 MANDATORY FID OPERAND OMITTED

Explanation: The FID operand must be coded, but was not found.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Code the FID operand in the FDFORM macro before assembling the program again.

8,IDF701 FORM NAME INVALID OR OMITTED; subname USED

Explanation: The form name is not a valid symbol or has been omitted. A generated name is substituted to permit syntax checking for the rest of the FDP. (The name field of the FDFORM macro must be coded and must comply with Assembler naming conventions.)

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF702 INVALID BRANCH

Explanation: The branch generated as the result of a GOTO or CYCLE target is to a point too far from the branch point to resolve. A branch may not be made to a target more than 15K unpacked bytes from the branch point.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Restructure the FDP to reduce the number of unpacked bytes between the branch and its target.

8,IDF703 PREVIOUS FORM NOT PROPERLY TERMINATED

Explanation: The previous FDP did not end with an FDEND macro. The current FDFORM macro causes the FDEND functions to be performed for the previous FDP.

System Action: Processing of the current form proceeds normally, but the previous form is flagged as invalid.

Programmer Response: Correct the error before assembling the program again.

8,IDF704 PAGE WITHIN CYCLE IGNORED

Explanation: Cyclic processing may include FDLINE, FDFIELD, and FDCTRL macros only. The FDPAGE macro is ignored.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF705 FORM ENDED BEFORE CYCLE LIMIT ENCOUNTERED

Explanation: An FDEND macro was found before the limit of the current CYCLE was found.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Make sure that the CYCLE limit appears before or at the FDEND macro. After the error is corrected, assemble the program again.

8,IDF706 EXPECTED CHAINING OF PRECEDING MACRO
0, NOT FOUND, CHAINING TERMINATED

Explanation: The preceding macro was chained, but the current macro is not the required continuation. The current macro causes termination of the chaining function.

System Action: Further code generation is suppressed; the chained macro is simulated (without operands) and the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF707 COMMAND GROUP n, ILLEGAL USE OF CLEAR

Explanation: The buffer to be cleared has not been specified.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF708 SKIPTO COMMAND ILLEGAL IN CYCLE OR SUMMARY

Explanation: The SKIPTO command is not permitted in a CYCLE or a summary block. The SKIP command should be used instead.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF709 COMMAND GROUP n, SKIP OR SKIPTO NONDECIMAL

Explanation: A SKIP command or a SKIPTO command was coded with a non-decimal number.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF711 COMMAND GROUP n, PRINT ILLEGAL AFTER CLEAR

Explanation: The PRINT command is not permitted following the CLEAR(LPB) command. CLEAR followed by SKIP may be used to obtain blank lines.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF712 COMMAND GROUP n, ILLEGAL CLEAR OR READ

Explanation: A CLEAR or READ command in the specified command group (n) specifies an invalid buffer or device.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF713 COMMAND GROUP n, ILLEGAL DUE TO
0, SPECIFICATION OF m DEVICE TYPES

Explanation: The parameters of the specified command group (n) should all refer to the same device. Either m different devices were referred to or no device was identifiable (m=0).

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF714 EXPECTED CHAINING OF op OPERAND
0, NOT FOUND, CHAINING TERMINATED

Explanation: A preceding operand was being chained, but no continuation was found. The chaining function is terminated.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF715 CHARACTER NEAR POSITION p OF op
0, [SUB] OPERAND [n] IS ILLEGAL

Explanation: An illegal character has been found in the named operand (op) or sub-operand near the specified position (p). The character is not a valid 3735 character.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the character in error or substitute a valid 3735 character before assembling the program again.

8,IDF716 macro1 MUST FOLLOW macro2

Explanation: This message reports that the required sequence of macros has not been found. macro1 is the expected macro and macro 2 is the previous macro.

System Action: Further code generation is suppressed; the missing macro is simulated (without operands) and the rest of the source program is checked for syntax errors.

Programmer Response: Correct the macro sequence before assembling the program again.

8,IDF717 op OPERAND INVALID

Explanation: The specified operand (op) has been coded in an invalid manner. This may be caused by coding non-decimal characters in a decimal parameter, or by not coding one of the required parameters of an operand. For the SOURCE operand, this message may indicate that SOURCE = 'string' was coded in some macro other than FDFIELD. For the PAGE NUMBER operand, this message is issued if the product of the page number and the page height exceeds 16,383 (indicating that there are too many lines in the form).

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF718 op OPERAND OMITTED

Explanation: The specified operand (op) is required, but has been omitted.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Supply the missing operand before assembling the program again.

8,IDF719 op [SUB] OPERAND [n]
0, NOT BETWEEN a AND b

Explanation: The specified operand parameter (op) is not within the permitted range defined by a and b.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF720 BATCH FOR op SUBOPERAND n INVALID

Explanation: The BATCH number for the specified operand (op) is invalid.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF721 FID FOR op SUBOPERAND n INVALID

Explanation: The FID parameter for the specified operand (op) is invalid.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF722 CTR FOR op SUBOPERAND n INVALID

Explanation: The counter parameter for the specified operand (op) is invalid.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF723 IND FOR op SUBOPERAND n INVALID

Explanation: The indicator parameter for the specified operand (op) is invalid.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF724 EOF FOR op SUBOPERAND n INVALID

Explanation: The EOF parameter for the specified operand (op) is invalid.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF725 OPERATOR FOR op SUBOPERAND n INVALID

Explanation: The arithmetic operator for the specified operand (op) is invalid.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF726 EMIT FOR op SUBOPERAND n INVALID

Explanation: The emitted ('string') data for the specified operand (op) is invalid.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF727 BATCH FOR op SUBOPERAND n NOT
0, BETWEEN a AND b

Explanation: The BATCH parameter for the specified operand (op) is not within the expected limits defined by a and b.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF728 FID FOR op SUBOPERAND n NOT
0, BETWEEN a AND b

Explanation: The FID parameter for the specified operand (op) is not within the expected limits defined by a and b.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF729 CTR FOR op SUBOPERAND n NOT
0, BETWEEN a AND b

Explanation: The CTR parameter for the specified operand (op) is not within the expected limits defined by a and b.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF730 IND FOR op SUBOPERAND n NOT
0, BETWEEN a AND b

Explanation: The IND parameter for the specified operand (op) is not within the expected limits defined by a and b.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF731 EOF FOR op SUBOPERAND n NOT
0, BETWEEN a AND b

Explanation: The EOF parameter for the specified operand (op) is not within the expected limits defined by a and b.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF732 SKIP FOR op COMMAND GROUP n NOT
0, BETWEEN a AND b

Explanation: The SKIP parameter for the specified operand (op) in the command group is not within the expected limits defined by a and b.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF733 INVALID CHARACTER IN MESSAGE SUBOPERAND n

Explanation: A character that is invalid on the 3735 has been found in the specified MESSAGE suboperand (n).

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Remove the character in error or substitute a valid character before assembling the program again.

8,IDF734 ATTEMPTED MOVEMENT TO A PREVIOUSLY
0, DEFINED LINE INVALID

Explanation: A reference that implies backward motion to a previously-defined line has been made. Backward motion is not permitted. This situation may arise when a GOTO target is found in a lower-numbered line than the one in which the GOTO is issued.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF735 CYCLE COUNT INVALID, COUNT OF 1 ASSUMED

Explanation: The count parameter of the CYCLE operand is invalid. A count of 1 is assumed so that operand checking may continue.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF736 CYCLE COUNT NOT BETWEEN a AND b,
0, COUNT OF 1 ASSUMED

Explanation: The cycle count is not within the limits defined by a and b. If coded in FDCTRL, the limits are 1 and 16383. If coded in FDLINE or FDFIELD, the count specified will cause the cycle to exceed the length of the page. A count of 1 is assumed so that the operand checking may continue.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF737 TOO MANY UNRESOLVED BRANCHES

Explanation: There are too many GOTOs and CYCLEs with targets that have not been resolved by finding the target names.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Restructure the FDP so that fewer GOTO and CYCLE target references are outstanding at any one time, then assemble the program again.

8,IDF738 INVALID FORM DESCRIPTION PROGRAM

Explanation: One or more severe errors (MNOTE level 8) have been detected in this FDP, and the FDP has been flagged as invalid. If this FDP is used as input to the Form Description utility, it will be rejected.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct all noted errors before assembling the program again.

8,IDF739 DOCUMENT FIELD LNG(d) IS NONDECIMAL

Explanation: A non-decimal character has been found in the length specification for the current field. Only decimal digits are permitted in a LNG (d) specification.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF740 DEAD CODE, MACRO IGNORED

Explanation: The current FDFIELD macro is located in such a position that it can never be executed. This may happen when the macro follows an unconditional GOTO, has no name, and has no named macro preceding it that can be used as an entry point.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: If the macro is not required, remove it. If the macro is required, establish some path by which the macro can be reached. When corrections are complete, assemble the program again.

8,IDF741 SOURCE KEYBOARD OPTIONS INVALID

Explanation: The specified source keyboard option is not one of the permitted options, or was coded more than once.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF742 SOURCE START OR END POSITION INVALID

Explanation: The specified source start or end position is non-decimal. The source start or end position cannot be properly determined.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF743 SOURCE START OR END POSITION NOT
0, BETWEEN a AND b

Explanation: The specified source start or end position is not within the permitted range defined by a and b.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF744 SOURCE LENGTH SPECIFICATION INVALID/INADEQUATE

Explanation: The specified source length is non-decimal. The source length cannot be properly determined.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF745 SOURCE LENGTH NOT BETWEEN a AND b

Explanation: The specified source length is not within the permitted range defined by a and b. The source length cannot be properly determined.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF746 MAX/EXACT COUNT NOT BETWEEN a AND b

Explanation: The maximum (or exact) count is not within the permitted range defined by a and b. The maximum/exact count cannot be properly determined.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF747 MINIMUM COUNT NOT BETWEEN a AND b

Explanation: The minimum count is not within the permitted range defined by a and b. The minimum count cannot be properly determined.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF748 START POSITION FOR SINK d INVALID

Explanation: The starting position for the specified sink (d) is non-decimal or is not a valid index counter. The starting position cannot be properly determined.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF749 START POSITION FOR SINK d
0, NOT BETWEEN a AND b

Explanation: The starting position for the specified sink (d) is not within the permitted range defined by a and b.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF750 LNG (d) FOR SINK d INVALID

Explanation: The length for the specified sink (d) is non-decimal. The length cannot be properly determined.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF751 LNG (d) FOR SINK d NOT BETWEEN a AND b

Explanation: The length for the specified sink (d) is not within the permitted range defined by a and b.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF752 END POSITION FOR SINK d INVALID

Explanation: The end position for the specified sink (d) is non-decimal or has been specified with a starting position determined by an index counter that requires a LNG (d) specification instead of an end position. The end position cannot be properly determined.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF753 END POSITION FOR SINK d
0, NOT BETWEEN a AND b

Explanation: The end position for the specified sink (d) is not within the permitted range defined by a and b.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF754 NUMBER OF EMITTED 'STRING' CHARACTERS
0, NOT BETWEEN a AND b

Explanation: The number of emitted 'string' characters is not within the permitted range defined by a and b.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF755 NUMBER OF CHARACTERS IN COMPARAND OF COMPARE
0, SUBOPERAND n NOT BETWEEN a AND b

Explanation: The number of characters in the specified comparand is not within the permitted range defined by a and b. The COMPARE suboperand cannot be properly evaluated.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the comparand in error before assembling the program again.

8,IDF756 INVALID ARITHMETIC OPERATION IN CTR
0, SUBOPERAND n

Explanation: An arithmetic operator in the specified CTR suboperand (n) is invalid. (The permitted arithmetic operators are described in the discussion of the CTR operand.) The arithmetic expression cannot be properly evaluated.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Locate and correct the operator in error before assembling the program again.

8,IDF757 INVALID COMPARAND LENGTH IN IND OPERAND

Explanation: The comparand length in the IND operand is not valid. The IND operand cannot be properly evaluated.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Locate and correct the comparand length in error before assembling the program again.

8,IDF758 UNRESOLVED BRANCH TO MACRO 'name'
0, FROM PATH p SEGMENT s

Explanation: The target specified in a GOTO or a CYCLE could not be found. Correct branching code cannot be generated.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Make sure that the target name specified in the GOTO or CYCLE appears on or before the FDEND macro. Correct the error before assembling the program again.

8,IDF759 LOGICAL OPERATOR NEAR POSITION p OF
0, IND SUBOPERAND n INVALID

Explanation: The logical operator specified is not AND or OR. The specified IND suboperand (n) cannot be properly evaluated.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Locate and correct the operator in error before assembling the program again.

8,IDF760 COMPARISON OPERATOR NEAR POSITION p OF
0, IND SUBOPERAND n INVALID

Explanation: The comparison operator specified is not valid. (The permitted comparison operators are described in the discussion of the IND operand.) The specified IND suboperand (n) cannot be properly evaluated.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Locate and correct the operator in error before assembling the program again.

8,IDF761 COMPARAND CHARACTER NEAR POSITION p OF
0, IND SUBOPERAND n INVALID

Explanation: Either the character must be decimal and is not, or the character is not a valid 3735 character. The specified IND suboperand (n) cannot be properly evaluated.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Locate and correct the character in error before assembling the program again.

8,IDF762 IND COMPARAND LENGTH NOT BETWEEN 1 AND 127

Explanation: The length of an indicator comparand was found to be outside the permitted range (from 1 to 127 characters).

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the comparand in error before assembling the program again.

8,IDF763 PICTURE ILLEGAL WITH EMITTED SOURCE

Explanation: Editing with a PICTURE operand is not permitted with SOURCE = 'string' (emitted string).

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF764 PICTURE ILLEGAL WITH NON-NUMERIC COMPARISONS

Explanation: A PICTURE has been coded for a field that is used in a non-numeric comparision. Non-numeric fields may not be associated with a picture operand.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Determine what the comparison in question should be, then correct the error before assembling the program again.

8,IDF765 LENGTH SPECIFICATION FOR SINK d IS INADEQUATE

Explanation: The length of the specified sink (d) was not explicitly coded, and the information provided by other parameters is not sufficient to determine the sink length.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Provide some explicit encoding that specifies the length of the indicated sink before assembling the program again.

8,IDF766 PICTURE SUBOPERAND n IMPROPERLY FRAMED

Explanation: Each PICTURE suboperand must be framed in apostrophes, for example: '$**9.99'.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the framing error before assembling the program again.

8,IDF767 CHARACTER c OF PICTURE SUBOPERAND n IS
0, INVALID, MUST BE ONE OF THE FOLLOWING
0, characters

Explanation: The character at position c (not counting the starting apostrophe) of the specified PICTURE suboperand (n) is either an invalid PICTURE character or a valid character that has been used improperly. The characters printed in the third line of the message are the only characters which are permitted in the specified position. The possible characters are:

9YZ*$+-SVCDB,/.

