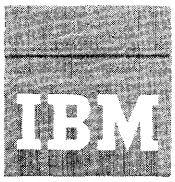
File Number S360-30
Order No. GC24-5037-12

DOS



Systems Reference Library

DOS Supervisor and I/O Macros

This reference publication is for the programmer planning to use the DOS Input/Output Control System macro instructions and supervisor communication macro instructions. These macro instructions can be combined with problem programs to produce generalized or specific file processing programs for a foreground or background environment. The major subjects describe the macro system, label processing, multitasking macros, physical IOCS, supervisor communication macros, and sequential, direct, and indexed sequential access methods. Prerequisite information is contained in the following publications.

Note: Although titles of some DOS publications have been simplified, the change does not affect the contents of the publications.

DOS System Programmer's Guide, GC24-5073

DOS DASD Labels, GC24-5072

IBM System/360 Principles of Operation, GA22-6821

IBM System/360 Disk and Tape Operating Systems, Assembler Language, GC24-3414

DOS Data Management Concepts, GC24-3427

For titles and abstracts of other associated publications, see the IBM System/360 and System/370 Bibliography, GA22-6822.

DOS Release 26.2

Thirteenth Edition (October, 1973)

This is a reprint of GC24-5037-11 incorporating changes released in the following Technical Newsletters:

GN33-8737 (January 1, 1973)
GN33-8757 (July 13, 1973).

This edition applies to Release 26.2 of the IBM Disk Operating System and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the specifications herein; before using this publication in connection with the operation of IBM Systems, consult the latest System/360 and System/370 SRL Newsletter, GN20-0360, 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.

Comments concerning the contents of this publication may be addressed to IBM Laboratory, Publications Department, P.O.Box 24, Uithoorn, The Netherlands. Comments become the property of IBM.

Preface

This reference publication is a guide for the programmer planning to use the DOS Input/Output Control System macro instructions and supervisor communication macro instructions. It describes the macros and their various operands. The publication has ten sections. The first two sections, Introduction and Macro System, introduce concepts and terminology. The third section describes Label Processing for all files. The fourth, fifth, and sixth sections describe the logical IOCS macro instructions for the Sequential Access Method, Direct Access Method, and Indexed Sequential Access Method, respectively. The seventh section contains the Multitasking Macros and the information required to use the multitasking function. The eighth section contains the Physical IOCS processing information. The ninth section describes the Supervisor-Communication Macros and the procedures for checkpointing a program. The tenth section discusses Additional Macro Instructions used in program linkage.

The problem programmer should be familiar with the publications listed in the abstract on the front cover of this manual, with every pertinent device manual, and with the following related publications.

Notes: Although titles of some DOS publications have been simplified, the change does not affect the contents of the publications. In case of discrepancies between this manual and IBM-supplied DOS component publications (such as: guides for language translators, sorts, utilities, etc, and specifications manuals), observe the specific restrictions of the component.

DOS System Control and Service, GC24-5036

DOS System Generation, GC24-5033

IBM System/360 Disk Operating System: Basic Telecommunications Access Method, GC30-5001

IBM System/360 Disk Operating System: Queued Telecommunications Access Method, Message Control Program, GC30-5004

IBM System/360 Disk Operating System: Queued Telecommunications Access Method, Message Processing Program Services, GC30-5003

IBM 1419 Model 32 Attached to IBM System/360 Models 25, 30, 40, 50, or 65, GA19-0023

IBM 2671 Paper Tape Reader and IBM 2822 Paper Tape Reader Control, GA24-3388

IBM System/360 Component Descriptions: 2826 Paper Tape Control Unit, 1017 Paper Tape Reader, and 1018 Paper Tape Punch, GA33-4500

IBM 1219 Reader Sorter; IBM 1419 Magnetic Character Reader, GA24-1499

IBM 1259 Magnetic Character Reader Component Description, GA24-3500

IBM System/360 Disk Operating System: User's Guide, GC20-1685

IBM 1275 Optical Reader/Sorter Component Description, GA19-0034.

Contents

Figures

MACHINE REQUIREMENTS

Minimum features required:

- 16K bytes of main storage (24K bytes are required for multiprogramming and/or MICR [Magnetic Ink Character Recognition] document processing).
- Standard instruction set. See Note 1.
- One I/O channel (either multiplexor or selector). See Notes 2 and 3.
- One Card Reader (IBM 1442, 2501, 2520, or 2540). See Note 4.
- One Card Punch (IBM 1442, 2520, or 2540). See Note 4.
- One Printer (IBM 1403, 1404, 1443, or 3211). See Note 4.
- One Printer-Keyboard. For an IBM System/360 Model 65 or larger, use the multiplexor channel.
- One Disk Storage Drive.

Note 1: Language translators may require extended instruction sets and/or additional main storage.

Note 2: Telecommunications require two channels, one for telecommunications and the other for a system resident device. An IBM 2701 attached to a System/360 Model 25 must be placed on the multiplexor channel.

Note 3: MICR processing requires at least two I/O channels. A DOS supervisor supporting MICR devices on the multiplexor channel cannot support burst mode devices on the same channel. MICRs should be attached as the highest priority device on the multiplexor channels. The IBM 1255, 1259, 1270, 1275, 1412, or 1419 with Single Address Adapter is supported on any selector channel, but device performance is maintained only if a selector channel is dedicated to a single MICR device. The IBM 1419 or 1275 with Dual Address Adapter is not attachable to selector channels.

Also, MICR processing requires either the direct-control feature or the external-interrupt feature.

Note 4: One 3420 or 2400-series magnetic tape unit may be substituted for this device. (7- or 9-track. If 7-track tape units are used, the data-conversion feature is required, except when substituted for a printer.)

Additional features supported:

- Timer feature.
- Simultaneous Read-While-Write Tape Control (IBM 2404 or 2804).
- Any channel configuration up to one multiplexor channel and six selector channels.
- Tape Switching Unit (IBM 2816).
- Storage-Protection feature (required for multiprogramming).
- Universal Character Set.
- Selective Tape Listing features (IBM 1403) for continuous paper tapes.
- Dual Address Adapter (IBM 1419, 1275) to allow more stacker selection processing. When this feature is used, MICR devices with a Single Address Adapter cannot be accessed by the system.
- Additional main storage up to 16,777,216 bytes.

Problem programs can request I/O operations on the following IBM devices:

- 1442 Card Read Punch
- 2501 Card Reader
- 2520 Card Read Punch
- 2540 Card Read Punch
- 1403 Printer
- 1404 Printer (for continuous forms only)
- 1443 Printer
- 1445 Printer
- 3211 Printer
- 1052 Printer-Keyboard (used for operator communication)

- 2671 Paper Tape Reader
- 1017 Paper Tape Reader with 2826 Paper Tape Control Model 1
- 1018 Paper Tape Punch with 2826 Paper Tape Control Model 1
- 2311 Disk Storage Drive
- 2314 Direct Access Storage Facility
- 2319 Disk Storage
- 2321 Data Cell Drive
- 2400-Series Magnetic Tape Units
- 3420 Magnetic Tape Unit
- 1285 Optical Reader (maximum of 8 optical readers are supported)
- 1287 Optical Reader (maximum of 8 optical readers are supported)
- 1288 Optical Reader
- 1255 Magnetic Character Reader
- 1259 Magnetic Character Reader
- 1412 Magnetic Character Reader (maximum number supported depends upon system configuration)
- 1419 Magnetic Character Reader (maximum number of devices supported depends upon system configuration)
- 2495 Tape Cartridge Reader
- Teleprocessing devices specified in BTAM and QTAM publications referenced in the Preface
- 1270 Optical Reader/Sorter (not available in the United States of America)
- 1275 Optical Reader/Sorter (not available in the United States of America)
- 3210 or 3215 Console Printer-Keyboard (System/370)

Macro Similarities

Macro similarities and differences between Basic Programming Support (BPS), Basic Operating System (BOS), Tape Operating System (TOS), and Disk Operating System (DOS) are:

- Imperative macro instructions and Supervisor Communication macro instructions available for TOS have identical expansions for DOS.
- File definition macro instructions available for TOS have identical expansions for DOS.
- Symbolic programs written for BPS and BOS can be assembled into the functional equivalents for DOS. However, the DTFSR macro instruction substantially prolongs program preparation time and, in many situations, execution time when used on DOS.
- The PRTOV macro implemented in BOS differs from that in DOS. In BOS, once a channel 9 or 12 is sensed during printing or carriage spacing, the overflow indicator is reset by every print or carriage control command. Thus, previous BOS programs used in DOS should be checked to determine whether the test for overflow after every carriage motion is made.
- Certain register parameters for BPS and BOS are enclosed in parentheses in DOS. In general, DOS accepts the parameters without the parentheses. For example, in DOS the correct format is IOREG=(r). DOS accepts the BOS operand IOREG=r.
- No change in register usage conventions is planned for BOS or BPS. However, to avoid compatibility problems resulting from transition between 8K and 16K support levels, installations using 8K support should observe the 16K register conventions indicated under Register Usage.

Compatibility of the Original and the Present DOS

To ensure compatibility between the present DOS system and the original, or partially updated original DOS, be aware of the following conditions:

- Disk Operating System job control information previously supplied on VOL, DLAB, and XTENT statements (VOL, TPLAB for tape) should now be supplied on the simplified DLBL (DASD label) and EXTENT (extent) statements (TLBL or tape label statement for tape). However, DOS still accepts the information in the previous form.

- A user with previously assembled tape DTF macro instructions may find that the standard label track has been modified because of error condition, multifile, or multivolume processing. To prevent modification of tape label job control information on the standard label track, the user must reassemble the tape DTF macro(s). For more information see OPTION STDLABEL in the System Control and Service publication listed in the Preface.

- User programs written for an earlier release of DOS can usually be run under the present release without change or recompilation. However, there are three main exceptions that may require recompilation or linkage editing:

  1. If higher level compilers (such as COBOL, PL/I) undergo extensive changes or modification, programs compiled under an earlier release may not be executable under a later release.

  2. If the user wishes to take advantage of a newly supported feature of DOS, recompilation of user programs is always required.

  3. If program modules undergo significant changes (again to support new features).

- A LABADDR routine that builds user-standard labels in the IOCS area of main storage cannot be executed in a storage-protected environment. Thus, it cannot be executed in a multiprogramming environment.