If any of these characters is not in the character set of the printer that printed the FDP listing, the appropriate substitution is used. For example, the FDP listing may have @ printed in places where a $ should appear. In this case, any appearance of an @ should be interpreted as the $ sign.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Identify the error and correct it before assembling the program again.

8,IDF768 PICTURE SUBOPERAND n NOT PROPERLY TERMINATED

Explanation: Either a terminating 'CR' or 'DB' is in error or excess characters are present in the PICTURE specification.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the PICTURE specification in question before assembling the program again.

8,IDF769 SINK COUNT NOT BETWEEN a AND b

Explanation: The sink count is not within the range specified by a and b.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF770 PRINT ELEMENT POSITION CANNOT BE DETERMINED
0, DUE TO INADEQUATE SOURCE COUNT SPECIFICATION

Explanation: The information provided is not sufficient to determine proper print element positioning.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Code a correct value of the COUNT operand in the FDFIELD macro that describes the field in question before assembling the program again.

8,IDF771 PRINTING SINK EXCEEDS FIELD MARGINS

Explanation: The current sink length exceeds the allowable field margins.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Check the sink and field in question and correct the error before assembling the program again.

8,IDF772 op OPERAND, SUBOPERAND n, FORMAT INVALID

Explanation: The indicated suboperand (n) is specified in an invalid manner.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF773 op OPERAND, SUBOPERAND n, NOT AN ALLOWED EXACT VALUE
0, OR NOT BETWEEN a AND b

Explanation: The specified suboperand (n) does not contain a proper value.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF774 COMMAND GROUP n, INVALID FORMAT OR COMMAND

Explanation: The specified command group (n) is coded with an invalid format or command.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF775 COMMAND GROUP n, SKIP OR SKIPTO VALUE
0, NOT BETWEEN a AND b

Explanation: The value of a SKIP or SKIPTO command in the specified command group (n) does not fall within the permitted range defined by a and b.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF777 MODE OPERAND INVALID

Explanation: This message indicates that the MODE operand on an FDFORM macro has been specified incorrectly with neither LOAD nor NOLOAD code.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF778 MANDATORY KEYLEN OPERAND OMITTED

Explanation: The mandatory key length specification for the records to be loaded by the FDLOAD macro has been omitted.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF779 KEYLEN OPERAND IS CODED NON-NUMERIC

Explanation: The KEYLEN operand on the FDLOAD macro has not been specified as a decimal number.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF780 KEY PARAMETER OF GETKEY COMMAND NOT
0, BETWEEN 1 AND 15 CHARACTERS LONG

Explanation: The key specified in a GETKEY command in an FDCTRL macro is not within the correct range. All keys for the File Storage capability must be from 1 to 15 characters long.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF782 KEYLEN NOT BETWEEN 1 AND 15

Explanation: The key length for records to be loaded into the file by the FDLOAD macro is not between the valid length specifications of 1 and 15, inclusive.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF783 MANDATORY DATALEN OPERAND OMITTED

Explanation: The mandatory DATALEN operand on the FDLOAD macro for the records to be loaded in the file has been omitted.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF784 DATALEN OPERAND IS CODED NON-NUMERIC

Explanation: The DATALEN operand on the FDLOAD macro has not been specified as a decimal number.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF785 DATALEN PLUS KEYLEN NOT BETWEEN n AND m

Explanation: The sum of the key and data lengths specified on the FDLOAD macro is not within the range determined by a minimum of KEYLEN+1 and a maximum of 234 characters.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF786 NUMBER IN CTR SUBOPERAND d NOT BETWEEN 1 AND 255

Explanation: A signed decimal number is being added to or subtracted from an index counter in FDCTRL CTR operations and the number is not within the valid range of 1 to 255.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF787 MORE THAN ONE SINK DELIMIT INVALID

Explanation: The FD macros have detected multiple SINK operands specifying DELIMIT when only one SINK operand in each FDFIELD macro may specify DELIMIT.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF788 DELIMIT/QUALIFIER FOR SINK n INVALID

Explanation: The fourth parameter of a SINK = IOB operand has been coded, but is not DELIMIT.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF789 IND OPERATOR OR INDEX COUNTER FOR IND SUBOPERAND d INVALID

Explanation: The second parameter of an IND suboperand is not ON, OFF, INV, X1, or X2.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF790 INDEX COUNTER COMPARISON OPERATOR INVALID

Explanation: The comparison operator for an index counter compare operation is not GT, LT, GE, LE, EQ, NE, NL, NG, or one of their symbolic equivalents.

System Action: Further code generation is suppressed; the rest of the source program is checked for syntax errors.

Programmer Response: Correct the error before assembling the program again.

8,IDF999 FDM SYSTEM ERROR

Explanation: A severe FD macro internal error has occurred, such as the exhaustion of the internal work space.

System Action: The assembly is terminated.

Programmer Response: If the error persists after all other noted errors have been eliminated, do the following before calling IBM for programming support:

- Have the job stream and program listing available.

Any data following the message line is printed to help the IBM program systems representative locate the error. Some of these additional data lines are:

DESTINATION QUEUE IS EMPTY – The first element on the destination queue is placed there during initialization and should never be removed.

UNRESOLVED IMPLICIT ELEMENT ON ORIGIN QUEUE – All implicit elements are resolved by end control, so none should remain at FDEND.

INVALID CALL OF INNER MACRO name [MESSAGE CALL PARAMETER = m] – The parameters passed to the specified macro are invalid. For calls to message macros, the call parameter is the message call number.

Appendix E. OS Form Description Utility Diagnostic Messages

This appendix describes the messages that may be issued to the programmer during OS FD utility processing. These messages provide diagnostic information of interest to the programmer adding members to (or creating) a library of FDPs. The listing provided consists of a header line, several blank lines, and one or more detail lines for each of the three utility steps (control, link-edit, and storage).

If the SYSPRINT data set is not opened, or if an error condition arises in writing to the SYSPRINT data set, one of these messages is directed to the system console:

IDF019I SYSPRINT CANNOT BE OPENED

Return Code: 4.

System Action: Processing is terminated.

Routing Code/Descriptor Code: 11 (Programmer Information)/7 (Application Program/Processor).

Programmer Response: Provide the missing CTRL.SYSPRINT or STG.SYSPRINT DD statement and execute the job again.

IDF020I I/O ERROR SYSPRINT

Return Code: 4.

System Action: The message supplied by the SYNADAF routine is printed at the console on the following line and processing is terminated.

Routing Code/Descriptor Code: 4, 10, 11 (Direct Access Poll, System/Error Maintenance, Programmer Information)/4 (System Status).

Programmer Response: Correct the error condition and execute the job again. The appropriate system utility (IEHLIST or IEBPTPCH) may be needed to determine the status of the library if the error occurred in the STG step.

If the SYSPRINT data set is open and no error is encountered in writing to it, all messages are directed to it. The messages that may be issued during each of the three utility steps are described in the following sections.

OS Control Step Messages

IDF001I SYSIN CANNOT BE OPENED

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: The SYSIN data set cannot be opened because the DD statement is missing (and the Operating System did not generate one) or specified incorrectly. Provide a correct DD statement and execute the job again.

IDF002I SYSUT1 CANNOT BE OPENED

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: The SYSUT1 data set cannot be opened because the DD statement is missing or specified incorrectly. Provide a correct DD statement and execute the job again.

IDF004I ESD CARD TYPE CODE ERROR LAST DECK ID xxxx NO yyyy

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: xxxx is the identification (columns 73-76), and yyyy is the sequence number (columns 77-80), of the last valid ESD card image in the object module. The control routine has detected an error in the ESD type code field. A type code of '00' (indicating SD) should be present, but is not. Check the field in question and correct the error before executing the job again.

IDF005I ESD CARD ADDRESS FIELD ERROR LAST DECK ID xxxx NO yyyy

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: xxxx is the identification (columns 73-76) and yyyy is the sequence number (columns 77-80), of the last valid ESD card image in the object module. The control routine has detected an error in the ESD address field. Check the field in question and correct the error before executing the job again.

IDF006I ESD CARD LENGTH FIELD ERROR LAST DECK ID xxxx NO yyyy

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: xxxx is the identification (columns 73-76) and yyyy is the sequence number (columns 77-80) of the last valid ESD card image in the object module. The control routine has detected an error in the ESD length field. Check the field in question and correct the error before executing the job again.

IDF007I ESD CARD NAME FIELD ERROR LAST DECK IS xxxx NO yyyy

Return Code: 4.

System Action: Processing is terminated

Explanation/Programmer Response: xxxx is the identification (columns 73-76) and yyyy is the sequence number (columns 77-80), of the last valid ESD card image in the object module. The control routine has detected an error in the ESD name field. Check the field in question and correct the error before executing the job again.

IDF008I CARD OUT OF SEQUENCE LAST DECK ID xxxx NO yyyy

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: xxxx is the identification (columns 73-76) and yyyy is the sequence number (columns 77-80), of the last valid card image in the object module. The control routine has detected a card image out of sequence. Check the card image in question and correct the error before executing the job again.

IDF009I DECK ID FIELD ERROR LAST DECK ID xxxx NO yyyy

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: xxxx is the identification (columns 73-76) and yyyy is the sequence number (columns 77-80), of the last valid card image

in the object module. The control routine has detected a card image with an invalid object module (deck) identification. Check the card image in question and correct the error before executing the job again.

IDF010I I/O ERROR ON SYSIN

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: An uncorrectable I/O error occurred while the SYSIN data set was being read. If the error persists, do the following before calling IBM for programming support:

- Have the job stream and program listing available.
- Have the master console sheet available.

IDF011I I/O ERROR ON SYSUT1

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: An uncorrectable I/O error occurred while the SYSUT1 data set was being read. If the error persists, do the following before calling IBM ofr programming support:

- Have the job stream and program listing available.
- Have the master console sheet available.

IDF012I SUCCESSFUL COMPLETION LAST DECK ID xxxx NO yyyy

Return Code: 4.

System Action: None.

Explanation/Programmer Response: xxxx is the identification (columns 73-76), and yyyy is the sequence number (columns 77-80), of the last valid card image in the object module. The control routine has finished processing. The output from this step has been passed to the link-edit step.

IDF013I OBJECT DECK IS INCOMPLETE

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: An incomplete object deck has been detected. A check was made at EOF (SYSIN) to see if an END card was read, and no END card was found. Check the object deck in question and correct the error before executing the job again.

OS Link-Edit Step Messages

IEW0000 (control statement)

Severity Code: 0.

System Action: Not applicable.

Explanation/Programmer Response: The control statement is printed as a result of the LIST option. No programmer response is required.

IEW0254 ERROR - TABLE OVERFLOW - TOO MANY EXTERNAL SYMBOLS IN ESD

Severity Code: 4.

System Action: The load module is marked "not executable."

Explanation/Programmer Response: The capacity of the Linkage Editor has been exceeded. Link-edit the object modules in a larger main storage environment or prepare fewer FDPs for linkage editing.

IEW0284 ERROR - DDNAME PRINTED CANNOT BE OPENED

Severity Code: 4.

System Action: The load module is marked "not executable." The data data definition name in the name field of the DD statement for the data set is printed after the message code.

Explanation/Programmer Response: The specified data set cannot be opened because the DD statement defining the data set is missing or specified incorrectly. Provide a DD statement that correctly defines the data set and execute the job again.

IEW0294 ERROR - DDNAME PRINTED HAD SYNCHRONOUS ERROR

Severity Code: 4.

System Action: Processing is terminated. The data definition name in the name field of the DD statement for the data set is printed after the message code. If an input/output error occurred, the information provided by the SYNADAF macro instruction is printed after the message code in the following format: SYNAD EXIT, jobname, stepname, unit address, device type, ddname operation attempted, error description, block count or BBCCHHR, and access method.

Explanation/Programmer Response: Either (1) an uncorrectable physical input/output error occurred, (2) an object module is missing an END card as the last card, or (3) if the data definition name printed is for a DD statement that defines a blocked input data set of fixed format, an input record larger than the specified block size or logical record length was found. For any fixed format data set, specify the correct block size. If the proper block size was specified, have the computing system checked.

IEW0324 ERROR - MAXIMUM NUMBER OF SEGMENTS EXCEEDED

Severity Code: 4.

System Action: The load module is marked "not executable."

Explanation/Programmer Response: The capacity of the Linkage Editor has been exceeded. Link-edit the object modules in a larger main storage environment or prepare fewer FDPs for linkage editing.

OS Storage Step Message

Errors detected in the storage step (except for I/O errors) require that the FDP in error be reassembled.

IDF021I SYSLIB CANNOT BE OPENED

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: The SYSLIB data set cannot be opened because the DD statement is missing or specified incorrectly. Provide a correct DD statement and execute the job again.

IDF022I ERROR IN PARM FIELD

Return Code: 0.

System Action: Processing continues.

Explanation/Programmer Response: The user has provided an incorrect parameter to the storage routine. The storage routine will process as if no parameter had been specified. Correct the parameter and execute the job again.

IDF023I I/O ERROR ON SYSLIB

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: An uncorrectable I/O error occurred on the SYSLIB data set. If the error persists, do the following before calling IBM for programming support:

- Have the job stream and program listing available.
- Have the master console sheet available.

IDF024I XXXXXXXX HAS AN INVALID NAME

Return Code: 4.

System Action: Processing continues. The form is not added to the user's data set.

Explanation/Programmer Response: The storage routine has detected an error in the name field of a sector. The form name is XXXXXXXX. Correct the problem before executing the job again.

IDF025I XXXXXXXX IS AN INCOMPLETE FORM

Return Code: 4.

System Action: Processing continues.

Explanation/Programmer Response: The storage routine has detected a flag in a sector indicating that the form is incomplete. The form name is XXXXXXXX. This form is not added to the SYSLIB data set. Correct the problem before executing the job again.

IDF026I XXXXXXXX IS AN INVALID FORM

Return Code: 4.

System Action: Processing continues.

Explanation/Programmer Response: The storage routine has detected a flag in a sector indicating that the form is invalid. The form name is XXXXXXXX. This form is not added to the SYSLIB data set. Correct the problem before executing the job again.

IDF027I NO SPACE LEFT IN DIRECTORY

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: The library defined by SYSLIB has no more directory space available. Restructure the data set to provide more directory space before executing the job again.

IDF028I I/O ERROR READING SYSLIB DIRECTORY

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: A return code indicating a permanent I/O error was returned from a STOW macro that attempted to stow a member on the SYSLIB data set. If the error persists, do the following before calling IBM for programming support:

- Have the job stream and program listing available.
- Have the master console sheet available.

IDF029I XXXXXXXX HAS BEEN ADDED TO SYSLIB DATA SET

Return Code: None.