- Programs written for the original DOS can be run as background jobs in a multiprogramming environment with this restriction. The background program must have control of the Interval Timer feature when a program using the timer (SETIME macro instruction) is executed in a multiprogramming environment.

- Programs using PIOCS to process Direct Access Storage Device (DASD) files, such as disk or data cell, must open the file if the program is executed in a system that has DASD file protect. The channel program for the DASD must begin with a Seek (X'07') command.

- All data files created for the original DOS can be used.

- Programs checkpointed in the original DOS cannot be restarted in the present DOS.

To ensure problem program compatibility with future releases of the Disk Operating System, the following coding practices are recommended:

1. The problem program should initialize its own registers, declare a save area, and initialize register 13 to point to the save area. In particular, problem programs should not assume that register 0 contains the origin of the problem program phase, especially if the name of the phase appears in the execute statement.

2. Problem program references to file data should logically follow an OPEN(R) to the file. Whenever possible, opened files should be closed before terminating a program.

3. When exchanging information between subprograms, use the CALL, SAVE, and RETURN macros, and avoid load register and branch instructions whenever possible.

4. Restrict problem program references to the communication area to bytes 0 through 9 and 12 through 23. References outside the defined communication region or any other system tables are not to be relied upon.

5. In general, avoid macro expansions and modifications because it is the function that is supported and not the expansion. In particular, use the macro for the planned function and do not modify it, or code a substitute. For example, use DTFCD only with card devices and not with tape units to maintain device independence.

Physical IOCS vs Logical IOCS

Consider the input/output control as consisting of two parts: physical IOCS (PIOCS) and logical IOCS (LIOCS).

Physical IOCS controls the actual transfer of records between the external medium and main storage. It performs the functions of initiating the execution of channel commands and handling associated I/O interrupts. Physical IOCS consists of the following routines:

- Start I/O routine
- Interrupt routine
- Channel scheduler
- Device error routines.

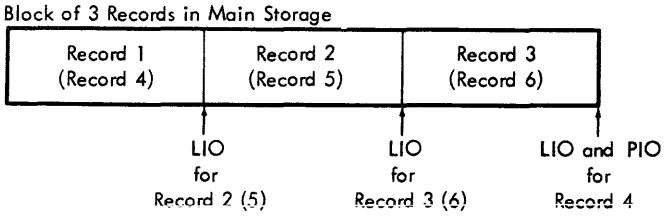
These physical IOCS routines are part of the supervisor, which is permanently located in lower main storage while problem programs are being executed. The device error routine for SYSRES resides in the supervisor area. Other error routines are called into the transient area.

Logical IOCS performs the functions a user needs to locate and access a logical record for processing. A logical record is one unit of information in a file of like units, such as one employee's record in a master payroll file, one part-number record in an inventory file, or one customer account record in an account file. One or many logical records may be included within one physical record, such as a physical tape record (gap-to-gap). The term logical IOCS refers to the routines that perform the following functions:

- Blocking and deblocking records.
- Switching between I/O areas when two areas are specified for a file.
- Handling end-of-file and end-of-volume conditions.
- Translating American National Standard Code for Information Interchange (ASCII) into Extended Binary Coded Decimal Interchange Code (EBCDIC) on input, and EBCDIC into ASCII on output.
- Checking and writing labels.

Logical IOCS uses physical IOCS to execute I/O commands whenever it determines that a transfer of data is required. For example, if a file consists of blocked records and a block has been read into main storage, logical IOCS makes each record in succession available to the user until the end of the block is reached. No physical IOCS is required. When logical IOCS determines that the last record in the block has been processed, it requests physical IOCS to start an I/O operation to transfer the next physical record (gap to gap) into main storage.

In Figure 1, only logical IOCS is required to make records 2 and 3 (and 5 and 6) available. Records 1-3 are already in main storage. Physical IOCS is also required to make record 4 available (records 4 through 6 are transferred in one block).



LIO = Logical IOCS
PIO = Physical IOCS

Figure 1. Physical IOCS vs Logical IOCS

Logical IOCS macros (such as GET, PUT, READ, WRITE, etc) and physical IOCS macros (such as EXCP and WAIT) are available to the programmer for handling records. The logical IOCS macro routines perform all the functions of both logical and physical IOCS. Thus, when a GET instruction is issued, a logical record is made available for processing. Logical IOCS routines use registers 0, 1, 14, and 15.

The physical IOCS routines are completely distinct from the routines used by logical IOCS to perform functions such as blocking and deblocking. They permit the problem program to use physical IOCS functions directly. To transfer a physical record (such as a DASD or tape record), the problem program issues an EXCP macro instruction (EXecute Channel Program). This causes the channel scheduler to handle a request for data transfer. Program execution immediately continues with the next problem program instruction. However, the DASD or tape record is not available in main storage until some later time. When the record is needed for processing, the program must test (WAIT macro instruction) to determine if the transfer has been completed. Physical IOCS uses registers 0 and 1.

Figure 2 shows the functions of physical and logical IOCS routines.

DASD File Protection

For a 24K or larger machine, logical and physical IOCS can provide DASD data file protection. Users desiring to implement DASD file protection in the future should not include any seeks, other than the initial full seek, within their DASD channel programs (see DASD Channel Programs).

DASD file protection means that the user program is logically prohibited by the DOS supervisor from reading or writing on

cylinders (disk) or strips (data cell) other than those specified in the file extent statements. The feature does not protect the file itself from being overwritten.

In order to obtain any DASD file protection, the

$$\text{DASDFP=(n,n}\left[,\left\{\begin{matrix}2311\\2314\\2321\end{matrix}\right\}\right])$$

operand in the FOPT supervisor generation macro must be specified at system generation time.

n,n indicates the range of channels that is to provide DASD file protection.

2321 indicates all disk and data cell devices are to be file protected.

2311 or 2314 indicates disk only.

DASD file protection then becomes effective when the file is opened and only if the file involved contains unexpired labels.

LIOCS and ASCII Tape Files

Tape files may be written in EBCDIC or ASCII, but DOS processes all files in EBCDIC. Logical IOCS automatically translates ASCII to EBCDIC on input and EBCDIC to ASCII on output, as required. The conversion takes place in the user's I/O area. When the user wants to update records in his I/O area, the records must then be in the EBCDIC mode. If, however, they have already been converted to ASCII, the user himself can make use of the translate tables in the supervisor. The address of the ASCII to EBCDIC table is in bytes 44-47 of the extension of the communication region for each partition. The address of the EBCDIC to ASCII table is 256 bytes higher than that of the ASCII to EBCDIC table. The address of the extension of the communication region is found in bytes 136-139 of the communication region.

Note: Some EBCDIC characters have no direct equivalent in ASCII, so logical IOCS substitutes a character (SUB) during translation. If an EBCDIC file is translated into ASCII, and then the user translates back into EBCDIC, this substitute character may not receive the expected value. See Appendix H for the correspondence between EBCDIC and ASCII.

Types of LIOCS Processing

The logical IOCS routines process records:

1. In sequential order by the Sequential Access Method (SAM),
2. In random order by the Direct Access Method (DAM), or
3. Randomly and sequentially by the Indexed Sequential Access Method (ISAM).

SAM processing applies to all files in serial I/O devices (such as card reader, tape, printer, etc), and to records on the IBM 2311 Disk Storage Drive, or IBM 2314 Direct Access Storage Facility, or IBM 2319 Disk Storage, or IBM 2321 Data Cell Drive when they are processed serially. The types of processing performed by DAM and ISAM apply only to files of Direct Access Storage Device (DASD) records.

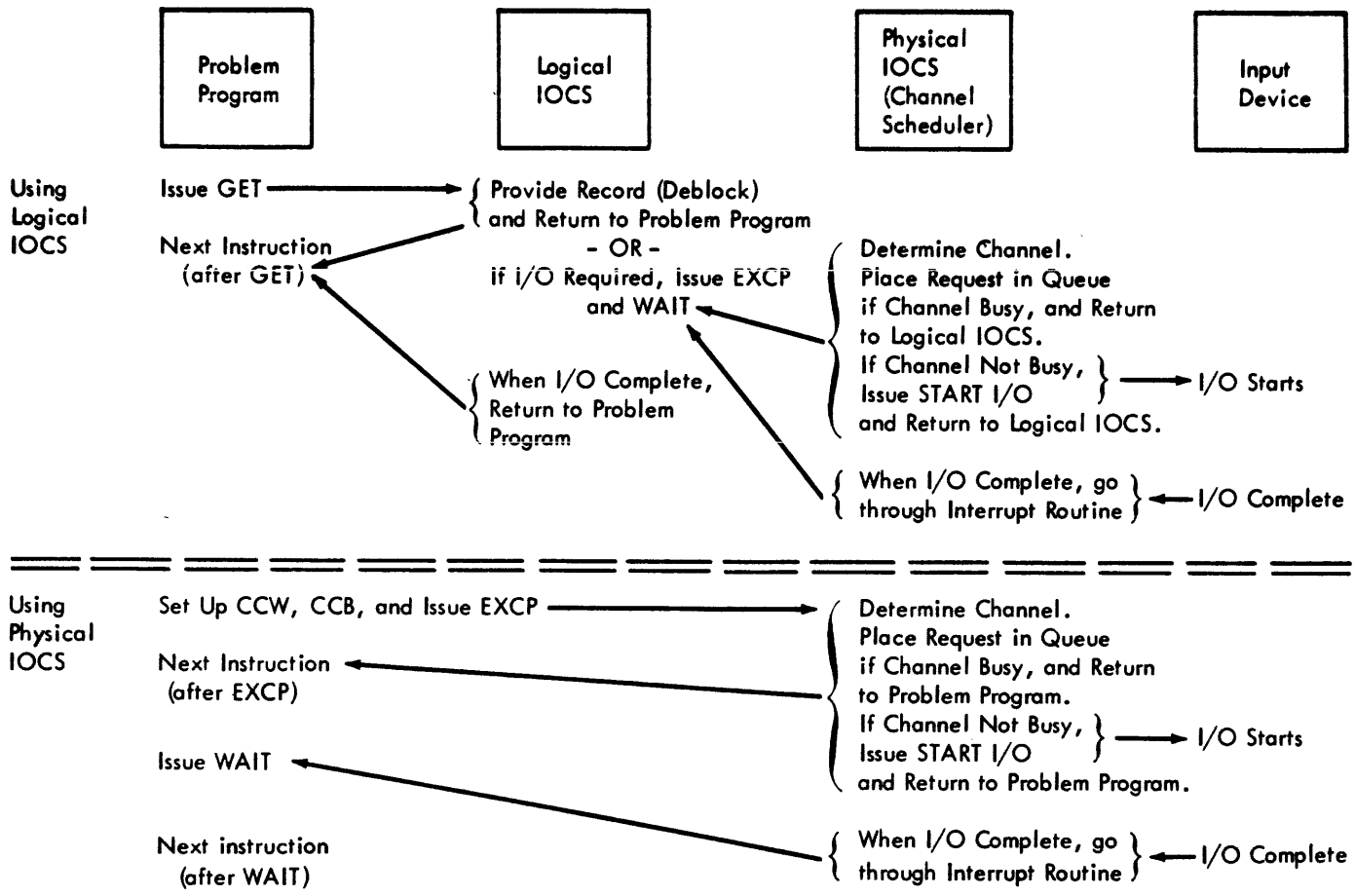


Figure 2. Retrieving a Record Using Logical IOCS (One I/O Area) or Physical IOCS

SEQUENTIAL ACCESS METHOD (SAM)