System Action: Processing continues.

Explanation/Programmer Response: The form named by XXXXXXXX has been successfully stored on the SYSLIB data set. No programmer action is required.

IDF030I NO TEMPORARY NAMES AVAILABLE

Return Code: None.

System Action: Processing continues.

Explanation/Programmer Response: All temporary names in the series IDFTEMP0-IDFTEMP9 have been used. Rename or delete the temporary forms, or rename the form before executing the job again.

IDF031I XXXXXXXX STOWED AS TEMPORARY IDFTEMPn

Return Code: None.

System Action: Processing continues.

Explanation/Programmer Response: The form named by XXXXXXXX is a duplicate of an existing name in the SYSLIB data set. No REPLACE operation was specified in the EXEC statement of the STG job step, so the form was stored with the temporary name IDFTEMPn. (Values of "n" may range from 0 to 9.)

IDF032I XXXXXXXX HAS AN INVALID COUNT

Return Code: 4.

System Action: Processing is terminated.

Explanation/Programmer Response: The storage routine detected an error in the count field of a sector. Correct the error before executing the job again.

IDF033I END OF STORAGE PROCESSING

Return Code: 0.

System Action: Control is returned to the Operating System.

Explanation/Programmer Response: The storage routine has completed normal processing. Check all messages issued during this processing step to make sure that the FD programs were stored properly. If necessary, make corrections and execute the job again.

Appendix F. DOS Form Description Utility Diagnostic Messages

This appendix describes the messages that may be directed to the programmer during DOS FD utility execution. These messages provide diagnostic information of interest to the programmer making additions to an indexed sequential data set of FDPs. The listing provided consists of a header line, several blank lines and one or more detail lines for each of the three utility steps (control, link-edit, and storage).

DOS Control Step Messages

4F01I OBJECT DECK INCOMPLETE

System Action: Processing is terminated.

Explanation/Programmer Response: An incomplete object deck has been detected. A check was made at EOF (SYSIPT) to see if an END card was read, and no END card was found. Check the object deck in question and correct the error before executing the job again.

4F03I ESD CARD OUT OF SEQUENCE

System Action: Processing is terminated.

Explanation/Programmer Response: The FD utility control step has detected a sequence error in an object module. Correct the input sequence and execute the job again.

4F04I ESD CARD TYPE CODE ERROR

System Action: Processing is terminated.

Explanation/Programmer Response: The control step has detected an error in the ESD type code field. A type code of '00' (indicating SD) should be present, but is not. Check the field in question and correct the error before executing the job again.

4F05I ESD CARD ADDRESS FIELD ERROR

System Action: Processing is terminated.

Explanation/Programmer Response: The control step has detected an error in the ESD address field. Check the field in question and correct the error before executing the job again.

4F06I ESD CARD LENGTH FIELD ERROR

System Action: Processing is terminated.

Explanation/Programmer Response: The control step has detected an error in the ESD length field. Check the field in question and correct the error before executing the job again.

4F07I ESD CARD NAME FIELD ERROR

System Action: Processing is terminated.

Explanation/Programmer Response: The control step has detected an error in the ESD name field. Check the field in question and correct the error before executing the job again.

4F08I CARD OUT OF SEQUENCE

System Action: Processing is teminated.

Explanation/Programmer Response: The control step has detected a card image out of sequence. Check the card image in question and correct the error before executing the job again.

4F09I DECK ID FIELD ERROR

System Action: Processing is teminated.

Explanation/Programmer Response: The control step has detected a card image with an invalid object module (deck) identification. Check the card image in question and correct the error before executing the job again.

4F10I I/O ERROR ON SYSIPT

System Action: Processing is terminated.

Explanation/Programmer Response: An uncorrectable I/O error occurred while the SYSIPT data set was being read. If the error persists, do the following before calling IBM for assistance:

- Have the job stream and program listing available.
- Have the console sheet available.

4F11I I/O ERROR ON SYSPCH

System Action: Processing is terminated.

Explanation/Programmer Response: An uncorrectable I/O error occurred on SYSPCH while the control step was attempting to punch a deck. If the error persists, do the following before calling IBM for assistance:

- Have the job stream and program listing available.
- Have the console sheet available.

4F12I SUCCESSFUL COMPLETION. LAST DECK ID XXXX NO. YYYY

System Action: None.

Explanation/Programmer Response: XXXX is the deck identification (columns 73-76), and YYYY is the sequence number (columns 77-80), of the last valid card image in the object module. The control step has finished processing. The output of this step is on SYSPCH, and should be passed to the link-edit step with the aid of DOS Job Control.

DOS Link-Edit Step Messages

The messages that may be issued during the execution of the DOS link-edit step are described in the *DOS System Control and Service Programs* publication, Order No. GC24-5036.

DOS Storage Step Messages

4F13I CONTROL CARD name INCORRECT - JOB TERMINATED

System Action: Processing is terminated.

Explanation/Programmer Response: The control card named (one of // OPTION =, // RPLACE, // DEVICE =, or // RPLACE =) is incorrect. Correct the control card and execute the job again.

4F14I NUMBER OF RPLACE CARDS EXCEEDS TWENTY - JOB TERMINATED

System Action: Processing is terminated.

Explanation/Programmer Response: The number of // RPLACE = control cards is greater than twenty. Only twenty // RPLACE = cards are allowed. Remove the excess cards and execute the job again.

4F15I ERROR IN SECTOR NAME FIELD - JOB TERMINATED

System Action: Processing is terminated.

Explanation/Programmer Response: The storage step has detected an error in the name portion of the 10-byte key field of a 486-byte sector. Check the name fields in the first module to be loaded on the ISAM file and correct the error before executing the job again.

4F16I ERROR IN FDP SECTOR - RECOMPILE FDP - JOB TERMINATED

System Action: Processing is terminated.

Explanation/Programmer Response: An error has been detected in the first module to be loaded. Check the FDP assembly and correct the error before executing the job again.

4F17I SECTOR INCOMPLETE - RECOMPILE FDP - JOB TERMINATED

System Action: Processing is terminated.

Explanation/Programmer Response: The first module to be loaded is incomplete. Check the FDP assembly and correct the error before executing the job again.

4F18I ERROR IN SECTOR COUNT FIELD - RECOMPILE FDP - JOB TERMINATED

System Action: Processing is terminated.

Explanation/Programmer Response: The count portion of the 10-byte key field of a 486-byte sector is incorrect or out of sequence. Check the FDP assembly and correct the error before executing the job again.

4F19I PRIME DATA AREA FULL OR OVERFLOW - ENLARGE DASD EXTENTS - JOB TERMINATED

System Action: Processing is terminated.

Explanation/Programmer Response: Not enough room has been allocated for the ISAM file. Enlarge the DASD extents to allow more space and execute the job again.

4F20I CYLINDER INDEX AREA FULL - ENLARGE DASD EXTENTS - JOB TERMINATED

System Action: Processing is terminated.

Explanation/Programmer Response: Not enough room has been allocated for the ISAM index area. Enlarge the DASD extents to allow more space and execute the job again.

4F21I ATTEMPT TO ADD DUPLICATE RECORD - JOB TERMINATED

System Action: Processing is terminated.

Explanation/Programmer Response: An attempt has been made to load the ISAM file with a record already existing on the file. Check the DLBL statement to ensure that ISC has been specified and execute the job again.

4F22I SEQUENCE ERROR - JOB TERMINATED

System Action: Processing is terminated.

Explanation/Programmer Response: The record being loaded is not in sequential order. Check the sequence in the count portion of the key field and correct the error before executing the job again.

4F23I UNRECOVERABLE I/O ERROR - JOB TERMINATED

System Action: Processing is terminated.

Explanation/Programmer Response: An unrecoverable I/O error has occurred on the ISAM file while trying to write a record. If the error persists, do the following before calling IBM for assistance:

- Have the job stream and program listing available.
- Have the console sheet available.

4F24I SUCCESSFUL COMPLETION OF LOAD PHASE

System Action: None.

Explanation/Programmer Response: The storage step LOAD phase is complete. To complete the loading of the ISAM file, IJLFST must be executed again with ISE specified in the DLBL card and // OPTION = LOADFST specified.

4F25I FORM name CONTAINS A NAME ERROR - FORM NOT ADDED

System Action: The form named is not added to the file. An attempt is made to add the next form.

Explanation/Programmer Response: The name indicated is the user's form name. An error has been detected in the name portion of the 10-byte key field of a 486-byte sector. Check the name fields in the form named and correct the error before executing the job again.

4F26I FORM name CONTAINS AN ERROR - FORM NOT ADDED

System Action: The form named is not added to the file. An attempt is made to add the next form.

Explanation/Programmer Response: The name indicated is the user's form name. An error has been detected in the form being added to the file. Check the FDP assembly and correct the error before executing the job again.

4F27I FORM name INCOMPLETE - FORM NOT ADDED

System Action: The form named is not added to the file. An attempt is made to add the next form.

Explanation/Programmer Response: The name indicated is the user's form name. The sector being added is incomplete. Check the FDP assembly and correct the error before executing the job again.

4F28I EOF ON IJFDLIB

System Action: Processing is terminated.

Explanation/Programmer Response: An EOF condition has been encountered while attempting to replace a form in the ISAM file. If the error persists, do the following before calling IBM for assistance:

- Have the job stream and program listing available.
- Have the console sheet available.

4F29I FORM name NOT FOUND

System Action: None.

Explanation/Programmer Response: An attempt was made to locate a form in order to replace it, but the form was not found. If the error persists, do the following before calling IBM for assistance:

- Have the job stream and program listing available.
- Have the console sheet available.

4F30I DATA AREA OVERFLOW ON IJFDLIB

System Action: Processing is terminated.

Explanation/Programmer Response: Not enough room has been allocated for the ISAM file. Enlarge the DASD extents to allow more space and execute the job again.

4F31I UNRECOVERABLE I/O ERROR ON IJFDLIB

System Action: Processing is terminated.

Explanation/Programmer Response: An unrecoverable I/O error has occurred on the ISAM file. If the error persists, do the following before calling IBM for assistance:

- Have the job stream and program listing available.
- Have the console sheet available.

4F32I FORM name ADDED

System Action: None.

Explanation/Programmer Response: The form named has been added to the ISAM file.

4F33I FORM name ASSIGNED TEMPORARY NAME IJLFTM0n

System Action: None.

Explanation/Programmer Response: The form name indicated is the user's form name. IJLFTM0n is the temporary name assigned the form when a duplicate name is detected and // RPLACE = name or // RPLACE is not coded. The "n" value in the temporary name may range from 0 to 9.

4F35I FORM name NOT ADDED - NO TEMPORARY NAMES LEFT

System Action: The form named is not added to the ISAM file. An attempt is made to add the next form.

Explanation/Programmer Response: The form name indicated is the user's form name. The storage step has discovered that all temporary names (IJLFTM00-09) have been used. The form named can be added only by replacing the form of the same name in the ISAM file.

4F36I COMPLETION OF STORAGE STEP

System Action: None.

Explanation/Programmer Response: The storage step has completed processing.

4F37I SECTOR COUNT ERROR IN FORM NAME

System Action: The form named is not added to the ISAM file. An attempt is made to add the next form.

Explanation/Programmer Response: The count portion of the 10-byte key field of the 486-byte data sector is incorrect or out of sequence. Check the FDP assembly and correct the error before executing the job again.