Sequential processing reads/writes and processes successive records in a logical file. For example, card records are processed in the order the cards are fed. Tape records are processed starting with the first record after a header label and continuing through the records to the trailer label. DASD records are processed starting with a beginning DASD address and continuing in order through the records on successive tracks and cylinders to the ending address.

A sequential file on DASD is contained within one or more sets of limits specified by the job control extent cards. If the logical file consists of more than one set of limits, IOCS automatically processes each set as required by the user. The records within each set must be adjacent and contained within one volume (disk pack or data cell). The sets are not required to be adjacent or on the same volume. Sequential processing of a file written on DASD by the direct access method can be performed.

The basic macros used for sequential processing are GET and PUT. These instructions overlap data transfer and processing. The extent of overlap depends on the user's I/O area assignment. In any case, when a GET or PUT has been executed, the transfer of data is complete before the instruction following the GET or PUT is executed.

DIRECT ACCESS METHOD (DAM)

The Direct Access Method (DAM) processes records contained on IBM 2311, 2314, 2319, or 2321 that are usually organized in a random manner. DAM is a method of processing records and not an organizational method.

IOCS locates a DASD record for processing by referring to a record location reference supplied by the problem

program. The location reference consists of two parts: a track reference and a record reference. The track reference specifies the track (or the first of multiple tracks) to be searched for the record. The record reference may be the record key, if records contain key areas, or the record identifier (ID) in the count area of each DASD record. IOCS seeks the specified track and searches for the record on that track, or on the succeeding tracks in the cylinder.

The basic macros for the direct access method of processing are READ and WRITE. Variations within these macros permit records to be read, written, updated, replaced, or added to a file. Thus, this method maintains a logical file in a random (or sequential) order. When a READ or WRITE instruction is executed, the actual I/O operation either starts or is placed in a queue for later execution. When the record is required for processing, the program must test (WAITF macro) to ensure that the transfer is complete.

INDEXED SEQUENTIAL ACCESS METHOD (ISAM)

DASD records contained within an indexed sequential file may be processed in a random order or in sequential order by control information. Both orders use the control information of the records (such as employee number, part number, etc), which is available in the key area of each DASD record. Any record stored at any location in the logical file can be processed using the random method. The user supplies ISAM with the key (control information) of the desired record. ISAM searches for the record and makes it available for processing. The ISMOD macro (combined with the loading, adding, and retrieval functions) is provided by the indexed sequential access method (ISAM). This is known as the Indexed Sequential File Management System (ISFMS).

In sequential processing, ISAM makes a series of records available. The records are available, one after the other in order by the control information (key) in the records. The user specifies the first record to be processed. ISAM retrieves the succeeding records (on demand) from the logical file, in key order, until the problem program terminates the operation.

ISAM makes it possible to create an organized file and then add to, read from, and update records in that file. The file is organized from records that have been presorted by their control information. As the records are loaded onto DASD, ISAM constructs indexes for the logical file. The indexes permit individual records to be found in subsequent processing operations. The indexes are created in such a way that records can be retrieved randomly or sequentially. If records are added to the file at a later date, ISAM updates the indexes to reflect the new records.

The basic macros for processing the indexed sequential files are READ/WRITE and GET/PUT. READ and WRITE are for random operations. A READ or WRITE instruction in the problem program starts the I/O operation or places it in a queue, and execution of the problem program continues. When an instruction later in the program requires that the transfer of data be complete, a test must be made. A WAITF macro is provided for the test. For sequential operation, GET and PUT macros are used. When a GET or PUT instruction for a record is executed, the transfer of data is completed before the next instruction in the problem program is executed.

Basic and Queued Telecommunications Access Methods

Disk Operating System allows communication with remote terminals by either the Basic (BTAM) or Queued (QTAM) Telecommunications Access Method. The publications listed in the Preface gives a general description of the available telecommunications facilities and specific information on the imperative macro instructions, DTFs, and modules used with BTAM and QTAM.

Macro System

A macro is a group of computer instructions. Thus, for one macro instruction, many computer instructions may be assembled.

The macro system has two basic parts:

- Macro definitions. General routines written as source statements and stored in the Assembler Sublibrary of the Source Statement Library. These routines enable the user and IBM to build source program macro instructions.
- Source-program macro instructions:
  1. Imperative input/output control macro instructions identify what I/O operation is desired. For example, in Appendix C, GET indicates that the user wants to obtain a card record.
  2. Supervisor communication macro instructions communicate with the supervisor and give access to the communication region.
  3. For processing with IOCS, two additional macro instructions are used:
     a. Declarative logic module generation macro instructions (with LIOCS) give information about the type of module to be generated. A module is an object code routine that can handle the conditions specified in the module generation macro. For example, in Appendix C, the CDMOD generates a module to handle card input on a 2540 using a work area.
     b. Declarative DTFxx macro instructions (with the LIOCS and PIOCS) define the characteristics of the specific file to be processed. The information in the macro instruction is assembled into a DTF Table. For example, in Appendix C the DTFCD macro instruction specifies that the symbolic unit containing the file is SYS004, that the file uses a work area and an I/O area called A1, and that control should be given to EOFCD when the last card is read.

A direct relationship exists between these parts. During assembly, the macro instruction specifies which macro definition is to be called. Figure 3 shows how the macro definition is extracted, tailored, and inserted into the program. This is accomplished by a selection and substitution process using the general information in the macro definition and the specific information in the macro instruction. The insertion is a module, a table, or a small in-line routine and is called the macro expansion.

After the insertion is made, the complete program consists both of source program statements and assembler language statements generated from the macro definition. In subsequent phases of the assembly, the entire program is processed to produce the machine-language program.

IBM provides a number of prewritten macro definitions and specifies the macro instructions that the programmer can use for the definitions. Other macro definitions can be written by the user. See the Assembler publication listed on the cover of this manual for additional information on this subject.

The IBM-supplied macro instructions that are explained in this publication are organized into these categories:

- Sequential Access Method (SAM) LIOCS macro instructions.
- Direct Access Method (DAM) LIOCS macro instructions.
- Indexed Sequential Access Method (ISAM) LIOCS macro instructions.
- PIOCS macro instructions.
- Supervisor Communication macro instructions.
- Additional macro instructions.

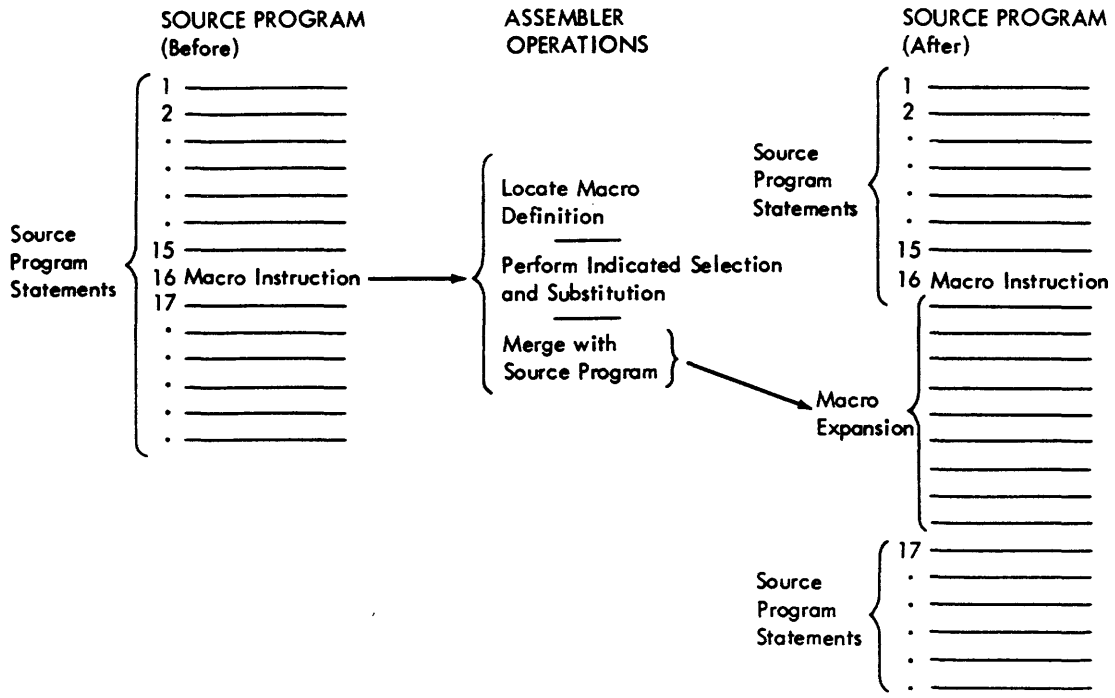


Figure 3. Schematic of Macro Processing

Self-Relocation and IOCS

To make LIOCS and PIOCS imperative and Supervisor Communication macro instructions self-relocating the user must:

1. Use the OPENR and CLOSER macro instructions, and
2. Use register notation within all his imperative macro instructions.