This DOS BTAM sample program reads data from the 3735, dumps it on the system printer if requested to do so, then sends FDPs (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.

```
PGTEST   CSECT
         BALR  R11,R0          ESTABLISH CSECT ADDRESSABILITY AND
         USING *,R11           DEFINE BASE REGISTER
         OPEN  IJFDLIB         OPEN IJFDLIB FILE
         LA    R5,L'OPNMSG     SET UP TO WRITE 'IJFDLIB OPEN - BTAM
         LA    R4,OPNMSG       NEEDED?' MESSAGE TO CONSOLE
         BAL   R6,GETLOG
         CLI   WTOIN,YESCHAR   IS RESPONSE LOWERCASE 'Y'?
         BE    BTAMOPEN        IF SO, OPEN BTAM DTFBT
PDUMPW   LA    R5,L'PDMSG      SET UP TO WRITE 'DUMPS WANTED?'
         LA    R4,PDMSG        MESSAGE TO CONSOLE
         BAL   R6,GETLOG
         CLI   WTOIN,YESCHAR   IS RESPONSE LOWERCASE 'Y'?
         BNE   CKBTAM          IF NOT, GO SEE IF BTAM IS WANTED
         OI    TSTFLAG,TEST2   IF PDUMPS WANTED, SET PDUMP FLAG
         B     GETISAM         GO TO SET UP FOR FDP RETRIEVAL
BTAMOPEN EQU   *
         OI    TSTFLAG,TEST1   TURN ON BTAM FLAG
         OPEN SRDTF            OPEN BTAM DTFBT
         B     PDUMPW          CHECK FOR PDUMPS
CKBTAM   EQU   *
         TM    TSTFLAG,TEST1   WAS BTAM SELECTED?
         BO    GETISAM         IF SO,GET ISAM RECORDS
         LA    R5,L'NOBTM      IF NOT, SET UP TO WRITE 'EOJ- DUMP
         LA    R4,NOBTM        WANTED?' MESSAGE TO CONSOLE
         BAL   R6,GETLOG
CKDUMP   CLI   WTOIN,YESCHAR   IS RESPONSE LOWERCASE 'Y'?
         BNE   EOJ             IF NOT, GO TO EOJ
DUMP     DUMP                  IF SO, DUMP
EOJ      EOJ                   END OF JOB
GETISAM  EQU   *
         LA    R5,L'ENTER      SET UP TO REQUEST FIRST FORM
         LA    R4,ENTER        NAME FROM CONSOLE
         BAL   R6,GETLOG
         MVC   KEYFLD(TEN),WTOIN  SET UP KEYARG FIELD
         SETL  IJFDLIB,GKEY    SET STARTING ADDRESS FOR FILE
GETRCD   EQU   *
         MVI   IJFDLIBC,NULCHAR    SET X'00' IN IJFDLIBC
         GET   IJFDLIB,WKAREA GET A RECORD
         L     R5,ADRDLIB      GET DTF ADDRESS
         TM    THIRTY(R5),TEST1       DASD ERROR?
         BO    DASDERR         IF SO, GO RETRY THE READ
EOFCHK   TM    THIRTY(R5),TEST3    END OF FILE?
         BO    EOF             IF SO, GO PREPARE TO STOP TRANSMISSION
BAL      BAL   R10,BTAM        GO TO BTAM SECTION TO TRANSMIT
         B     GETRCD          GET ANOTHER RECORD ON RETURN FROM BTAM
```

```
DASDERR  EQU   *
         ERET  RETRY           RETRY THE READ OPERATION
         TM    THIRTY(R5),TEST1   ANOTHER DASD ERROR?
         BZ    EOFCHK          IF NOT, GO CHECK FOR EOF
         LA    R5,L'UNREC      IF SO, PREPARE TO WRITE 'UNRECOVERABLE
         LA    R4,UNREC        I/O ERROR' MESSAGE TO CONSOLE
         BAL   R6,GETLOG
         B     CKDUMP          GO CHECK FOR 'DUMP WANTED' RESPONSE
EOF      EQU   *
         TM    TSTFLAG,TEST1   BTAM OPEN?
         BZ    EOF1            IF NOT, DONT CLOSE
         TM    TSTFLAG,TEST7   IS END-OF-FILE FLAG SET?
         BO    EOF12           IF SO, EXIT
         OI    TSTFLAG,TEST7   SET END OF FILE FLAG
         LA    R2,ENDMSG       GET ADDRESS OF 'END' MESSAGE
         B     TXTSEND         GO TO SEND 'END OF FDPS' MESSAGE
EOF12    EQU   *
         WRITE SRDECB,TR,MF=E    NO MORE DATA TO SEND - WRITE EOT
         BNE   DUMP            CHECK FOR GOOD SIO
         WAIT  ECB=SRDECB      WAIT FOR I/O COMPLETION
         CLI   SRDECB,NORMCOMP   NORMAL COMPLETION?
         BE    CHKEOTD         IF SO, GO CHECK FOR EOT RESPONSE
         BAL   R6,ERRCHK       IF NOT, GO CHECK FOR ERROR TYPE
CHKEOTD  EQU   *
         TM    SRDECB+EOTSPOT,TEST2   EOT RECEIVED?
         BO    DISCONT         IF SO, GO WRITE DISCONNECT
         TM    SRDECB+EOTSPOT,TEST3   DLE-EOT RECEIVED?
         BO    CLOSE           IF SO, GO WRITE DISCONNECT & CLOSE
DISCONT  EQU   *
         WRITE SRDECB,TD,MF=E WRITE DISCONNECT SEQUENCE
         BNE   DUMP            CHECK FOR GOOD SIO
         WAIT  ECB=SRDECB      WAIT FOR I/O COMPLETION
         B     EOF1            GO SET UP TO SHUT DOWN PROGRAM
CLOSE    EQU   *
         CONTROL SRDECB,TD,MF=E     DISCONNECT THE LINE
         WAIT  ECB=SRDECB      WAIT FOR I/O COMPLETION
         CLOSE SRDTF           CLOSE BTAM DTFBT
EOF1     EQU   *
         ESETL IJFDLIB         END SETL SPECIFICATION FOR IJFDLIB
         CLOSE IJFDLIB         CLCSE IJFDLIB FILE
         LA    R5,L'EOFMSG     PREPARE TO WRITE 'EOF ON IJFDLIB'
         LA    R4,EOFMSG       MESSAGE TO CONSOLE
         BAL   R6,GETLOG
         B    CKDUMP           GO CHECK FOR 'DUMP WANTED' RESPONSE
BTAM     EQU   *
         TM    TSTFLAG,TEST1   BTAM REQUESTED?
         BZ    PDUMP           IF NOT, PDUMP
         TM    TSTFLAG,TEST3   LINE CONNECTED AND DATA READ
         BO    SENDFDP         GO CHECK IF WRITE INITIAL NEEDED
         READ  SRDECB,TI,MF=E    ISSUE FIRST READ OPERATION
         BNZ   DUMP
         WAIT  ECB=SRDECB      WAIT FOR I/O COMPLETION
         PDUMP SRDECB,SRDECB+FORTY DUMP DECB INFORMATION
         CLI   SRDECB,NORMCOMP   NORMAL COMPLETION?
         BE    CHKEOT          IF SO, GO CHECK FOR EOT RESPONSE
         BAL   R6,ERRCHK       IF NOT, GO CHECK FOR ERROR TYPE
```

```
TIOK     EQU   *
         TM    TSTFLAG,TEST2  PDUMP WANTED?
         BZ    NOPDUMP        IF NOT, GO READ NEXT BLOCK
         PDUMP IOAREA,IOAREA+FIVEC IF SO, DUMP CONTENTS OF IOAREA
NOPDUMP  READ  SRDECB,TT,SRDTF,IOAREA,FIVEC,MF=E  READ NEXT BLOCK
         BNZ   DUMP
         WAIT  ECB=SRDECB     WAIT FOR I/O COMPLETION
         CLI   SRDECB,NORMCOMP  READ OK?
         BE    CHKEOT         IF SO, GO CHECK FOR EOT RESPONSE
         BAL   R6,ERRCHK      IF NOT, GO CHECK FOR ERROR TYPE
CHKEOT   TM    SRDECB+EOTSPOT,TEST2  EOT RECIEVED?
         BO    SENDFDP        IF SO, GO SET UP TO WRITE FDPS
         TM    SRDECB+EOTSPOT,TEST3  DLE-EOT RCVD?
         BO    CLOSE          IF SO, GO SET UP TO CLOSE
         B     TIOK           IF NOT, GO ISSUE ANOTHER READ
SENDFDP  EQU   *

         OI    TSTFLAG,TEST3  SET SEND CYCLE FLAG
         TM    TSTFLAG,TEST8  IS WRITE INITIAL NEEDED?
         BO    WRITETT        IF NOT, ISSUE WRITE TT
         OI    TSTFLAG,TEST8  SET WRITE TT FLAG
         WRITE SRDECB,TQ,SRDTF,IOAREA,FIVEC,INLIST,ZERO,MF=E
         BNE   DUMP
         WAIT  ECB=SRDECB     WAIT FOR I/O COMPLETION
         CLI   SRDECB,NORMCOMP  NORMAL COMPLETION?
         BE    WRITETT        IF SO, GO SET UP TO SEND FDP HEADER
         BAL   R6,ERRCHK      IF NOT, GO CHECK FOR ERROR TYPE
WRITETT  EQU   *
         LA    R2,FDPMSG      GET ADDRESS OF 'FDP HEADER' MESSAGE
         TM    TSTFLAG,TEST4  HAS 'FDP HEADER' MESSAGE BEEN SENT?
         BZ    TXTSEND        IF NOT, GO SEND IT
         MVI   STXSPOT,STXCHAR     IF SO, SET UP STX
         WRITE SRDECB,TT,SRDTF,STXSPOT,MSGLEN,MF=E SEND AN FDP BLOCK
         BNE   DUMP
         WAIT  ECB=SRDECB     WAIT FOR I/O COMPLETION
         CLI   SRDECB,NORMCOMP  NORMAL COMPLETION?
         BE    PDUMP          IF SO, GO CHECK FOR PDUMP REQUEST
ERRCHK   EQU   *
         CLI   SRDECB,ERRCOMP ERROR COMPLETION INDICATED?
         BNE   CHKAK          IF NOT, GO CHECK FOR WRONG ACK
         PDUMP SRDECB,SRDECB+FORTY IF SO, DUMP DECB INFORMATION, THEN
         B     EOF            GO TO CHECK FOR EOF
CHKAK    EQU   *
         CLI   SRDECB,WRONGACK WRONG ACK RECEIVED?
         BNE   CHKID          IF NOT, GO CHECK FOR WRONG ID
         WRITE SRDECB,TQ,SRDTF,IOAREA,MSGLEN,INLIST,MF=E    IF SO,
*              SEND FDP BLOCK AGAIN
         WAIT  ECB=SRDECB     WAIT FOR I/O COMPLETION
         CLI   SRDECB,NORMCOMP     NORMAL COMPLETION?
         BNE   ERRCHK         IF NOT, GO CHECK FOR ERROR TYPE
         BR    R6             RETURN VIA REGISTER 6
CHKID    EQU   *
         PDUMP SRLIST,SRLISTND     DUMP DFTRMLST INFORMATION
         PDUMP SRDECB,SRDECB+FORTY DUMP DECB INFORMATION
         B     CLOSE          GO SET UP TO CLOSE
PDUMP    TM    TSTFLAG,TEST2  PDUMP REQUESTED?
         BZ    BTAM1          IF NOT, GO SET UP TO GET NEXT FDP BLOCK
         PDUMP WKAREA,WKAREA+LASTONE IF SO, DUMP WKAREA CONTENTS
```

```
BTAM1    EQU   *
         BR    R10              GO TO GET ANOTHER FDP BLOCK
TXTSEND  EQU   *
         OI    TSTFLAG,TEST4    SET 'FDP HEADER' FLAG ON
         WRITE SRDECB,TT,SRDTF,(2),FIVE,SRLIST,ZERO,MF=E    SEND 'FDP
         BNE   DUMP                                  HEADER' MESSAGE
         WAIT  ECB=SRDECB       WAIT FOR I/O COMPLETION
         CLI   SRDECB,NORMCOMP      NORMAL COMPLETION?
         BE    CHKEOF1          IF SO, GO CHECK FOR EOF
         BAL   R6,ERRCHK        IF NOT, GO CHECK FOR ERROR TYPE
CHKEOF1  EQU   *
         TM    TSTFLAG,TEST7    IS EOF FLAG SET?
         BO    EOF123           IF SO, GO TO EXIT ROUTINE
         B     WRITETT          IF NOT, GO TO WRITE ANOTHER FDP BLOCK
EOF123   EQU   *
         WRITE SRDECB,TT,,TCMMSG,FIVE,,,MF=E SEND 'TERMINATE COMMUNI-
         BNE   DUMP                      CATE MODE' MESSAGE
         WAIT  ECB=SRDECB       WAIT FOR I/O COMPLETION
         CLI   SRDECB,NORMCOMP    NORMAL COMPLETION ?
         BE    EOF12            IF SO, GO TO EXIT ROUTINE
         BAL   R6,ERRCHK        IF NOT, GO CHECK FOR ERROR TYPE
         B     EOF12            GO TO EXIT ROUTINE
GETLOG   EQU   *
         OUTLOG BUFFER=(4),COUNT=(5),RETURN=NO    COMMUNICATE WITH
         MVI   WTOIN,BLANK      CONSOLE --  SET UP FOR RESPONSE
         MVC   WTOIN+ONE(SEVEN),WTOIN
         LA    R4,WTOIN
         INLOG BUFFER=(4),CCUNT=(8)
         BR    R6               GO TO ANALYZE RESPONSE
         EJECT
*****************************************
****     DCB'S, DC'S, & ETC.     **********
*****************************************
WTOIN    DS    8X               WORK AREA FOR CONSOLE RESPONSE
         DC    X'0000'
         DS    0F
INAREA   DS    500X             INPUT AREA
         DS    0F
KEYFLD   DS    10X              KEY FIELD
WKAREA   DS    486X             WORK AREA FOR SENDING
         DC    X'03'            ETX CHARACTER
IOAREA   DC    500X'00'         I/O AREA FOR RECEIVING
TSTFLAG  DC    X'00'            FLAGS BYTE
EOFMSG   DC    C'EOF ON IJFDLIB - DUMP WANTED?'
OPNMSG   DC    C'IJFDLIB OPEN - IS BTAM NEEDED?'
PDMSG    DC    C'PDUMPS WANTED?'
UNREC    DC    C'UNRECOVERABLE I/O ERROR - DUMP WANTED?'
ENTER    DC    C'ENTER FIRST FORM NAME'
NOBTM    DC    C'NO BTAM = NO PDUMPS = EOJ - DUMP WANTED?'
FDPMSG   DC    X'0200C60003'  'BEGIN FDP TRANSMISSION' MESSAGE
ENDMSG   DC    X'0200C50003'  'END OF FDPS' MESSAGE
TCMMSG   DC    X'020CE30003'  'TERMINATE COMMUNICATE MODE' MESSAGE
         DS    0F
INLIST   DC    A(IOAREA,50)
         READ  SRDECB,TI,SRDTF,IOAREA,500,SRLIST,0,MF=L  CREATE DECB
SRLIST DFTRMLST IDLST,0,6,C1C1C1C1C12D,5,C1C1C1C1C1
SRLISTND EQU   *
```

```
SRDTF    DTFBT LINELST=(004),CU=2701,DEVICE=BSC2,FEATURE=(BSC,SIW,RIW),-
               MODNAME=SRMOD,CTLCHAR=EBCDIC,MODELST=(0)
IJFDLIB  DTFIS DSKXTNT=3,IOROUT=RETRVE,KEYLEN=10,NRECDS=1,RECFORM=FIXUN-
               B,RECSIZE=476,CYLOFL=8,DEVICE=2311,ERREXT=YES,HINDEX=231-
               1,IOAREAR=INAREA,IOREG=(3),KEYARG=KEYFLD,MODNAME=RDMOD, -
               TYPEFLE=RANSEQ,IOAREAS=INAREA,WORKS=YES
*****************************************
****     EQUATES - REGISTERS     **********
*****************************************
R0       EQU   0               REGISTER 0
R2       EQU   2               REGISTER 2
R4       EQU   4               REGISTER 4
R5       EQU   5               REGISTER 5
R6       EQU   6               REGISTER 6
R10      EQU   10              REGISTER 10
R11      EQU   11              REGISTER 11
*****************************************
****     EQUATES - CONSTANTS     **********
*****************************************
ZERO     EQU   0               NUMERIC CONSTANT
CNE      EQU   1               NUMERIC CONSTANT
FIVE     EQU   5               NUMERIC CONSTANT
SEVEN    EQU   7               NUMERIC CONSTANT
TEN      EQU   10              NUMERIC CONSTANT
EOTSPOT  EQU   24              DECB FLAGS LOCATION
THIRTY   EQU   30              NUMERIC CONSTANT
FORTY    EQU   40              NUMERIC CONSTANT
MSGLEN   EQU   478             NUMERIC CONSTANT - MESSAGE LENGTH
LASTONE  EQU   487             NUMERIC CONSTANT
FIVEC    EQU   500             NUMERIC CONSTANT
STXSPOT  EQU   WKAREA+9        SPOT WHERE STX IS INSERTED IN DATA
ADRDLIB  EQU   A(IJFDLIB)      ADDRESS CONSTANT
NORMCOMP EQU   X'7F'           NORMAL COMPLETION
ERRCOMP  EQU   X'41'           ERROR COMPLETION
WRONGACK EQU   X'60'           WRONG ACK RECEIVED
NULCHAR  EQU   X'00'           'NULL' CHARACTER
BLANK    EQU   X'40'           BLANK CHARACTER
STXCHAR  EQU   X'02'           STX CHARACTER
YESCHAR  EQU   X'A8'           LOWER CASE 'Y'
TEST1    EQU   X'80'           TEST FLAG 1
TEST2    EQU   X'40'           TEST FLAG 2
TEST3    EQU   X'20'           TEST FLAG 3
TEST4    EQU   X'10'           TEST FLAG 4
TEST7    EQU   X'02'           TEST FLAG 7
TEST8    EQU   X'01'           TEST FLAG 8
         END
```

This OS BTAM sample program reads data from the 3735, places it in the data set that has the ddname PRINT, then sends FDPs (if any) to the 3735. When through the program sends the power down message to the 3735 and concludes processing.

```
PG3735   CSECT
         PRINT NOGEN
         SAVE  (14,12)                  SAVE CALLER'S REGISTERS
ENTRY    BALR  BASEREG,R0               ESTABLISH CSECT ADDRESSABILITY
         USING *,BASEREG                AND BASE REGISTER
         USING IECTDECB,DECBREG
         ST    SAVEREG,SAVESPOT         SAVE CALLER'S SAVEAREA ADDRESS
         LA    SAVEREG,SAVE             LOAD ADDRESS OF MY SAVEAREA
         WTO   'PG3735 HAS BEGUN EXECUTION'
         LA    DECBREG,MYDECB
         OPEN  (PRINT,(OUTPUT))
         OPEN  (SNAPDCB,(OUTPUT))
         OPEN  (MYDCB)
         TM    OPENCHK,GOODCHK          DID OPEN COMPLETE SUCCESSFULLY
         BO    BEGIN                    IF SO ISSUE READ CONNECT
         WTO   'OPEN ERROR NO EXECUTION'
         B     EXIT
ERRBLOCK LERB  1
BEGIN    EQU   *
         LA    CTREG,TWO
         READ  MYDECB,TI,MF=E
         BAL   R9,TIO                   CHECK SIO CONDITION CODE
CKEOT    EQU   *
         CLI   AREA,EOT                 HAS EOT BEEN RECEIVED
         BE    WTQ
         PUT   PRINT,AREA               PRINT DATA LINE
         MVI   AREA,BLANK               BLANK FIRST CHARACTER
         MVC   AREA+ONE(ALL),AREA       CLEAR FIRST 256 CHARACTERS
         MVC   AREA+ALL(LAST),AREA      CLEAR LAST 230 CHARACTERS
         LA    CTREG,TWO
         READ  MYDECB,TT,MF=E
         BAL   R9,TIO                   CHECK SIO CONDITION CODE
         B     CKEOT
WTQ      WRITE MYDECB,TQ,MF=E           WRITE ENQ
         BAL   R9,TIO
         CLOSE PRINT
         OPEN  (DISK,INPUT)
         MVI   AREA,STX
**       IF NO FDP HAS TO BE SENT CODE
**       DISCO  DD    DUMMY
**       THE FOLLOWING GET CAUSES A BRANCH TO ENDMEMB
         GET   DISK,DIREC     FOR ONE DIRECTORY BLOCK
         GET   DISK,DIREC2
ENDDIR   EQU   *
         CLOSE DISK
         CLC   DIREC,DIREC2
         BE    CLOSDIS                  NO FDP SENDING
```

```
         LA     R11,DIREC       BEGINNING OF 256-BYTE DATA AREA
         AH     R11,DIREC       USE COUNT TO FIND END OF DATA AREA
         S      R11,LENGTH      SUBTRACT LENGTH OF COUNT FIELD
         MVC    DIREC,DIREC+TWO
         MVC    ZERO(254,R11),DIREC2+TWO
         LA     CTREG,TWO
WTFDP    WRITE  MYDECB,TT,MYDCB,SENDFDP,5,MF=E    PREPARE FDP SENDING
         BAL    R9,TIO                    CECK SIO CONDITION CODE
         OPEN   (DISKPO,INPUT)
         LA     R3,DIREC
FINDA    MVC    FINDNAME,ZERC(R3)
         FIND   DISKPO,FINDNAME,D
LL       READ   DATA,SF,DISKPO,AREA+ONE
         CHECK  DATA
         LA     CTREG,TWO
         MVI    AREA+477,ETB              INSERT ETB CHARACTER
WTT      WRITE  MYDECB,TT,,AREA,478,MF=E
         BAL    R9,TIO                    CHECK SIO CONDITION CODE
         B      LL
ENDMEMB  EQU    *
         SR     R11,R11
         IC     R11,ELEVEN(R3)            LENGTH IN HALFWORD
         N      R11,A31
         LA     R11,TWELVE(R11,R11)       LENGTH IN BYTE
         AR     R3,R11
         CLC    FFFF,ZERO(R3)
         BNE    FINDA
CLOSDIS  CLOSE  DISKPO
         LA     CTREG,TWO
         WRITE  MYDECB,TT,MYDCB,SENDTXT,5,MF=E    END OF FDP'S
         BAL    R9,TIO
         LA     CTREG,TWO
         WRITE  MYDECB,TT,MYDCB,PWRDOWN,5,MF=E    POWER DOWN TERMINAL
         BAL    R9,TIO
         LA     CTREG,TWO
         WRITE  MYDECB,TR,MF=E                    WRITE  EOT
         BAL    R9,TIO
WTD      WRITE  MYDECB,TD,MF=E
         BAL    R9,TIO                    CHECK SIO CONDITION CODE
         WTO    'PG3735 HAS SUCCESFULLY COMPLETED'
CLOSE    EQU    *
         CLOSE  (MYDCB)                   CLOSE LINE DCB
         CLOSE  (SNAPDCB)                 CLOSE THE SNAP DCB
EXIT     EQU    *
         L      SAVEREG,SAVESPOT          RESTORE SAVEAREA ADDRESS
         RETURN  (14,12)                  RESTORE CALLER'S REGISTERS
TIO      LTR    R15,R15                   EXCP ISSUED
         BZ     WAIT                      ISSUE WAIT IF GOOD SIO
         WTO    'SIO WAS NOT GOOD'
         SNAP   ID=4,MF=(E,SDUMP)         LOOK AT DECFLAGS
         B      CLOSE
WAIT     EQU    *
         WAITR  1,ECB=(DECBREG)           WAIT FOR COMPLETION
         TM     DECTYPE+ONE,READCHK       IS THIS A READ OPERATION
         BZ     WRTRTN                    IF NOT,GO TO WRITE
         EJECT
```

```
*****************************************
****     READ  ERRORS    ****************
*****************************************
         TM    DECSDECB,NORMAL          WAS ECB POSTED NORMALLY
         BNO   COMPL41                  IF NOT CHECK ERROR
         CLI   DECFLAGS,NULL            ARE ALL FLAGS ZERO
         BE    ZERO(R9)                 IF SO, CONTINUE NORMALLY
         SNAP  DCB=SNAPDCB,ID=5,PDATA=(REGS),STORAGE=(ANSRLIST,SD)
         CLI   DECFLAGS,GOODCHK         WAS AN INVALID ID RECEIVED
         BNE   WTD                      IF NOT DISCONNECT LINE
         WTO   'AN INVALID ID WAS RECEIVED'
         B     WTD
COMPL41  EQU   *
         SNAP  DCB=SNAPDCB,ID=6,PDATA=(REGS),STORAGE=(ANSRLIST,SD)
*        DECSENS0=01      TEXT TIME-OUT IF TP OP CODE = '11'
*        DECSENS0=01  NON-TEXT TIME-OUT IF TP OP CODE = '07'

         TM    DECSENS0,FLAGCHK         WAS ERROR DATA CHECK
         BNO   FINISH                   IF NOT,PRINT ERROR MESSAGE
         BCT   CTREG,REPEAT             FIRST DATA CHECK
         B     FINISH                   SECOND DATA CHECK
REPEAT   EQU   *
         TM    DECERRST,FLAGCHK         LINE DISABLED BY ERP
         BO    FINISH
         READ  MYDECB,TP,MF=E
         B     TIO
*****************************************
****    WRITE  ERRORS    ****************
*****************************************
WRTRTN   TM    DECSDECB,NORMAL          WAS ECB POSTED NORMALLY
         BNO   WRERR                    IF NOT CKECK ERROR
         CLI   DECFLAGS,NULL            ALL FLAGS ZERO
         BE    ZERO(R9)                 IF SO, CONTINUE NORMALLY
         CLI   DECTYPE+ONE,OPTYPE       IS OPERATION WRITE ENQ
         BNE   WTD                      IF NOT, DISCONNECT THE LINE
*                                       WRONG ACK RECEIVED
         CLI   DECFLAGS,WACKCHK         WAS WACK RECEIVED
         BNE   FIN1
         BCT   CTREG,WTQ                YES, RETRY 1 TIME
FIN1     EQU   *
         SNAP  DCB=SNAPDCB,ID=7,PDATA=(REGS),STORAGE=(ANSRLIST,SD)
         B     WTD
WRERR    EQU   *
         SNAP  DCB=SNAPDCB,ID=8,PDATA=(REGS),STORAGE=(ANSRLIST,SD)
         CLI   DECSENS0,ERRCHK          WAS ERROR TIME OUT
         BNE   FINISH                   IF NOT PRINT ERROR MSG
         TM    DECERRST,FLAGCHK         LINE DISABLED BY.ERP
         BO    FINISH
         BCT   CTREG,AGAIN              RETRY 1 TIME
FINISH   WTO   'ERROR CANNOT BE HANDLED BY PROGRAM'
         B     CLOSE
AGAIN    WRITE MYDECB,T,MF=E            REISSUE LAST WRITE
         B     TIO
SDUMP    SNAP  DCB=SNAPDCB,ID=1,PDATA=(REGS),STORAGE=(ANSRLIST,SD),MF=L
         EJECT
```

```
*****************************************
****     EQUATES - REGISTERS     **********
*****************************************
R0        EQU   0               REGISTER 0
R3        EQU   3               REGISTER 3
DECBREG   EQU   7               REGISTER 7  - DECB REGISTER
R9        EQU   9               REGISTER 9
CTREG     EQU   10              REGISTER 10 - COUNTER REGISTER
R11       EQU   11              REGISTER 11
BASEREG   EQU   12              REGISTER 12 - BASE REGISTER
SAVEREG   EQU   13              REGISTER 13 - SAVE AREA REGISTER
R15       EQU   15              REGISTER 15
*****************************************
****     EQUATES - CONSTANTS     **********
*****************************************
ZERO      EQU   0               NUMERIC CONSTANT
ONE       EQU   1               NUMERIC CONSTANT
TWO       EQU   2               NUMERIC CONSTANT
ELEVEN    EQU   11              NUMERIC CONSTANT
TWELVE    EQU   12              NUMERIC CONSTANT
ALL       EQU   256             NUMERIC CONSTANT
LAST      EQU   230             NUMERIC CONSTANT
NULL      EQU   X'00'           FLAG CHECK BYTE
ERRCHK    EQU   X'01'           FLAG CHECK BYTE
STX       EQU   X'02'           STX CHARACTER
FLAGCHK   EQU   X'08'           FLAG CHECK BYTE
GOODCHK   EQU   X'10'           FLAG CHECK BYTE
OPTYPE    EQU   X'16'           FLAG CHECK BYTE
ETB       EQU   X'26'           ETB CHARACTER
EOT       EQU   X'37'           EOT CHARACTER
BLANK     EQU   X'40'           SPACE CHARACTER
NORMAL    EQU   X'7F'           FLAG CHECK BYTE
WACKCHK   EQU   X'C0'           FLAG CHECK BYTE
SAVESPOT  EQU   SAVE+4          ADDRESS CONSTANT
OPENCHK   EQU   MYDCB+48        ADDRESS CONSTANT
*****************************************
****     DCB'S, DC'S, & ETC.     **********
*****************************************
SNAPDCB   DCB   DSORG=PS,RECFM=VBA,MACRF=W,BLKSIZE=1632,LRECL=125,      *
                DDNAME=SNAPSW,DEVD=PR
PRINT     DCB   DSORG=PS,                 PHISICAL SEQUENTIAL ORGANZATION *
                MACRF=PM,                 PUT MOVE MACROS                 *
                DDNAME=PRINT              NAME OF DD CARD
DISK      DCB   DSORG=PS,                 SEQUENTIAL  ORGANIZATION        *
                MACRF=GM,                 GET MOVE                        *
                DDNAME=DISCO,             NAME OF DD CARD                 *
                EODAD=ENDDIR,             END OF DIRECTORY                *
                SYNAD=DUMP2,              I/O ERROR                       *
                RECFM=U,                  UNDEFINED RECORD                *
                BLKSIZE=256
DISKPO    DCB   DSORG=PO,                 PARTITIONED ORGANIZATION        *
                MACRF=R,                  READ ONLY                       *
                EODAD=ENDMEMB,            END OF DATA SET ADDRESS         *
                DDNAME=DISCO,             NAME OF DD CARD                 *
                SYNAD=DUMP2               I/O ERROR ADDRESS
```

```
DUMP2    ABEND 2,DUMP
         DS    0D
ANSRLIST DFTRMLST  BSCLST,0,6,C1C1C1C1C12D,2,1070
         READ  MYDECB,TI,MYDCB,AREA,480,ANSRLIST,1,MF=L
MYDCB    DCB   DSORG=CX,DEVD=BS,MACRF=(R,W),DDNAME=TERMINAL,                *
               LERB=ERRBLOCK,EROPT=TC
AREA     DC    4CL120' '
SD       DS    0F
SAVE     DC    18F'0'
DIREC    DC    CL256' '
DIREC2   DC    CL256' '
SENDTXT  DC    X'0200C50026'       SEND TEXT TO 3735          SN NE
SENDFDP  DC    X'0200C60026'       SEND OBJECT FDP TO 3735 *TU UT*
PWRDOWN  DC    X'0200E30003'       END OF TRANSMISSION        XL LB
FINDNAME DS    D                   (BUT LEAVE TERMINAL UP)
LENGTH   DC    F'2'
A31      DC    F'31'
FFFF     DC    X'FFFFFFFFFFFFFFFF'
READCHK  DC    X'01'
    LTORG
*        DCBD  DEVD=BS,DSORG=CX
         IECTDECB
         END
```

This chart is a condensation of basic 3735 operating procedures. Complete details of 3735 operations are contained in the 3735 Operator's Guide, Order No. GA27-3061.