See the discussion on Register Notation in this publication.

DTF Declarative Macro

Whenever logical IOCS imperative macro instructions (GET, PUT, READ, WRITE, etc) are used in a program to control the input/output of records in a file, that file must be defined by a declarative macro instruction, called a DTF (Define The File). The DTF macro instruction describes the characteristics of the logical file, indicates the type of processing, and specifies main-storage areas and routines.

For example, if a GET is issued, the file definition supplies such factors as:

- Record type and length.
- Input device from which the record is to be retrieved.
- Address of the main-storage area where the record is to be located for processing by the problem program.

Device-oriented file-definition declarative macro instructions are available for defining files processed by LIOCS. DTFPH is available for magnetic tape or DASD files processed by PIOCS. Figure 4 contains an example of a DTF source statement. For LIOCS operations, the file-definition macro instructions used depend on the type of processing that will be performed for the file and in some cases, the I/O device upon which the file resides.

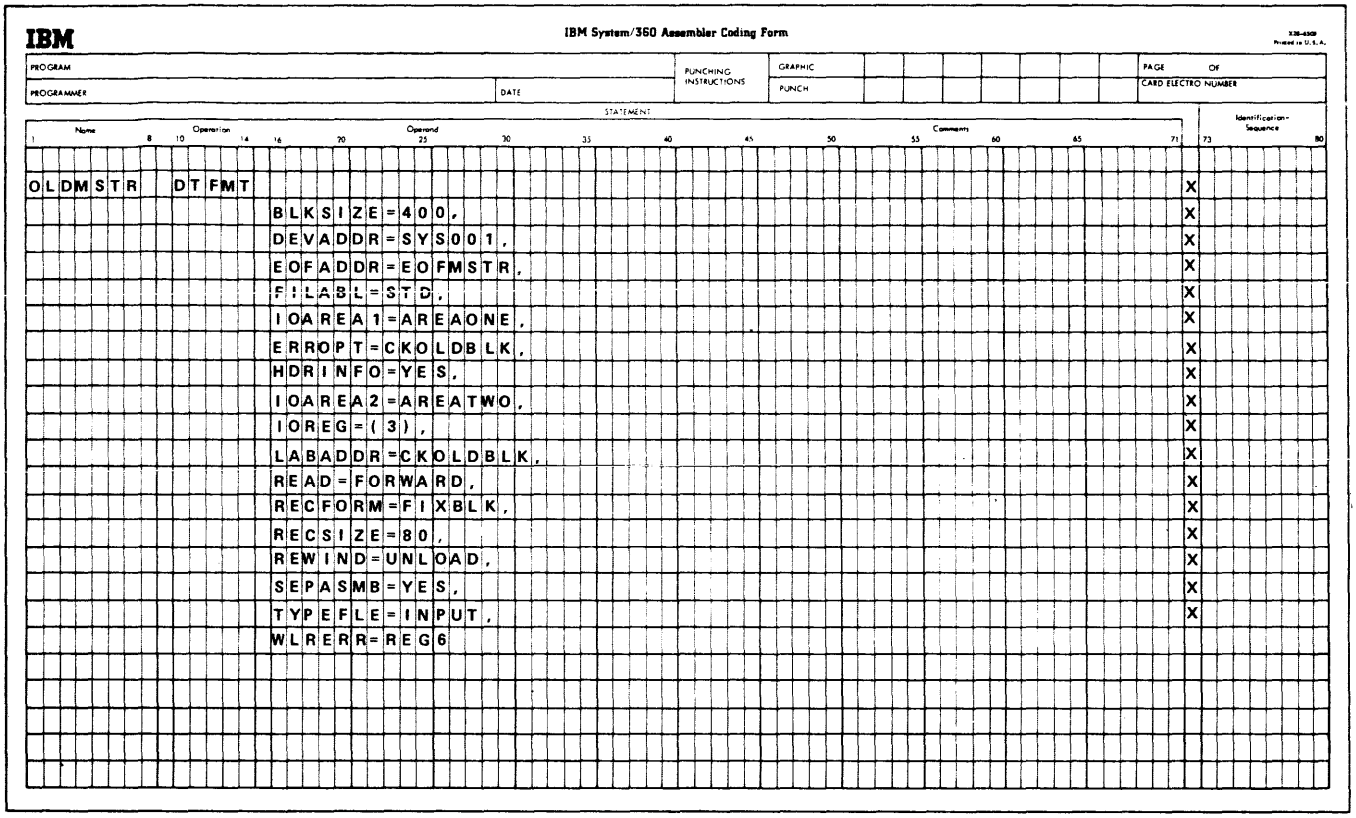


Figure 4. Sample DTFMT Macro Instruction

SEQUENTIAL PROCESSING: This applies to input/output files in serial devices or to 2311, 2314, 2319, or 2321 DASD when records are processed sequentially. The following macros are used for sequential processing:

Macro Instruction	Defines the File for
DTFCD	Card Device
DTFCN	Console (Printer-Keyboard)
DTFDI	Device Independence
DTFMR	Magnetic Reader (MICR) and Optical Reader/Sorter
DTFMT	Magnetic Tape
DTFOR	Optical Reader
DTFPR	Printer
DTFPT	Paper Tape Reader or Punch
DTFSD	Sequential DASD
DTFSR	Serial type device (for compatibility only)

DIRECT ACCESS METHOD: Whenever a logical DASD file is to be processed randomly, DTFDA should be used.

INDEXED SEQUENTIAL SYSTEM: Whenever a logical DASD file is to be organized or processed by the indexed sequential access method (ISAM), DTFIS should be used.

PHYSICAL IOCS PROCESSING: When PIOCS macro instructions (EXCP, WAIT, etc) are used for a file, the DTFPH macro instruction is required if standard labels are to be checked or written on a DASD or magnetic tape file, or if the DASD file is file protected.

A DTFxx macro instruction generates a DTF table that contains indicators and constants describing the file. The user can reference this table by using the symbol filename+constant or filenamex, where x is a letter. When such a reference is necessary, the constant or letter is specified in the text. When referencing the DTF table, the user must ensure addressability through the use of an A-type constant, or through reference to a base register.

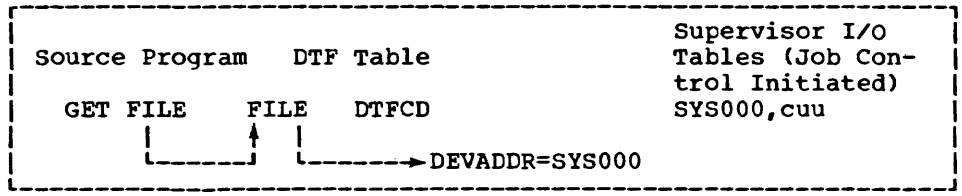


Figure 5. Relationship between Source Program, DTF Table, and Job Control I/O Assignment

SYMBOLIC UNIT ADDRESSES

In each of the DTF macro instructions, except those for DTFIS, and DTFPH MOUNTED=ALL files, the user can specify a symbolic unit name in the DEVADDR=SYSnnn operand. This symbolic unit name is also used in the job control ASSGN statement to assign an actual I/O device address to the file. For DASD files, the symbolic unit name is supplied in the job control extent statement.

The programmer chooses the symbolic unit name of a device from a fixed set of symbolic names. He writes his program considering only the device type (tape, card, etc) of his file. At execution time, the actual physical device is determined and assigned to a given symbolic unit. For instance, a programmer can write a program that processes tape records and can call the tape SYS000. At execution time the operator (using ASSGN) assigns any available tape drive to SYS000.

Figure 5 shows the relationship between the source program, the DTF table, and the job control I/O assignment.

The fixed set of symbolic names that can be used with a declarative macro for either a background or batched job foreground program are the same and are represented by SYSxxx. If a foreground program is in the single program mode (via START statement), the system logical units for foreground programs must be assigned to unit record devices and SYSRLB, SYSSLB, SYSCLB, and SYSLNK cannot be used. Use of SYSCLB and SYSLNK for foreground programs is possible only in systems which support the batch-job foreground and private core image library options. Except for the preceding, both foreground and background programs can reference the same logical unit providing different devices, or DASD extents, are assigned. These symbolic units are system logical units and programmer logical units.

System Logical Units

SYSRDR Card reader, magnetic tape unit, or disk extent primarily for job control statements.

SYSIPT Card reader, magnetic tape unit, or disk extent as the primary input unit for programs.

SYSPCH Card punch, magnetic tape unit, or disk extent as the primary unit for punched output.

SYSLST Printer, magnetic tape unit, or disk extent as the primary unit for printed output.

SYSLOG Printer-keyboard for operator messages and logging job control statements. Can also be assigned to a printer.

SYSRES System residence area on a disk drive.

SYSLNK Disk extent as input to the linkage editor.

SYSRLB Disk extent for a private relocatable library.

SYSSLB Disk extent for a private source statement library.

SYSCLB Disk extent for a private core image library.

Figure 6. Relationship between Source Program and Job Control I/O Assignment

Programmer Logical Units

SYSnnn SYSnnn represents all the other symbolic units in the system. These units vary from SYS000 to SYSmax, where SYSmax represents the highest numbered programmer logical unit available for a partition. The largest number available in the system is 222 when MPS=BJF, and 244 when MPS=YES or NO. SYSmax can be determined by the user's distribution of the programmer logical units among the partitions.

Each declarative macro, except DTFIS in which SYSnnn applies, requiring a symbolic unit to be specified has a list of symbolic units that are valid for that macro. In that list, SYSnnn represents only the programmer logical units. However, SYSxxx indicates either a system or a programmer logical unit.

Note: For the direct-access method, not only must the extent statements be supplied in ascending order, but the symbolic units for multivolume files must be assigned in consecutive order.

For files processed by either the Sequential Access Method (SAM), or the Direct Access Method (DAM), only one symbolic unit may be assigned to a file on one volume.

In physical IOCS, the symbolic unit name is specified in the CCB (as well as the DTFPH when used). Figure 6 shows the relationship between the source program and the job control I/O assignment.

Logic Module Generation Macro Instructions

Each DTF except DTFCN and DTFSR must link to an IOCS logic module. These modules provide the necessary instruction to perform the input/output functions required by the problem program. For example, the module reads or writes data, tests for unusual input/output conditions, blocks or deblocks records if necessary, or places logical records in a work area. Most imperative macro instructions enter a logic module to perform the necessary function.