OPERATION SELECTION

SELECT MODE

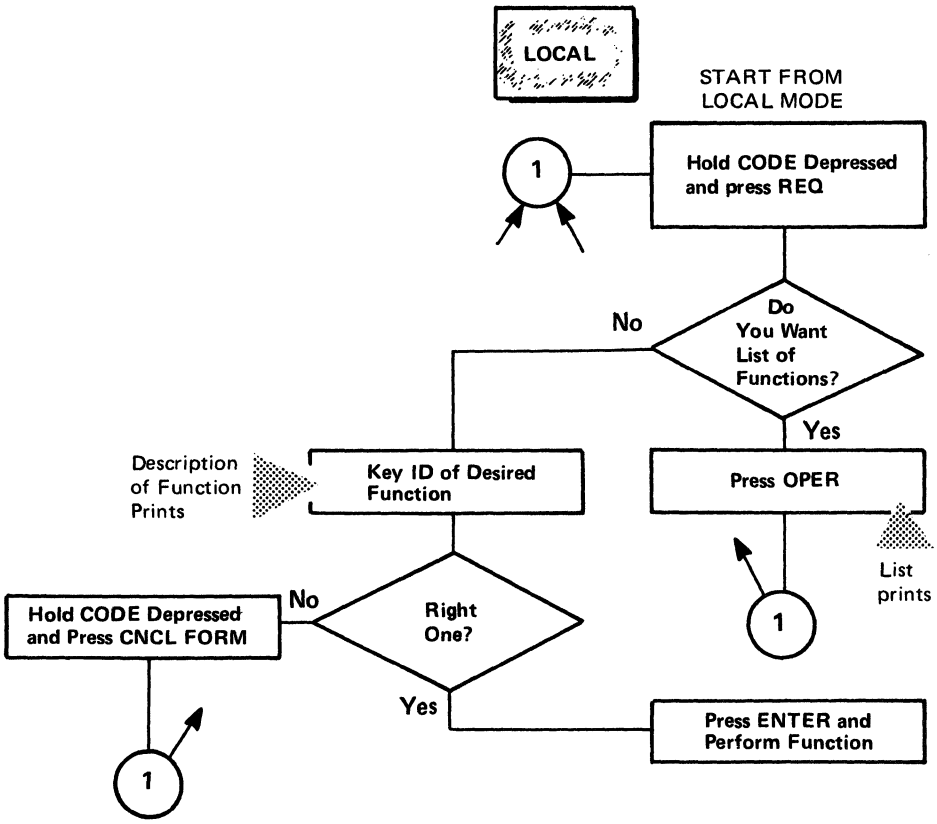
is on and you want to:

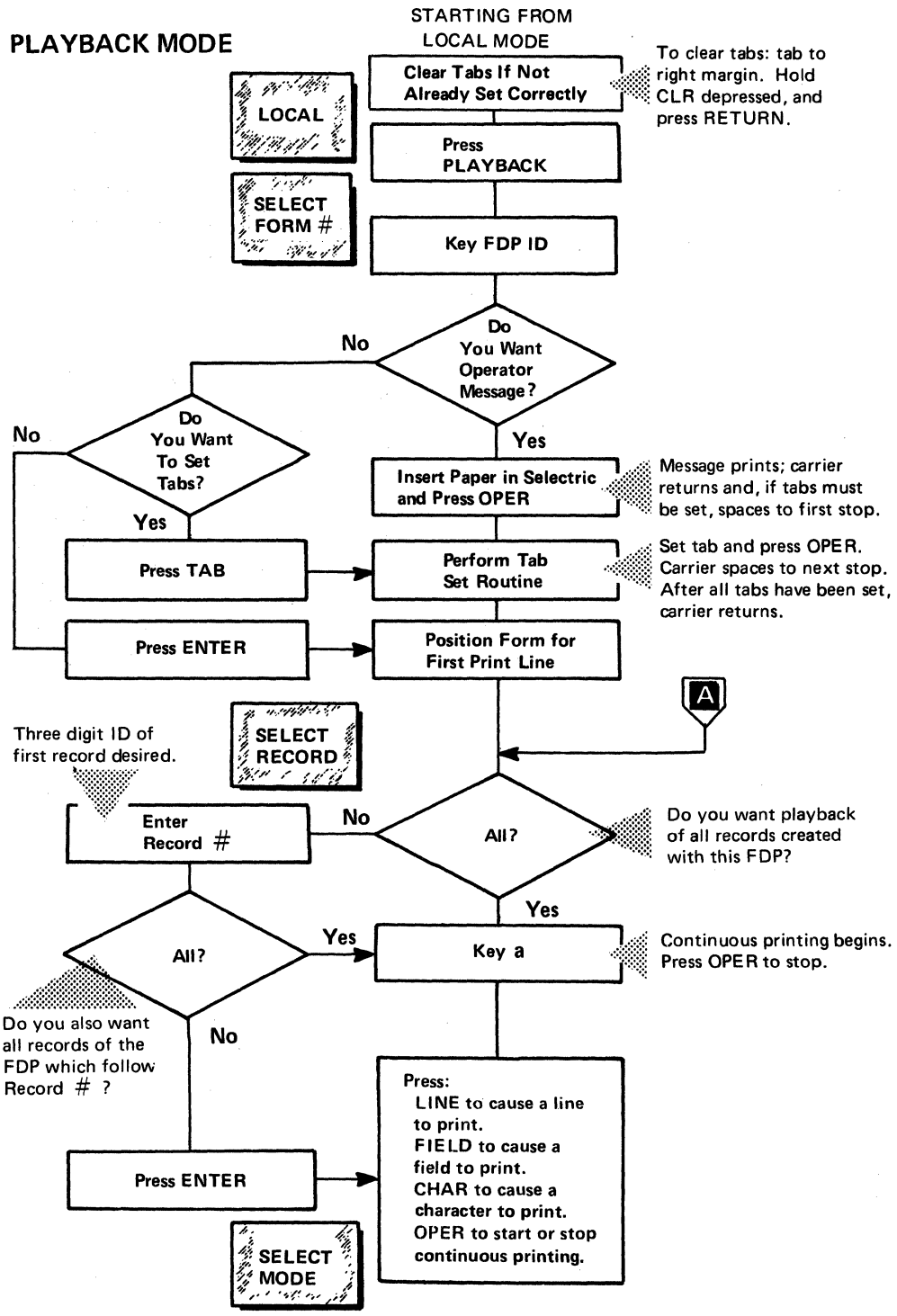
1. **Playback a record created with active FDP:**
   a. **Press PLAYBACK.**
   b. **Position form for its first print line.**
   c. **Refer to A under PLAYBACK MODE.**

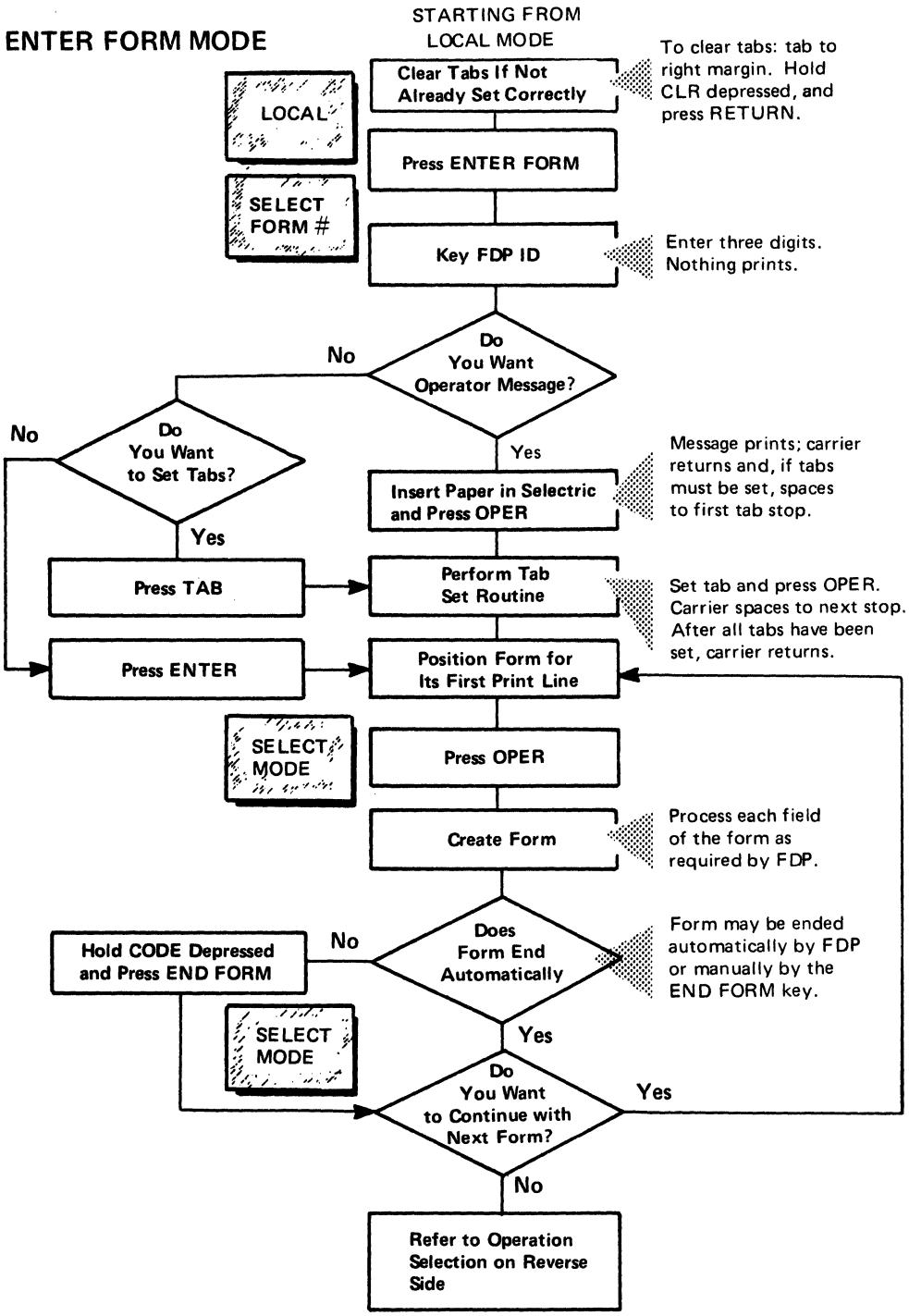
2. **Create or playback a form created with a different FDP:**
   a. **Press LOCAL.**
   b. **Begin ENTER FORM MODE or PLAYBACK MODE procedure at the beginning.**

3. **Perform a request function:**
   a. **Press LOCAL.**
   b. **Refer to REQUEST MODE.**

REQUEST MODE




PLAYBACK MODE
STARTING FROM LOCAL MODE
Clear Tabs If Not Already Set Correctly
To clear tabs: tab to right margin. Hold CLR depressed, and press RETURN.
LOCAL
Press PLAYBACK
SELECT FORM #
Key FDP ID
Do You Want Operator Message?
No
Yes
Do You Want To Set Tabs?
No
Yes
Insert Paper in Selectric and Press OPER
Message prints; carrier returns and, if tabs must be set, spaces to first stop.
Press TAB
Perform Tab Set Routine
Set tab and press OPER. Carrier spaces to next stop. After all tabs have been set, carrier returns.
Press ENTER
Position Form for First Print Line
A
Three digit ID of first record desired.
SELECT RECORD
Enter Record #
No
All?
Do you want playback of all records created with this FDP?
Yes
All?
Yes
Key a
Continuous printing begins. Press OPER to stop.
Do you also want all records of the FDP which follow Record # ?
No
Press:
LINE to cause a line to print.
FIELD to cause a field to print.
CHAR to cause a character to print.
OPER to start or stop continuous printing.
Press ENTER
SELECT MODE



IBM 3735 Programmable Buffered Terminal Condensed Operating Instructions.

INDICATED ERRORS (Keyboard is locked and a check light is on)

ALPHA/ NUMERIC	Press OPER. Key correct character.	RANGE	Press OPER. Re-key field.
PROCE- DURAL	Press OPER. Continue with correct operating procedure.	BUFFER FULL	Complete form then refer to your Operator's Guide: Request Mode, Communicate Mode.
SELF CHECK	Press OPER. Verify check-digit Re-key field.	MACHINE CHECK	Refer to your Operator's Guide: Problem Determination.
LENGTH	Press OPER. Complete entry of field, or press FIELD and re-key field.	INPUT OUTPUT	Correct I/O device condition.

NON-INDICATED ERRORS (You have noticed the errors)

In field being processed.	Use either of the following: 1. Re-key field: a. Press FIELD. b. Re-key. 2. Backspace for correction: a. Backspace to error. b. Re-key. c. Use ADV key to reach original point.
In a previous field on the same line.	a. Press LINE. b. Use FIELD and CHAR keys to advance to error. c. Re-key. d. Press LINE.
On a previous line (for correction of forms already stored, use the second procedure).	Use either of the following: 1. Cancel form and start again: a. Hold CODE depressed and Press CNCL FORM. b. Insert next form. c. Press OPER. 2. Complete form and playback for correction: a. Complete form. b. Insert next form. c. Press PLAYBACK. d. Key three digit record number. e. Press Enter. f. Use LINE, FIELD and CHAR to advance to error. g. Key correction. h. Press OPER.

This appendix provides detailed information on use of the Katakana character set options in the Form Description macros. The areas affected are:

- FDFORM macro instruction: DEVICES operand; MESSAGE operand.
- FDFIELD macro instruction: SOURCE, KIND, UL, COMPARE, IND, and PICTURE operands.

The following paragraphs discuss the coding of each affected operand.

DEVICES = (3735,K[D])

The DEVICES operand is an optional operand that may be specified in the FDFORM macro. If it is not coded, the FD macros treat all other operands as applying to the domestic 3735. If it is coded as either DEVICES = (3735,K) or DEVICES = (3735,KD), the FD macros treat all other operands as applying to the Katakana 3735. Coding the qualifier D also causes the FD macros to display in the Assembler listing the phonetic results of character-string translation in the string-handling operands described later (MESSAGE, SOURCE, COMPARE, and IND).

MESSAGE = ({ cc[({ 1 / d })] / 'string' } [, { cc[({ 1 / d })] / 'string' }])

If Katakana has not been specified (through coding the DEVICES operand), then the MESSAGE operand is coded as explained in the operand description in the body of this publication. If Katakana has been specified, then the rules for encoding 'string' are changed as follows:

1. The characters that can be printed at the 3735 are restricted to the kana characters (48 characters and the currency symbol), the arabic digits (0 through 9), the graphics (asterisk, comma, period, and space), and the Roman alphabetic characters (A through Z, uppercase only).
2. Doubled apostrophes in the 'string' do not signify the graphic apostrophe, but instead divide the 'string' into alternating k-strings and r-strings, as follows:

   'k-string' 'r-string' 'k-string' 'r-string' . . .

   A k-string is always the first, third, or other odd-numbered division in 'string'. An r-string is always the second, fourth, or other even-numbered division in 'string'. This division of 'string' into k-strings and r-strings allows the FD macros to distinguish between Katakana characters encoded in Roman letters (Romaji) and actual Roman characters.
3. Kana characters will appear in the message at the 3735 if they are encoded in a k-string. Kana characters may be coded as their assigned EBCDIC codes (that is, multi-punched), or may be encoded in Roman letter (Romaji). The use of Romaji characters is explained in detail in the "Character Aids" section later in this Appendix.
4. Arabic digits will appear in the message at the 3735 if they are encoded in either a k-string or an r-string. The graphic characters will appear if they are encoded in either a k-string or an r-string, with one exception. The exception is that the first blank character following a Romaji encoding in a k-string is not translated, in keeping with standard practice for English transliteration of Japanese.