Some of the module functions are provided on a selective basis, according to the parameters specified in the xxMOD macro instruction. The programmer has the option of selecting (or omitting) some of these functions according to the requirements of his program. The omission of some of these functions results in smaller main storage requirements for a particular module.

Note: If the user issues an imperative macro, such as WRITE or PUT, to a module that does not contain that function, an invalid supervisor call (SVC 50) is generated, the job is terminated, and a message is displayed.

A logical IOCS module is defined as a subset to another logic module if all the functions available to the subset module, plus additional functions, are also available to the superset module. For example:

Superset Module Functions	Subset Module Functions
Optional use of CNTRL macro	CNTRL macro cannot be used
Workarea and I/O area processing	I/O area processing only
Support of printer overflow	No printer overflow support
Read backward and forward on magnetic tape	Read forward only

The relationship between subsets and supersets is shown in diagrams at the end of the discussion for each module.

Some CDMOD, PRMOD, PTMOD, ISMOD, SDMOD, DAMOD, and MTMOD macro instructions correspond to two or more DTFCD, DTFPR, DTFPT, DTFIS, DTFSD, DTFDA, and DTFMT macro instructions. The functions required by these DTFs are thus available in a single

xxMOD macro instruction, even if the DTFs have slightly different parameters.

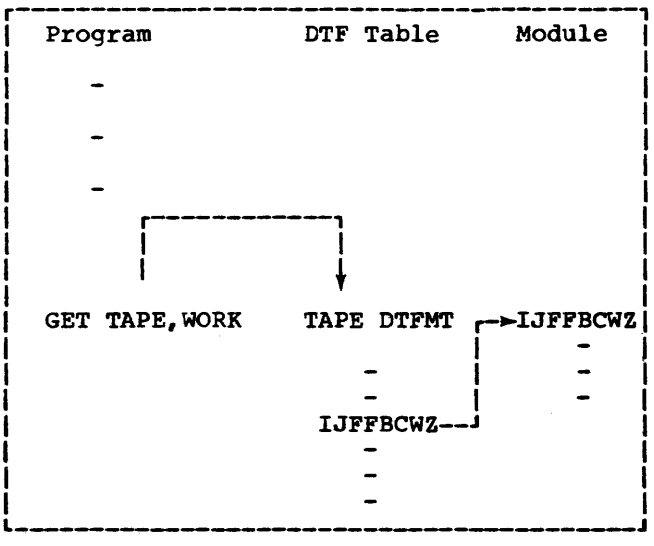
Interrelationships of the Macro Instructions

HOW THE IOCS MODULE IS LINKED WITH THE DTF TABLE

Regardless of the method of assembling logic modules and DTFs (that is, with the main program or separately), a symbolic linkage results between the DTF table and the logic module. The Linkage Editor resolves these linkages at edit time.

The IOCS module and DTF table linkage is accomplished by generating a V-type address constant in the DTF table and a named CSECT in the logic module. To resolve this linkage, the module names (or linkage symbols) must be identical.

The following example shows the relationship of the program, the DTF, and the logic module. Imperative macros initiate the action to be performed on the file by branching to the logic module entry point generated in the DTF table.



TAPE is the name of the file. IJFFBCWZ is the name of the logic module.

GENERATION OF MODULE NAMES IN DTF TABLES AND LOGIC MODULES

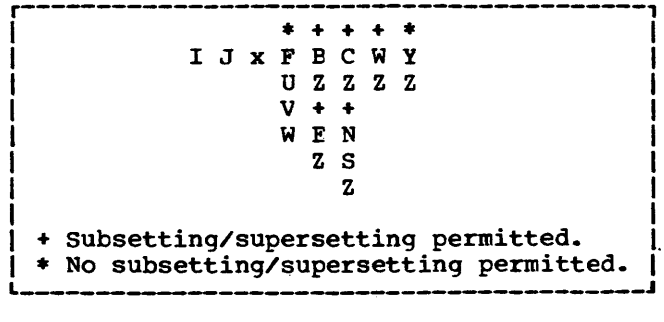
A module name is generated in the DTF table by one of two methods:

1. The user may explicitly specify the module name by entering it in the DTF operand MODNAME=name.
2. The user may allow the macro definition that processes the DTF macro instruction to generate the module name as determined by the functions required by the DTF macro instruction.

A module name is generated for a logic module in a similar manner. The generated names are referred to as standard module names. Information on standard module names follows the discussions of the logic module generation macros.

SUBSET/SUPERSET MODULE NAMES

When a DTF table is assembled (with the main program or separately), a module name is generated that exactly reflects the functions required by the DTF macro instruction. However, if similar DTFs are assembled together, the services required by the similar files are collected by the macro definition during the assembly process, and one superset module name is generated. The problem programmer can become familiar with the quick reference Subset/Superset xxMOD Names charts following the logic module description by studying this example:



The letters indicate several functions that can be performed by the logic module. A superset is a logic module that performs all the functions of its subset module. For example, the logic module name IJxWENZZ is a superset module for subset modules IJxWESZZ and IJxWEZZZ. Similarly, IJxWESZZ is a superset module for subset module IJxWEZZZ.

An asterisk (*) over a column indicates that no subsetting or supersetting is permitted, whereas a plus (+) sign in a column indicates that both are permissible. Two plus signs in a single column divide that column into mutually exclusive sets. In this example, C is not a superset of N, S, or Z, and conversely N, S, or Z, is not a subset of C.

The vertical arrangement of letters within a column is always in alphabetical order by group. The only significance of the horizontal order is the letters themselves that comprise the module name.

Do not use subset/superset charts to decide functions to be performed. The requirements of the problem program should decide these functions. Then, the logic module name is formed according to the Recommended Module Name for xxMOD section. These charts also make it possible to obtain a compromise between the number of functions a module can perform and the resulting size of the module.

EDITING LOGICAL IOCS PROGRAMS

The programmer has the option of either assembling DTFs and logic modules with his main program or assembling them separately for later linkage editing with the main program. To take full advantage of the linkage editing facilities for DTF tables and logic modules, the operand SEPASMB=YES should be specified when DTF tables or logic modules are separately assembled.

Logical IOCS programs always generate symbolic linkages between DTF tables and logic modules. These linkages are resolved by the Linkage Editor at edit time. Furthermore, if DTF tables are assembled separately, the programmer must define any additional symbolic linkages in the form of EXTRN-ENTRY symbols. Appendix C contains a full description of the different symbolic linkages that must be defined when separately assembled programs are edited.

When the operand SEPASMB=YES is specified in a DTF macro instruction, a CATALR card with the file name supplied to the DTF is generated ahead of the object deck. When the operand SEPASMB=YES is specified in an xxMOD macro instruction, a CATALR card with the module name is generated ahead of the object deck. In either case, a START card must not be used in a separate assembly.

Cataloging DTF tables and logic modules to the relocatable library is recommended to reduce the user coding effort and to minimize total time needed to prepare and test programs that use logical IOCS. Using DTF tables cataloged in the relocatable library requires standardization of the labels referred to by the DTFs. This is necessary if these tables are used by different programs.

If the I/O modules, DTF tables, and the main program are assembled together, the linkage editor searches the input stream and resolves the symbolic linkages between tables and I/O modules. This is accomplished by external-reference information (V-type address constants generated in DTF tables) and the section-definition information (CSECT definitions in logic modules). Further information may be found in the Linkage Editor section, under Structure of a Program, in the System Control and Service publication listed in the Preface.

If any of the elements that constitute a program are assembled separately, the different object modules (assemblies) may be supplied to the input stream at linkage-edit time. The linkage editor then resolves the symbolic linkages between them. If any of the separately assembled elements have been cataloged to the relocatable library, the linkage editor finds unresolved external references in the input stream and performs the AUTOLINK function of the linkage editor. The relocatable library then searches for a relocatable module name identical to the external reference. If the module is not defined in the relocatable library, the external references to this name remain unresolved. Therefore, if the modules are assembled separately and cataloged to the relocatable library, the programmer must determine that at least one of the DTFs in his program includes a module name that can be successfully autolinked from the relocatable library. Programmer control of the module to be autolinked from the relocatable library is achieved by the MODNAME operand in the DTF macro instruction. This operand overrides the standard module name generated by the macro definition.

LINKAGE-EDITING PREASSEMBLED LOGIC MODULES

A small number of IOCS logic modules can serve a large number of DTF macros. (This applies only to CDMOD, PRMOD, SDMOD, ISMOD, DAMOD, PTMOD, and MTMOD.) For example, the following module can serve 64 different DTFMT files by using these options: TYPEFLE=INPUT or OUTPUT, RECFORM=FIXUNB or FIXBLK, WORKA=YES or NO, IOAREA2=Name or (not used), CKPTREC=YES or NO, and

READ=FORWARD or BACK. The same module can also serve files with varying block sizes, record sizes, I/O area addresses, and exit addresses.

```
                                  Col 72
MTMOD                              X
      TYPEFLE=INPUT,               X
      RECFORM=FIXUNB,              X
      WORKA=YES,                   X
      CKPTREC=YES,                 X
      READ=BACK
```

A preassembled logic module may be furnished to the linkage editor in three ways:

1. INCLUDE the module from SYSIPT.

2. INCLUDE the module from the system relocatable library.

3. AUTOLINK the module from the system relocatable library.

If a module is included from SYSIPT, its name offers no problem. However, the user assumes responsibility for a module name and functional match to the DTFs in his program.

If a module is included from the system relocatable library, the situation is similar. The user should verify that the desired modules have already been cataloged to the library by consulting a DSERV listing of the library.

If the module is to be autolinked from the system relocatable library, the user must determine whether the module name generated by the DTF (or furnished by the MODNAME parameter) coincides exactly with the name in the system relocatable library. If the names are identical, the autolink can be accomplished. Otherwise, the user must either include some module that meets these needs (from SYSIPT or from the system relocatable library), or consider the logic module needs of other DTFs in his program.

If a needed module is not available in the relocatable library, the user should determine if any other DTF will need a module that is named in the library and furnishes at least the functions required by the first DTF. For example, the following DTFs generate a request for the module named IJFFZZZZ.

DTF Example 1

```
                                  Col 72
FILE1 DTFMT                        X
            TYPEFLE=INPUT,         X
            RECFORM=FIXUNB,        X
            IOAREA1=A1,            X
            IOAREA2=A2
```

DTF Example 2

```
                                  Col 72
FILE2 DTFMT                        X
            TYPEFLE=OUTPUT,        X
            RECFORM=FIXBLK,        X
            IOAREA1=A3,            X
            WORKA=YES
```

If the module named IJFFZZWZ is available in the system relocatable library and the two files are defined in the same assembly, the autolink facility can be used. However, if only the first file is defined in the assembly and no IJFxxxxx modules are cataloged in the system relocatable library, the user must either furnish a private copy of the IJFFZZZZ module at linkage-edit time or include the larger module (IJFFZZWZ). In systems with ample main storage and/or for small programs, the user may choose to sacrifice a modest amount of main storage to achieve this capability. If the name of a DTF differs from that of a superset module, entry points (in addition to the CSECT) are generated within the logic module. This condition could arise if the DTF does not use all the functions provided for in the module.

The entry points in the module define all the subset module names that can be handled by the superset module. V-type address constants may then be resolved against the entry points if they do not match the CSECT name. For example, the module named IJFFZZWZ has a secondary entry point named IJFFZZZZ. This explains why autolink works in the previous examples. However, autolink can only be used with catalog names (which correspond to the CSECT name) but not with secondary entry points.

If the programmer gives an explicit module name to the xxMOD macro instruction, this name overrides the standard module name in the CSECT definition and no ENTRY points are generated. The DTF tables that access the module may employ the operand MODNAME=name to link to a previously named logic module.

Each installation should generate a certain set of IOCS logic modules when the system residence volume is built. These modules should reflect equipment configuration, installation standards for record formats and exits, etc. Any user requiring a special tailored module can generate it by using the logic module macro instruction and its specific parameters.

Macro-Instruction Format

Macro instructions have the same format as assembler statements. That is, each macro instruction can consist of a name, operation, and operand field.

The name field in a macro instruction may contain a symbolic name. Some macro instructions require a name; for example, CCB, TECB, DTFxx.

The operation field must contain the mnemonic operation code of the macro instruction.

The operands in the operand field must be written in either positional, keyword, or mixed formats.

Positional Operands: In this format, the parameter values must be in the exact order shown in the individual macro discussion. Each operand, except the last, must be followed by a comma with no imbedded blanks. If an operand is to be omitted in the macro instruction and following operands are included, a comma must be inserted to indicate the omission. No commas need to be included after the last operand. Column 72 must contain a continuation punch if the operands fill the operand field and overflow into another card. Any nonblank character in column 72 causes the next parameter to be read in the following card. For example, GET uses the positional format. A GET for a file named CDFILE using WORK as a work area is punched:

```
GET  CDFILE,WORK
```

Keyword Operands: The exact parameters used are equated to a keyword value. Thus, an operand written in keyword format has the form:

```
LABADDR=MYLABELS
```

where LABADDR is the keyword and MYLABELS is the parameter and LABADDR=MYLABELS is the complete operand. The operands in the macro instruction may appear in any order, and any that are not required may be omitted. Different keyword operands may be punched in the same card, each followed by a comma. Or, they may be punched in separate cards as in Figure 4.

Mixed Format: The operand list contains both positional and keyword operands. The keyword operands can be written in any order, but they must be written to the right of any positional operands in the macro instruction.

For additional information on coding macro statements, see the Disk and Tape Operating Systems Assembler Language publication.

OPERAND CARDS FOR DECLARATIVE MACROS

The operands of the DTFxx and the module generation macro instructions can be punched in a set of entry cards in the assembler format. Figure 4 shows an example of the entry cards used for a DTFMT macro instruction. The macros may be assembled in any order.

The first entry card is a header card, and the continuation cards are detail cards. The header card is punched with:

- The symbolic name of the file in the name field. Programming Note: Avoid defining symbols beginning with IJ since they may conflict with IOCS symbols beginning with IJ. Avoid symbols that are identical to a filename plus a single character suffix. Thus, for the filename RECIN, IOCS generates the symbols RECINS, RECINL, etc by concatenating the filename with an additional character.

  In a DTF, the symbolic filename may be up to seven characters long. This filename, if it is required on any of the standard label job control statements, must be the same as that used in the DTF header card.

  For a module generation macro, the name may or may not be specified. See Generation of Module Names in DTF Tables and Logic Modules.

- The macro instruction mnemonic operation code in the operation field.

- Keyword operands in the operand field, if desired.

- A continuation punch in column 72, if detail cards are necessary.

The detail cards follow the header card, and may be arranged in any order. Each

detail card is punched, beginning in column 16, with one or more keyword operands separated by commas. All detail cards except the final one must be punched with a comma immediately following the last operand and with a continuation punch in column 72. Comments may be included if a space is left after the comma following the last operand.

NOTATION CONVENTIONS

The following conventions are used in this publication to illustrate macro instructions:

1. Uppercase letters and punctuation marks (except as described in these conventions) represent information that must be coded exactly as shown.
2. Lowercase letters and terms represent information that must be supplied by the programmer. More specifically, an n indicates a decimal number, an r indicates a decimal register number, and an x indicates an alphameric character.
3. Information contained within brackets [] represents an optional parameter that can be included or omitted, depending on the requirements of the program.
4. An ellipsis (a series of three periods enclosed by commas) indicates that a variable number of items may be included.
5. Options contained within braces {} represent alternatives, one of which must be chosen. When the alternatives appear in a string, they are separated by a vertical bar (logical OR).
6. [name / label / address] — A name-field symbol in this assembly, or an operand of an EXTRN statement, or * (the location counter).
7. filename — Symbol appearing in the name field of a DTF macro instruction.
8. {n / (r)} — Self-defining value, such as 3, X'04', (15), B'010'.
9. length — Absolute expression, as defined in the Assembler publication.
10. {A / B / C} — Underlined elements represent an assumed value in the event a parameter is omitted.
11. {name / (r)} — Ordinary register notation.
12. {name / (0) / name / (1)} — Special register notation (ordinary register notation can be used.)

Register Notation

Certain operands can be specified in either of two ways:

1. The user can specify the operand directly.
2. The user can preload address of the value into a register before executing the macro instruction. When using register notation, the register should contain only the specific address and high order bits should be set to 0.

In the latter case, the user must specify the register in the macro instruction. (The registers that can be used for this purpose are discussed under Register Usage.) This method is known as ordinary register notation. When the macro instruction is assembled, instructions are generated to pass the information specified in the operand to IOCS or to the supervisor. For example, if an operand is written as (8), and if the corresponding parameter is to be passed to the supervisor in register 0, the macro expansion contains the instruction LR 0,8.

The user can conserve main storage and save execution time by using what is known as special register notation. In this method, the operand is expressed as either (0) or (1). This notation is special for two reasons:

- The register notation designation of registers 0 and 1 is not allowed unless specifically designated.
- The designation must be made by the specific three characters (0) or (1). When special register notation is indicated by (0) or (1) in a macro instruction, the user can use ordinary register notation and the macro expansion will contain the extra (LR) instruction.

The format description for each macro instruction shows whether special register

notation can be used, and for which operands. For example,

```
GET    {filename}  [,{workname}]
       {  (1)   }   {  (0)   }
```

The format description shows that the filename operand can be written as (1), and the work name operand as (0). If either of these special register notations is used, the user's problem program must load the designated parameter register before execution of the macro expansion. Ordinary register notation can also be used.

Register Usage

Registers for Special Use

General registers 0, 1, 13, 14, and 15 have special uses, and are available to the problem program only under certain conditions.

The following paragraphs describe the general uses of registers 0, 1, 13, 14, and 15 by IOCS, but the description is not meant to be all inclusive. For more information on problem program subroutine linkage through registers, refer tc the Linkage Registers section. In addition, special applications, such as a MICR user stacker selection routine, may require different registers.

Registers 0 and 1: Logical IOCS macros, the supervisor macros, and other IBM-supplied macros use these registers to pass parameters. Therefore, these registers may be used without restriction only for immediate computations, where the contents of the register are no longer needed after the computation. However, if the programmer uses them, he must either save their contents himself (and reload them later) or finish with them before IOCS uses these registers.

Register 13: Control program subroutines, including logical IOCS, use this register as a pointer to a 72-byte, doubleword aligned save area. When using the CALL, SAVE, or RETURN macros, the problem programmer can set the address of the save area at the beginning of each program phase, and leave it unchanged thereafter. However, when sharing a reentrant (read only) logic module among tasks, each time that module is entered by another task, register 13 must contain the address of another 72-byte save area to be used by that logic module.

Registers 14 and 15: Logical IOCS uses these two registers for linkage. Register 14 contains the return address (to the problem program) from DTF routines, called programs, and user's subroutines. Register 15 contains the entry point into these routines, and is used as a base register by the OPEN, CLOSE, and certain DTF macros. IOCS does not save the contents of these registers before using them. If the programmer uses these registers, he must either save their contents himself (and reload them later) or finish with them before IOCS uses them.

Registers for Programmer Use

Registers 2-12 are available for general usage by programmers. Registers 0, 1, 13-15 are used by the operating system.

> Note: If these registers are used by programmers, their contents should be saved before use and restored before returning control to the system.

The assembler instruction for translate and test (TRT) makes special use of register 2. It is the programmer's responsibility to save the contents of this register before executing the TRT instruction if register 2 contains valuable information (such as pointers or counters) for later use in his program. After TRT instruction has been executed, the programmer can then restore the contents of register 2 from the save area.

Registers 2-12: If an ISMOD logic module precedes an assembler language USING statement or follows the problem program, registers 2-12 remain unrestricted even at assembly time. However, if the ISMOD logic module(s) lies within the problem program, the problem programmer should issue the same USING statement [which was issued before the logic module(s)] directly following the logic module(s). This action is necessary because the ISMOD logic module uses registers 1, 2, and 3 as base registers, and the ISMOD CORDATA logic module uses registers 1, 2, 3, and 5 as base registers. Each time either module is assembled, these registers are dropped.

Label processing is currently a function of:

- DASD
- 2400-series magnetic tape
- 3420 magnetic tape

This section discusses the IOCS portion of that function. Appendix A gives a complete discussion of the actual physical labels.