   For example, coding MESSAGE = 'KOKObEbIRASSHAI.' produces a string of 10 characters, including the period, in the 3735 message. The standard Japanese practice of writing kana characters without spaces between words is observed. Coding MESSAGE = 'KOKObbEbbIRASSHAI .' produces a string of 12 characters, including the period and the blank characters between the three words.

5. Roman characters will appear in the message at the 3735 if they are encoded in an r-string. If the message is to contain Roman characters only, then a null k-string is required (that is, two adjacent apostrophes immediately following the left framing apostrophe). For example, coding MESSAGE = 'BATTERII' causes the 3735 message to produce the six kana characters that signify the Japanese word battery. Coding MESSAGE = ' ' 'BATTERY' produces the English word battery in uppercase Roman letters. Coding MESSAGE = ' ' ' BATTERY IS WRITTEN ' ' BATTERII' produces the 3735 message BATTERY IS WRITTEN kkkkkk, where the letters kkkkkk represent the generated kana characters.

SOURCE = 'string'

If Katakana has not been specified, then the KIND operand is coded as explained in the operand description in the body of this publication. If Katakana has been specified, and SOURCE = 'string' is coded, then 'string' must follow the rules previously described for the MESSAGE operand of FDFORM.

KIND = { U / A / N / AN / K }

If Katakana has not been specified, then the KIND operand is coded as explained in the operand description in the body of this publication. KIND may not be coded as K unless Katakana was specified in the DEVICES operand of FDFORM, since this encoding would erroneously request that a Katakana check be performed on a non-Katakana 3735. If Katakana has been specified, any encoding of KIND is permitted. Coding KIND = K restricts the incoming characters to the 49 recognized kana characters and the blank character.

UL = ({ NO / YES } [, { NO / YES }] . . .)

If Katakana has been specified in the DEVICES operand of FDFORM, any coding of the UL operand is ignored, since the character set for the Katakana 3735 does not include the underscore character.

COMPARE = ([FIELD,] comparopr,comparand
[, { AND / OR } ,[FIELD,] comparopr,comparand] . . .)

If Katakana has not been specified, then the COMPARE operand is coded as explained in the operand description in the body of this publication. If Katakana has been specified, then the format of the comparand remains the same, except that 'string' comparands must follow the coding rules previously described for the MESSAGE operand of FDFORM.

When Katakana has been specified, and the COMPARE operand is coded, the characters allowed in 'string' must be restricted to the valid characters permitted as input to the field when the FDP is executed at the 3735. The permitted character sets are determined by the coding of the KIND operand. Thus, if KIND = K is in affect, the only character allowed in an r-string is the blank character. If KIND = A is in affect, the only character allowed in a k-string is the blank character. If KIND = AN is in affect, the only characters allowed in a k-string are the Arabic digits and the blank character.

IND = ((d, logexp) [, (d,logexp)] . . .)

If Katakana has not been specified, then the IND operand is coded as explained in the operand description in the body of this publication. If Katakana has been specified, then the coding of each logical expression (logexp) must follow the rules and restrictions defined previously for the COMPARE operand.

PICTURE = (['picturespec'] [,['picturespec']] . . .)

If Katakana has not been specified, then the PICTURE operand is coded as explained in the operand description in the body of this publication. If Katakana has been specified, 'picturespec' must not include the characters / , + , or S. Other valid PICTURE characters may be used normally.

Character Coding Aids

The encoding (multi-punching) of Katakana characters in the EBCDIC code defined for those characters may be diffcult for customers who lack either Katakana-featured key-punches or Katakana-featured printers on which to print the assembly listing. Without a Katakana printer, customers will not be able to see, in the listing, the characters they have multi-punched in their source statements. The FD macros provide two aids to overcoming the difficulties: MNOTE messages that display the contents of each Katakana string, and use of Romaji to avoid multi-punching the Katakana characters.

Katakana Display MNOTEs

These coding aids allow the forms encoder to verify the correctness of the Katakana translation performed in processing character-string operands. If DEVICES = (3735,KD) is coded in the FDFORM macro, then MNOTE messages describing the string are produced for each character string processed in the MESSAGE operand of FDFORM, or the SOURCE, COMPARE, or IND operands of FDFIELD.

For example, the MNOTES might display such information as:

```
IDF397 START OF STRING
IDF398 DIGIT 0
IDF398 DIGIT 5
IDF399 END OF STRING

IDF397 START OF STRING
IDF398 GRAPHIC *
IDF399 END OF STRING

IDF397 START OF STRING
IDF398 ROMAN A
IDF398 ROMAN B
IDF399 END OF STRING

IDF397 START OF STRING
IDF398 HA
IDF398 N
IDF398 TO
IDF398 DATUKEN (HARD VOWEL)
IDF398 RU
IDF399 END OF STRING
```

The last example would result from coding SOURCE = 'HANDORU' in the FDFIELD macro.

Romaji Character Coding

These coding aids remove the need for multi-punching Katakana characters, by allowing the use of Romaji in k-strings within 'string'. The Romaji system in the FD macros is similar to other Romaji systems, except that slight modifications have been made to reduce ambiguities present in other systems. The following chart represents the common Romaji and the Romaji used in the FD macros for the 49 Katakana characters permitted with the 3735 terminal.

Common Romaji	FDM Romaji	Common Romaji	FDM Romaji
a	A	ha	HA or WA
i	I	hi	HI
u	U	fu	FU.
e	E or (E)	he	HE or E
o	O	ho	HO
ka	KA	ma	MA
ki	KI	mi	MI
ku	KU		
ke	KE	mu	MU
ko	KO	me	ME
sa	SA	mo	MO
shi	SHI	ya	YA
su	SU	yu	YU
se	SE	yo	YO
so	SO	ra	RA
ta	TA	ri	RI
chi	CHI	ru	RU
tsu	TSU	re	RE
te	TE	ro	RO
to	TO	wa	WA or (WA)
na	NA	n	N or M
ni	NI	(hard vowel)	n. a.
nu	NU	(soft vowel)	n. a.
ne	NE	(currency)	YEN
no	NO	(long vowel-"ichi")	(YI)

Notes:

1. If E is coded as a separate word, with a blank character preceding and following, it is translated as HE. You can code (E) to force translation of the other E as a separate word.
2. If WA is coded as a separate word, with a blank character preceding and following, it is translated as HA. You can code (WA) to force translation of the other WA as a separate word.
3. YEN is used to represent the currency symbol.
4. You can code (YI) to force the generation of the long vowel symbol ("ichi").
5. The hard vowel and soft vowel symbols and the terminal "n" (sometimes written as "m" are generated in the translation of the FD macro Romaji. To force their generation, you can encode them using the defined EBCDIC codes (multi-punching).

An additional 63 Romaji encodings are translated into two or three output characters. These are hard and soft derivatives of the basic syllabic set presented in the previous chart, the so-called kana compounds (syllables ending in -ya, -yu, and -yo), and the syllables "fo" and "fa". In the following chart, "hv" represents the hard vowel symbol, and "sv" represents the soft vowel symbol.

Common Romaji	FDM Romaji	Expansion	Common Romaji	FDM Romaji	Expansion
ga	GA	KA+hv	kya	KYA	KI+YA
gi	GI	KI+hv	kyu	KYU	KI+YU
gu	GU	KU+hv	kyo	KYO	KI+YO
ge	GE	KE+hv	sha	SHA	SHI+YA
go	GO	KO+hv	shu	SHU	SHI+YU
za	ZA	SA+hv	sho	SHO	SHI+YO
ji	JI	SHI+hv	cha	CHA	CHI+YA
zu	ZU	SU+hv	chu	CHU	CHI+YU
ze	ZE	SE+hv	cho	CHO	CHI+YO
zo	ZO	SO+hv	nya	NYA	NI+YA
da	DA	TA+hv	nyu	NYU	NI+YU
ji	DI	CHI+hv	nyo	NYO	NI+YO
zu	DU	TSU+hv	hya	HYA	HI+YA
de	DE	TE+hv	hyu	HYU	HI+YU
do	DO	TO+hv	hyo	HYO	HI+YO
ba	BA	HA+hv	mya	MYA	MI+YA
bi	BI	HI+hv	myu	MYU	MI+YU
bu	BU	FU+hv	myo	MYO	MI+YO
be	BE	HE+hv	rya	RYA	RI+YA
bo	BO	HO+hv	ryu	RYU	RI+YU
pa	PA	HA+sv	ryo	RYO	RI+YO
pi	PI	HI+sv	gya	GYA	KI+hv+YA
pu	PU	FU+sv	gyu	GYU	KI+hv+YU
pe	PE	HE+sv	gyo	GYO	KI+hv+YO
po	PO	HO+sv	ja	JA	SHI+hv+YA
			ju	JU	SHI+hv+YU
fo	FO	FU+O	jo	JO	SHI+hv+YO
fa	FA	FU+A	ja	DYA	CHI+hv+YA
			ju	DYU	CHI+hv+YU
			jo	DYO	CHI+hv+YO
			bya	BYA	HI+hv+YA
			byu	BYU	HI+hv+YU
			byo	BYO	HI+hv+YO
			pya	PYA	HI+sv+YA
			pyu	PYU	HI+sv+YU
			pyo	PYO	HI+sv+YO

Notes:

1. The encodings DI, DU, DYA, DYU, and DYO are used to prevent occurrences of what are ambiguities in common Romaji.
2. FO is expanded into FU+O, FA into FU+A.

If the Romaji encoding contains any of the doubled consonants PP, TT, KK, SS, BB, DD, GG, or TC, the FD macros generate the character STU to represent the doubled consonant. If N or M is doubled, the terminal N is generated in the expansion of the preceding syllable.

The long vowels are represented by coding a hyphen (-) after a, e, o, and u, or by doubling i. Coding a long vowel causes the output of the long vowel character ("ichi"), except for the long o, for which the output character is U.

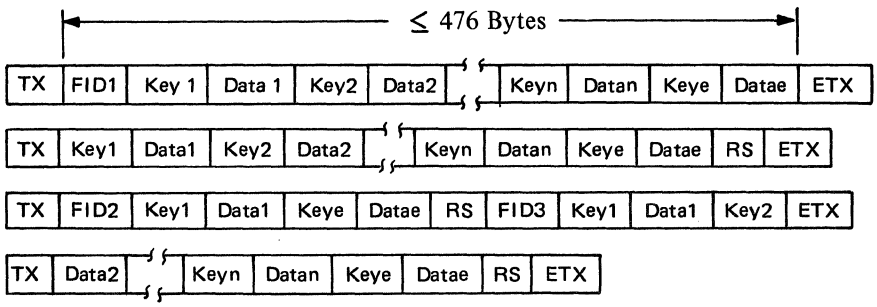
Appendix K. Summary of 3735 Data and Command Functions

The chart in this appendix should be helpful in pointing out where data is actually obtained and what operations are actually performed in the three main modes of 3735 operation (Enter Form, Error Correct, and Playback). The word *source* indicates that the data is obtained from the source specified in the FDP. The word *disk* indicates that the data is obtained from the data records on the customer area of the 3735 disk file.

Function	3735 Operating Mode		
	Enter Form	Error Correct	Playback
SOURCE			
1. FID, RSN, 'string', CTR, IDR	source	source	source
2. STG	source	source	disk
3. INQ, RDR, LPB, CCR	source	disk	disk
SINK			
1. STG, INQ	stored	stored	not stored
2. PCH, LPB	stored	stored	stored
FUNCTION PERFORMED			
Counter operations	yes	yes	yes
Indicator operations	yes	yes	yes
Selectric print (sink)	yes	yes	yes
Clear STG, INQ	yes	no	no
Clear PCH, LPB, IDR, CCR	yes	yes	yes
Send (Inquiry operation)	yes	no	no
Read RDR	yes	no	no
Read IDR	yes	yes	yes
Read CCR	yes	no	no
Punch 5496 cards	yes	yes	yes
Print 3286 data	yes	yes	yes
Skip (3286)	yes	yes	yes
Skipto (3286)	yes	yes	yes

The FDLOAD macro allows you to write an FDP that can use CPU-generated data to load or update the 3735 disk storage file when the File Storage capability is present. To load or update the file from CPU data you must have (1) data records in a standard format that have been transmitted from the CPU and (2) an FDP with only FDFORM, FDLOAD, and FDEND macros to load the file with the data records. You may transmit these data records to the 3735 with the same program that transmits FDPs provided the records follow the FDP transmission immediately. You may also transmit these records with a separate transmitting program. For more information concerning transmitting programs, refer to the sample BTAM program in Appendix G and Appendix H.

The CPU-generated data records should be in the following format for transmission to the 3735:



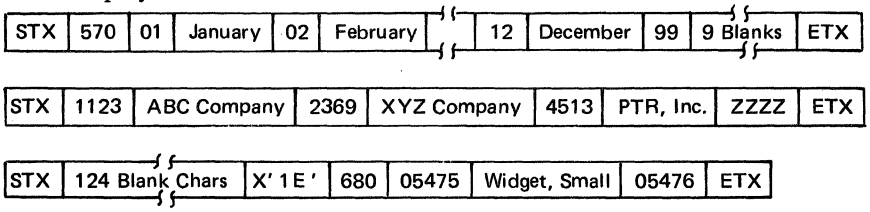
where

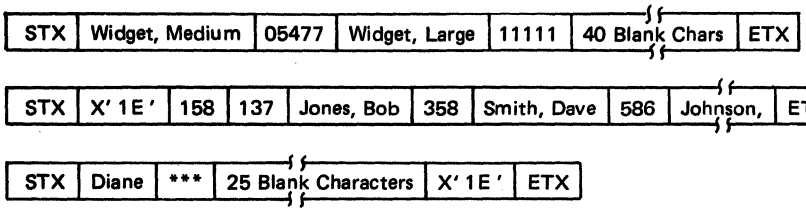
FID	is the 3-digit form ID of the FDLOAD FDP that is to read the data records and write them on the 3735 disk storage file.
Keyn - Datan	represents the last record of a logical file.
Keye - Datae	represents the dummy record whose key matches the string specified in the ENDCHAR operand of the corresponding FDLOAD macro and signals the end of the logical file.
RS	represents the record separator character X' 1E '.
ETX	is the end-of-text control character.

To load or update the file with the preceding data would require three FDLOAD FDPs, assuming that each FID is different. The first FDP (FID1) would have two FDLOAD macros, one for each logical file. The following two FDPs (FID2 and FID3) would have only one FDLOAD macro each.

Example:
Suppose that you have the following data records:

1. a month-by-month table look-up file and a customer name file.
2. an inventory file.
3. an employee file.