Because of the direct access capabilities of DASD devices, standard labels are required for logically accessing files by name and for adequate file protection. Tape files do not require file labels, but without them maximum file security and control cannot be attained. Also, tape file security can only be maintained when the proper density is used both on the ASSGN statements and on the tape drive.

Other files generally are not labeled, but processing for the file is similar. The file is opened to make it available for processing by logical IOCS routines. When IOCS detects an end-of-file on input, it branches to the EOFADDR address where the user can close the file. On output, he can close it at his discretion.

DASD Standard Labels

Labels are required when processing DASD files. Thus, the user must supply both a DASD label (DLBL) statement for each logical file to be processed, and an extent (EXTENT) statement area on a DASD device. The OPEN macro uses the information supplied in these cards and also certain information from the DTF table for the file. Also, when processing standard labels, a LBLTYP statement may be required to define storage for nonsequential DASD file labels (see the System Control and Service publication).

For input, the extent(s) for a file must either coincide with, or be within, an existing extent(s) that is defined in the Volume Table of Contents (VTOC). That is, on input, IOCS opens only an existing file or a subset of an existing file. For output, the file to be written cannot overlap existing unexpired files. IOCS does not destroy an unexpired file, except when an internal system file (IJSYS) overlays an identical system file, without an explicit request from the user. However, if OPEN determines that the output file will overlay an existing file whose expiration date has expired, OPEN deletes the expired label(s) from the VTOC. This, in effect, removes the file from the volume. In a multivolume file, the file may be removed from all the volumes that it occupies or from only some of the volumes.

If OPEN determines that the expiration date of an existing file to be overlaid by the output file has not expired, the old file cannot be destroyed automatically. The user has the following choices.

- For sequential or physical IOCS processing:
  1. Terminate the job.
  2. Bypass the extent. That extent and any remaining extents for that file are bypassed and the job is terminated.
  3. Delete the unexpired file.
- For work file and direct access processing:
  1. Terminate the job.
  2. Bypass the extent. That extent and any remaining extents for that file are bypassed and the job is terminated.
  3. Delete the unexpired file.
- For index sequential processing:
  1. Terminate the job.
  2. Delete the unexpired file.

For more information on processing labels, see the OPEN macro under the appropriate access method.

During processing, IOCS recognizes an end-of-volume condition when the extents on one volume have been processed and an extent for another volume is encountered. When this condition occurs, IOCS branches to the user's LABADDR routine (if provided)

to write or pass individually each user standard trailer label to be processed. After all user standard trailer labels are processed (if any), IOCS processes the standard labels on the next volume and branches to the user's routine to process user standard header labels. After the header labels are processed, IOCS continues to process the data.

When all records for an output logical file have been written, the CLOSE instruction must be issued to perform normal end-of-file procedures. IOCS then branches to the user's LABADDR routine (if provided) to write user trailer labels and the file is made inactive. If the end of the last extent specified for the file is reached before the CLOSE instruction is issued, IOCS assumes an error condition.

For input logical file, IOCS determines an end-of-file condition by either the ending address of the last extent specified for the file in the extent statement, or by an end-of-file record read from the data file. For sequential processing with DTFSD or DTFSR, IOCS branches to the EOFADDR routine upon an end-of-file condition. For sequential processing with DTFIS, IOCS posts the end-of-file condition in the field referred to as filenameC. The user can then test this bit and take action necessary to close his file. However, when processing in random order, the user must determine the end-of-file by checking filenameC (DTFIS) or ERRBYTE (DTFDA).

If further processing of a closed file is required at some later time in the program, the file must be reopened. When a file is processed in sequential order, IOCS checks the label(s) on the first volume and makes the first extent available, the same as at the original OPEN. When a file is processed by physical IOCS with the DTFPH operand MOUNTED=SINGLE, IOCS opens the next extent specified by the user's extent statement. When a file is processed by the direct access method (DTFDA), by the indexed sequential system (DTFIS), or by physical IOCS with MOUNTED=ALL on the DTFPH, all label processing is repeated and all extents are again made available.

If user standard labels are desired, the user must supply a LABADDR routine, unless processing with physical IOCS. The direct and sequential access methods process both user header and trailer standard labels. The indexed sequential access method does not process user standard labels. Also, user labels cannot be created for a file whose first extent is a split cylinder extent. The direct access method writes a user trailer label only on the first volume of a multivolume file.

When the LABADDR routine is entered, IOCS loads an alphabetic O, V, or F in the low-order byte of register 0. O indicates header labels, V indicates trailer end-of-volume labels, and F indicates end-of-file labels. The user's LABADDR routine can test this character to determine the labels to be processed. IOCS also loads the address of an 80-byte IOCS label area in register 1.

Within the LABADDR routine, the user cannot issue a macro that calls a transient routine (such as OPEN, CLOSE, DUMP, PDUMP, CANCEL, and CHKPT). For multivolume files, the LABADDR routine should save registers 14 and 15 upon entry, and restore them before issuing the LBRET macro.

Writing User Standard Labels on Disk

When the user specifies LABADDR (see DASD Standard Labels), OPEN reserves the first track of the first data extent as a user label area. When LABADDR is specified, at least one user header and trailer label must be written if the access method processes it. For the direct access method, when TRLBL=YES is specified with LABADDR, trailer labels are processed.

IOCS uses bytes 1-4 of the 80-byte label for the label identification (for example: UxLy, where x = H or T and y = 1, 2, ..., 8). The user can use the other 76 bytes as he wishes. The maximum number of header or trailer labels is eight for a 2311 or 2314 file, and five of each for a 2321 file. IOCS stores the label information (UHLx or UTLx) that it generates in bytes 1-4 of the IOCS label area. The user can test this information, in addition to registers 0 and 1, to determine the type and number of the label. (See User-Standard DASD File Labels for the label formats.)

Labels are built in either of the following ways:

1. Build an 80-byte (or a 76-byte) label in the user area of main storage, and load the address of the label area (label area minus four, if 76-byte) into register 0 before issuing the LBRET macro. When the label is moved into the IOCS area, IOCS adds four bytes to the address in register 0.

2. Build a 76-byte label in the IOCS area at the address (that IOCS supplies in register 1) plus four, and load the

contents of register 1 to register 0 before issuing the LBRET macro.

The IOCS area of main storage is a part of the supervisor. If the program is executed on a system with the storage protection feature, method 1 must be used because the user cannot write into the supervisor area. Thus, no user standard label routine using the second method can be executed in a multiprogramming environment.

When the label is ready to be written, the LBRET macro returns control to IOCS. If LBRET 2 is used, OPEN writes the label and returns control to the user's label routine unless the maximum number of labels has been written. If LBRET 1 is used, the label set is terminated and no more labels can be created.

When IOCS receives control, the IOCS routines move the label from the address the user loaded in register 0 into the IOCS label area. If the maximum number of labels has not been written, IOCS increases the identification number by 1 and returns to the user's label routine unless LBRET 1 was used. If the maximum number of labels has been created, IOCS automatically terminates the label set.

Checking User Standard Labels on Disk

When a DASD file contains user standard trailer and/or header labels, IOCS makes these labels available one at a time to the user if LABADDR is specified in the DTF for the file (see DASD Standard Labels). If the labels are to be checked against information obtained from another input file, that file must be opened ahead of the DASD file.

When the problem program has finished checking a label, it can update it or leave it unmodified. If it is to be updated, the problem program simply updates the appropriate label fields and issues the LBRET 3 macro. This causes the OPEN routine to update (rewrite) that label and read the next label. Register 1 points to the label (outside the problem program area). In a multiprogramming environment, the problem program must move the label to an area within the problem program before modifying it. After the label is modified, the problem program must initialize register 0 with the address of the modified label before issuing the LBRET 3 macro. If the label is to remain unmodified, the problem program can issue a LBRET 2 macro so OPEN will read the next label. In either situation, if the end-of-file record is encountered at the end of the labels, OPEN automatically terminates the label checking.

If the user wishes to end label checking before all the labels have been read, the LBRET 1 macro may be issued (Direct Access and Sequential Disk files only).

Tape Labels

TAPE OUTPUT FILES

For a magnetic tape output file, OPEN, CLOSE, and an end-of-volume condition rewinds the tape as specified in the DTFSR or DTFMT REWIND operand. No rewind is defined for the DTFPH, and tape file positioning depends on the labels to be processed and is the user's responsibility.

If a user writes any labels, a LABADDR routine must be supplied. (For ASCII tape files, the LABADDR routine may only be used to process user standard labels.) The user's LABADDR routine, specified in the DTF, cannot issue a macro that calls a transient routine. For example OPEN, CLOSE, DUMP, PDUMP, CANCEL, and CHKPT cannot be issued. Also when processing multivolume files, the user's label routine must save and restore register 15 if any logical IOCS macros other than LBRET are used. When user standard labels are written they always follow the standard labels on the tape.

When all records for a file are processed, CLOSE can be issued to execute the end-of-file (EOF) routines. These routines write any record or blocks of records that are not already written. A partially filled record block is truncated; that is, a short block is written on the tape. Following the last record, IOCS writes a tapemark, the trailer labels, and two tapemarks, and executes the rewind option. If no trailer labels are written, two tapemarks are written and the rewind option is executed. In either case, if no rewind is specified and no user positioning occurs, the tape is positioned between the two tapemarks at the end of the file.

If an end-of-volume (EOV) reflective marker is sensed on an output tape before CLOSE is issued, logical IOCS prepares for closing the file by ensuring that all records are written on the tape. If the programmer issues another PUT, indicating that more records are to be written on this output file, EOV procedures are initiated.

If the programmer issues a CLOSE, the EOF procedures are initiated.

Under certain conditions, an unfilled block of records may be written at an EOV or EOF condition, even though the file is defined as having fixed-length blocked records. When this file is used for input, logical IOCS recognizes and processes these short blocks. The user need not be concerned or aware of the condition.

Label processing for the EOV condition resembles that for the EOF condition, except that a standard label is coded EOV instead of EOF. Also, only one tapemark is written after the label set or after the data for unlabeled files. In an ASCII file, two tapemarks follow the end of volume labels. When IOCS detects the EOV condition, it switches to an alternate unit as designated in a job control ASSGN statement. If an alternate drive is not specified, the operator is requested to mount a new volume (on the same drive) or cancel the job. When the operator mounts the volume, IOCS checks the standard header labels and processing continues.