The section of code for the FDPs that load the file might be as follows:

```
FILELOAD    FDFORM FID = '570' ,MODE = LOAD
            FDLOAD KEYLEN = 2 ,DATALEN = 9 ,ENDCHAR = ' 99 '
CUSFILE     FDLOAD KEYLEN = 4 ,DATALEN = 124 ,ENDCHAR = ' ZZZZ '
            FDEND
INVLOAD     FDFORM FID = ' 680 ' ,MODE = LOAD
INVFILE     FDLOAD DATALEN = 40 ,KEYLEN = 5 ,ENDCHAR = ' 11111 '
            FDEND
EMPLOAD     FDFORM FID = ' 158 ' ,MODE = LOAD ,MESSAGE = (' EMPLOYEE
              FILE LOAD FDP ')
EMPFILE     FDLOAD KEYLEN = 3 ,DATALEN = 25
            FDEND
```

If this is the first load operation to the file since the last file purge or terminal control program system generation, the data records are written to the file in the order that they are processed. If a record being loaded has the same key as a record already existing in the file, the old record is overlayed with the new one. Both of these records should be the same length. All new records (without matching keys) being added to an already existing file are written following the last existing record in the file.

	SEL KEY	5496 PUNCH	3286 PRINT	XMIT HEX
				00
				05
				0D
*				15
				16
				1C
				1E
				25
	SP	SP	SP	40
**				4A
	.	.	.	4B
**				4C
	(	(	(	4D
	+	+	+	4E
**				4F
	&	&	&	50
	!	!	!	5A
	$	$	$	5B
	*	*	*	5C
	)	)	)	5D
	;	;	;	5E
**				5F
	-	-	-	60
	/	/	/	61
	,	,	,	6B
	%	%	%	6C
	-	-	-	6D
**				6E
	?	?	?	6F
	:	:	:	7A
	#	#	#	7B
	@	@	@	7C
	'	'	'	7D
	=	=	=	7E
	"	"	"	7F
	a	A	A	81
	b	B	B	82
	c	C	C	83
	d	D	D	84
	e	E	E	85
	f	F	F	86
	g	G	G	87
	h	H	H	88
	i	I	I	89
	j	J	J	91
	k	K	K	92
	l	L	L	93
	m	M	M	94
	n	N	N	95
	o	O	O	96
	p	P	P	97

	SEL KEY	5496 PUNCH	3286 PRINT	XMIT HEX
	q	Q	Q	98
	r	R	R	99
	±	SP	-	9E
	s	S	S	A2
	t	T	T	A3
	u	U	U	A4
	v	V	V	A5
	w	W	W	A6
	x	X	X	A7
	y	Y	Y	A8
	z	Z	Z	A9
	[	SP	-	AD
	]	SP	-	BD
	A	A	A	C1
	B	B	B	C2
	C	C	C	C3
	D	D	D	C4
	E	E	E	C5
	F	F	F	C6
	G	G	G	C7
	H	H	H	C8
	I	I	I	C9
**				D0
	J	J	J	D1
	K	K	K	D2
	L	L	L	D3
	M	M	M	D4
	N	N	N	D5
	O	O	O	D6
	P	P	P	D7
	Q	Q	Q	D8
	R	R	R	D9
	S	S	S	E2
	T	T	T	E3
	U	U	U	E4
	V	V	V	E5
	W	W	W	E6
	X	X	X	E7
	Y	Y	Y	E8
	Z	Z	Z	E9
	0	0	0	F0
	1	1	1	F1
	2	2	2	F2
	3	3	3	F3
	4	4	4	F4
	5	5	5	F5
	6	6	6	F6
	7	7	7	F7
	8	8	8	F8
	9	9	9	F9

5496 READ	SEL PRINT	3286 PRINT	XMIT HEX
¢	—	¢	4A
<	—	<	4C
=	—	=	4F
¬	—	¬	5F
>	—	>	6E
}	—	—	D0

*Valid text transmission characters without associated graphics.

**See 5496 Read.

EBCDIC Code

	SEL KEY	5496 PUNCH	3286 PRINT	XMIT HEX
				00
				08
				09
*				0A
				0D
				1C
				1E
	SP	SP	SP	20
	!	!	!	21
	"	"	"	22
	#	#	#	23
	$	$	$	24
	%	%	%	25
	&	&	&	26
	'	'	'	27
	(	(	(	28
	)	)	)	29
	*	*	*	2A
	+	+	+	2B
	,	,	,	2C
	-	-	-	2D
	.	.	.	2E
	/	/	/	2F
	0	0	0	30
	1	1	1	31
	2	2	2	32
	3	3	3	33
	4	4	4	34
	5	5	5	35
	6	6	6	36
	7	7	7	37
	8	8	8	38
	9	9	9	39
	:	:	:	3A
	;	;	;	3B
	<	<	<	3C

	SEL KEY	5496 PUNCH	3286 PRINT	XMIT HEX
	=	=	=	3D
	>	>	>	3E
	?	?	?	3F
	@	@	@	40
	A	A	A	41
	B	B	B	42
	C	C	C	43
	D	D	D	44
	E	E	E	45
	F	F	F	46
	G	G	G	47
	H	H	H	48
	I	I	I	49
	J	J	J	4A
	K	K	K	4B
	L	L	L	4C
	M	M	M	4D
	N	N	N	4E
	O	O	O	4F
	P	P	P	50
	Q	Q	Q	51
	R	R	R	52
	S	S	S	53
	T	T	T	54
	U	U	U	55
	V	V	V	56
	W	W	W	57
	X	X	X	58
	Y	Y	Y	59
	Z	Z	Z	5A
	[	SP	[	5B
	\	SP	\	5C
	]	SP	]	5D
	^	SP	^	5E
	—	—	—	5F
**			-	7D

5496 READ	SEL PRINT	3286 PRINT	XMIT HEX
|	—	—	5C
}	—	—	7D
¬	—	—	2D
¢	—	—	2D

Notes:

1. ASCII transmission code is available for the Unites States and Canada only.

2. ASCII transmission code supports alphabetic upper case only. If lower case characters are received from the line, they are folded to upper case.

3. A mono case print element for the Selectric is provided as standard with ASCII transmission code. A dual case element is available as a special feature. The dual case element differs from the mono case as follows:

   Dual case has upper and lower case alphabetic characters.
   Dual case does not have the \ character.

*Valid text transmission characters without associated graphics.

**See 5496 Read.

ASCII Code

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*, Order No. GC20-1699.

IBM is grateful to the American National Standards 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: Any of the data management techniques available to the user for transferring data between main storage and an input/output device.

Application program: Any program written by a user that applies to his own work.

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.*

Batch terminal: A terminal that accumulates and groups a number of input items to be sent to a central computer at one time for processing.

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).

Branch: (1) A set of instructions that are executed between two successive decision instructions. (2) To select a branch as in (1). (3) A direct path joining two nodes of a network or graph. (4) Loosely, a conditional jump.*

BSC: See binary synchronous communications.

Buffer: A routine or device used to compensate for difference in rate of flow of data, or time of occurrence of events, when transmitting data from one device to another.*

Central processing unit: (ISO). A unit of a computer that includes circuits controlling the interpretation and execution of instructions. (Synonymous with "main frame".)*

Common carrier: A company which furnishes communications services to the general public, and which is regulated by appropriate local, state, or federal agencies.

Communication lines: A medium over which data signals are transmitted.

Control character: A character whose occurence in a particular context initiates, modifies, or stops a control operation - for example, a character to control carriage return.*

CPU: see central processing unit.

*American National Standard definition.

Data set: (1) The major unit of data storage and retrieval in the operating system, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access. (2) A device which performs the modulation/demodulation and control functions necessary to provide compatibility between business machines and communication facilities.

Delimiting macro: The FDEND Form Description macro statement that closes the description of each form and prepares for a form description that may follow.

FD macro: see Form Description macro.

FD utility: see Form Description utility.

FDP: see form description program.

File: A collection of related records treated as a unit. For example, one line of an invoice may form an item, a complete invoice may form a record, a complete set of such records may form a file, the collection of inventory control files may form a library, and the libraries used by an organization are known as its data bank.*

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 processing 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 program blocks and writes the blocks into a user-specified data set.

Forms encoder: A person who designs and defines forms by coding Form Description macro statements.

Identification (ID) characters: Characters sent by a BSC terminal on a switched line to identify the terminal.

Keyword operand: An operand whose functions and meaning are known by its use of a special word (the keyword).

Linkage Editor: A processing program that prepares the output of language translators for execution. It combines separately produced object or load modules; resolves symbolic cross references among them; replaces, deletes, and adds control sections, and generates overlay structures on request; and produces executable code (a load module) that is ready to be fetched into main storage.

Macro instruction: An instruction in a source language that is equivalent to a specified sequence of machine instructions.*

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.

Modulo: The remainder after any division has been performed.

Multipoint line: A communication line or circuit that connects more than one terminal; also known as multidrop line.

*American National Standard definition.

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.*

Operand: That which is operated upon. An operand is usually identified by an address part of an instruction.*

Point-to-point line: A communication line that connects a single remote terminal to the computer. It may be either switched or nonswitched.

Positional operand: An operand whose function and meaning are known by its position in relation to other operands.

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 facility of the computing system or operating system required by a job or task, and including main storage, input/output devices, the central processing unit, data sets, and control processing programs.

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.

Sink: Pertaining to the destination of field data in a form description program.

Source: Pertaining to the origin of field data in a form description program.

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.

Suboperand: A subfield of an operand.

Summary block: A group of form lines that are specified so as to follow a cyclically-repeated group of FD macros a fixed number of form lines beyond the end of the repeated group.

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.

TCP: see terminal control program.

Telecommunications: Pertaining to the transmission of signals over long distances, such as by telegraph, radio, or television.*

Teleprocessing: A form of information handling in which a data processing system utilizes communication facilities. (Originally an IBM trademark.)

Terminal: A point in a system at which data can enter, leave, or enter and leave.*

Terminal control program: The microcoded 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 electrical transfer of a signal, message, or other form of intelligence from one location to another.

*American National Standard definition.

The following are publications referred to in the body of this programmer's guide. This list is annotated to refer readers who need further information about 3735 operations, telecommunications access methods, or System/360 and System/370 data processing techniques to the appropriate manuals.

General Information – Binary Synchronous Communications, Order No. GA27-3004. This publication describes the Binary Synchronous Communication (BSC) procedures in general terms. The major topics covered are: BSC concepts (including transmission codes and data-link operation), message formats, additional data-link capabilities, and planning considerations. Readers should be familiar with the concepts presented in this publication before attempting to establish a teleprocessing network that includes 3735 terminals as remote stations.

IBM 3735 Programmable Buffered Terminal:

Concept and Application, Order No. GA27-3043. This publication provides an introduction to the 3735. Readers should be familiar with the concepts presented in it before attempting to use this programmer's guide to design and implement form description programs that support 3735 operations.

Operator's Guide, Order No. GA27-3061. This publication contains detailed operating instructions for the 3735.

Operating System (OS):

PL/I Reference Manual, Order No. GC28-8201. This publication describes the rules for writing PL/I programs for compilation under the OS F-level compiler. While some application programs designed to process 3735-collected data may be written in PL/I, the sections in this publication of primary interest to the user of this programmer's guide will be the discussion of the PL/I PICTURE specifications.

Assembler Language, Order No. GC28-6514. This publication contains specifications for the OS Assembler Language (including macro instructions and conditional assembly facilities). Although knowledge of Assembler Language is not required to code the Form Description macro instructions, BTAM application programs are written in Assembler Language. In addition, this publication describes the Assembler listing control instructions that may be used to control the appearance of FD macro output listings.

Assembler (F) Programmer's Guide, Order No. GC26-3756. This publication provides information on program assembling, linkage editing, executing, interpreting listings, and assembler programming.

Linkage Editor and Loader, Order No. GC28-6538. This publication describes the operation of the Linkage Editor and Loader programs under the System/360 and System/370 Operating System. Readers will find this publication useful when they want to establish or modify a library of application programs to support the 3735 data gathering operations.

TCAM Programmer's Guide and Reference Manual, Order No. GC30-2024. This publication is a reference manual and coding guide for the programmer who must construct or modify a TCAM Message Control Program (MCP), or who must write a TCAM-compatible application program. Readers should be familiar with the concepts and macro instructions in this publication before attempting to use TCAM to control teleprocessing between a CPU and a 3735.

Basic Telecommunications Access Method, Order No. GC30-2004. This publication describes the Basic Telecommunications Access Method (BTAM) used with the System/360 and System/370 OS control program. BTAM provides the READ/WRITE level macro instructions for the assembler-language programmer who is implementing programs for telecommunications applications. Readers should be familiar with the contents of this publication before attempting to use BTAM to control teleprocessing activity between a CPU and a 3735.

Disk Operating System (DOS):

PL/I Subset Reference Manual, Order No. GC28-8202. This publication describes the rules for writing PL/I subset programs for compilation under the DOS D-level compiler. While some application programs designed to process 3735-collected data may be written in PL/I, the sections in this publication of primary interest to the user of this programmer's guide will be the discussions of the PL/I PICTURE specifications.

Assembler Language, Order No. GC24-3414. This publication contains specifications for the DOS Assembler Language (including macro instructions and conditional assembly facilities). Although knowledge of Assembler Language is not required to code the Form Description macro instructions, BTAM application programs are written in Assembler Language. In addition, this publication describes the Assembler listing control instructions that may be used to control the appearance of FD macro output listings.

System Control and Service Programs, Order No. GC24-5036. This publication describes the set of control programs and processing programs that make up the Disk Operating System. Readers should be especially familiar with the sections in this publication that discuss multiprogramming and telecommunications.

Basic Telecommunications Access Method, Order No. GC30-5001. This publication describes the Basic Telecommunications Access Method (BTAM) used with the System/360 and System/370 DOS control program. BTAM provides the READ/WRITE level macro instructions for the assembler-language programmer who is implementing programs for telecommunications applications. Readers should be familiar with the contents of this publication before attempting to use BTAM to control teleprocessing activity between a CPU and a 3735.

Data Management Concepts, Order No. GC24-3427. This publication describes the file formats, labeling procedures, and access methods available in DOS. Readers should be familiar with the concepts presented in this publication before attempting to use the DOS FD utility to create their ISAM file of form description programs.

Index

READER'S COMMENT FORM

IBM 3735 Programmer's Guide
(OS and DOS System)

Order No. GC30-3001-2

- 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	Good	Fair	Poor	Very Poor
As a text:	.....	....	.....	.....	.....
	Very Good	Good	Fair	Poor	Very 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

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)

GC30-3001-2

IBM 3735 Programmer's Guide Printed in U.S.A. GC30-3001-2

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)