In some cases, you may need to force an end-of-volume condition at a point other than the reflective marker. You may want to discontinue writing the records on the present volume and continue on another volume. This may be necessary because of some major change in category of records or in processing requirements. The FEOV (forced EOV) macro is available for this function. See FEOV Macro.

Writing Standard Labels on Tape (OUTPUT)

When standard labels are written (DTFMT or DTFSR FILABL=STD or DTFPH TYPFLE=OUTPUT), the user must supply the TLBL statement for standard label information. Also, when standard labels are processed, a LBLTYP statement may be required to define storage for tape label files (see the System Control and Service publication).

When OPEN is issued and the tape is positioned at load point, the volume (VOL1) label is checked. Whether at load point or not, the old file header, if present, is read and checked to make sure that the file on the tape is no longer active and may be destroyed. If the file is inactive or if a tapemark was read, the tape is backspaced and the new file header (HDR1) label is written with the information the user supplies in the tape label statement. The volume label is not rewritten, altered, or updated.

A comparison is made between the user specified density (800 or 1600 bpi) and the VOL1 density of the expired tape. If a discrepancy is found and the tape is at load point, the volume label(s) is rewritten according to the user specified density.

If an output file begins in the middle of a reel, it is the user's responsibility to properly position the tape immediately past the tapemark for the preceding file before issuing the OPEN macro. The MTC command can be used to do this. If the tape is improperly positioned, IOCS assumes an error condition and issues a message to the operator.

If user standard labels are written, the LABADDR operand must be specified in the DTF (see Tape Output Files). After writing the standard label (header or trailer), IOCS loads register 0 (low-order byte) as follows:

O indicates header labels.
V indicates end-of-volume labels.
F indicates end-of-file labels.

The user's LABADDR routine can test this character to determine what labels should be written. IOCS also loads the address of an 80-byte IOCS label area in register 1.

> Note: For ASCII files, the user processes his standard labels in EBCDIC.

A maximum of eight user standard header (UHL), or trailer (UTL) labels can be written following the standard header (HDR1), or trailer (EOV1 or EOF1) labels. The user standard labels are 80 bytes long and are built entirely by the user. Bytes 1-4 must contain the label identification (UxLy, where x=H or T and y=1, 2, ..., 8); the other 76 bytes can be used as desired.

For ASCII tape files, you can have any number of user standard header or trailer labels. To comply with the standards for an ASCII file, these labels are identified by UHLa and UTLa, where a represents an ASCII character in the range 2/0 through 5/14, excluding 2/7 (quote). The remaining 76 bytes can be used as desired. It is the user's responsibility to ensure that labels contain UHLa and UTLa in the first four bytes.

> Note: When creating user header and trailer labels for 7-track tapes, only

unpacked data is valid in the 76-byte data portion of the label.

The user can build his labels in either of the following ways:

1. Build the label in the user area of main storage, and load the address of the label into register 0 before issuing the LBRET macro.

2. Build the label in the IOCS area at the address that IOCS supplies in register 1, and load the address of the area from register 1 to register 0 before issuing the LBRET macro.

The IOCS area of main storage is part of the supervisor. If the program is to be executed on a system with the storage protection feature, method 1 must be used because the user cannot write into the supervisor area. Thus, no user standard label routine using the second method can be executed in a multiprogramming environment.

When the label is ready to be written, the user issues the LBRET macro, which returns control to IOCS. If LBRET 2 is used, IOCS writes the label and returns control to the user's label routine. If LBRET 1 is used, the label set is terminated and no more labels can be created. When IOCS receives control, IOCS writes the label on the magnetic tape and either returns control to the user (LBRET 2) or writes a tapemark (LBRET 1).

When a standard trailer label is written, IOCS accumulates the block count for the label when logical IOCS is used. However, if physical IOCS (DTFPH) is used, the user's program must accumulate the block count, if desired, and supply it to IOCS for inclusion into the standard trailer label. For this, the count (in binary form) must be moved to the 4-byte field within the DTF table named filenameB. For example, if the filename specified in the DTFPH header name is DELTOUT, the block count field is addressed by DELTOUTB.

If checkpoint records are interspersed with data records on an output tape, the block count accumulated by logical IOCS does not include a count of the checkpoint records. Only data records are counted. Similarly, if physical IOCS is used the user's program must omit checkpoint records and count data records only.

After all trailer labels (including user labels, if any) are written at end-of-volume or end-of-file, IOCS initiates the EOF or EOV routines (see Tape Output Files).

Writing Nonstandard Labels on Tape (OUTPUT)

Note: Nonstandard labels are not permitted with ASCII.

To write nonstandard labels, the user must specify FILABL=NSTD and LABADDR=name. When the file is opened, the tape must be positioned to the first label that the user wishes to process. The MTC job control command can be used to skip the necessary number of tapemarks or records to position the file. He must also write his own channel program and use physical IOCS macros to transfer the labels from main storage onto tape (see Appendix D).

When a file is opened or closed, or when a volume is finished, IOCS supplies the hexadecimal representation (in the two low-order bytes of register 1) of the symbolic unit currently in use. See the command control block bytes 6 and 7 for these values. IOCS also loads register 0 (low-order byte) as follows:

O indicates header labels.
V indicates end-of-volume labels.
F indicates end-of-file labels.

The user's LABADDR routine can then test this character to determine the type of labels to be written.

In the user's LABADDR routine, physical IOCS macros must be used to transfer labels from main storage onto tape. For each label record, a command control block (CCB) and channel command word(s) (CCW) must be established. Also, the EXCP macro must be issued for each label record. (See Physical IOCS.) Other logical IOCS macros can be used for any processing other than the transfer of the labels from main storage to tape. Additional LABADDR routine restrictions are discussed in the Tape Output Files section.

After all labels are written, the user returns control to IOCS by use of the LBRET 2 macro. For IOCS processing after LBRET is executed, see Tape Output Files.

Writing Unlabeled Files on Tape (OUTPUT)

For program efficiency, the user with unlabeled files should specify FILABL=NO and omit TPMARK=NO in the DTF. His file is positioned properly with the MTC job

control command, if necessary, and writing begins immediately. Other processing information is found under Tape Input Files.

For unlabeled ASCII files, TPMARK=NO is the only valid entry. If the parameter is omitted entirely, TPMARK=NO is the default. Tapemarks are not supported on unlabeled ASCII files. Special error recovery procedures facilitate reading backwards.

TAPE INPUT FILES

For a magnetic tape input file, OPEN, CLOSE, or an end-of-volume condition rewinds the tape as specified by the DTFSR or DTFMT REWIND parameter. No rewind is defined by the DTFPH. Tape file positioning depends on the labels to be processed and is the user's responsibility.

If any labels other than standard labels are to be checked by the user, a LABADDR routine must be supplied. The user's LABADDR routine, specified in the DTF, cannot issue a macro that calls a transient routine.

When an end-of-file condition occurs, IOCS branches to the user's EOFADDR routine specified in the DTF. Generally, the user issues a CLOSE in this routine to initiate rewind procedures for the tape (as specified by the DTF REWIND parameter), and deactivates the file. If CLOSE is issued for any tape input file before the end-of-data is reached, the rewind option is executed and the file is deactivated without any subsequent label checking.

When reading backwards (READ=BACK) a labeled tape must be positioned just past the tapemark following the label set. An unlabeled file must be positioned just past the tapemark after the data set. Although ASCII unlabeled tapes contain no leading tapemark, a read backwards operation can be performed due to special error recovery procedures.

Label checking of both standard and nonstandard labels is similar. That is, IOCS still processes standard labels, and the user's routine (if specified) still processes user or nonstandard labels. The only difference is that the volume label is not read immediately for standard labels, the trailer labels are processed in reverse order (relative to writing), and header labels are processed at EOF time, also in reverse order. If physical IOCS macros are used to read records backwards, labels cannot be checked (DTFPH must not be specified).

Because backward reading is confined to one volume, an end-of-file condition always exists when the file header label is encountered. At end-of-file for standard labels, IOCS checks only the block count (which was stored from the trailer label) and then branches to the user's EOFADDR routine. At EOF for nonstandard labels, IOCS branches to the user's LABADDR where the header label may be checked. To check labels, the user must evoke physical IOCS macro instructions to read the label(s). For example, OPEN, CLOSE, DUMP, PDUMP, CANCEL, and CHKPT cannot be issued. Also, when processing multivolume files, the user's label routine must save and restore register 15 if any logical IOCS macros other than LBRET are used. When user standard labels are checked, the checking follows that for standard labels.

When logical IOCS senses a tapemark on a tape input file, either an end-of-file or end-of-volume condition exists. This condition is determined by IOCS or by the user, depending on the type of labels (if any) used for the file, and the appropriate functions are performed.

IOCS can determine an end-of-volume condition only when trailer labels have been checked (see Checking Standard Labels or Checking Nonstandard Labels). If labels are not processed, the user's EOFADDR routine must process the condition (see FEOV Macro). When IOCS does detect the EOV condition, it switches to an alternate unit as designated in a job control ASSGN statement. If an alternate drive is not specified, a message to mount a new volume is issued. At this time, the operator may also cancel the job. When the operator mounts the volume, processing resumes. If the input file is processed by physical IOCS (DTFPH), the user must issue an OPEN macro for the new volume. Then, IOCS checks the header label(s) and processing continues.

In some cases, the user may desire to force an end-of-volume condition at a point other than at the normal tapemark. He may want to discontinue reading the records on the present volume and continue reading records on the next volume. This may be necessary because of some major change in record category or in processing requirements. An FEOV (forced end-of-volume) is available to the programmer for cases such as this. See FEOV Macro.

Checking Standard Labels on Tape (INPUT)

When standard labels are to be checked (DTFMT or DTFSR FILABL=STD or DTFPH TYPFLE=INPUT), the user must supply the TLBL statement for standard label information. Also, when processing standard labels, a LBLTYP statement may be required to define storage for tape label files (see the System Control and Service publication).

When standard labeled files positioned at load point are opened, IOCS requires that the first record be a volume (VOL1) label. The next label could be any HDR1 label preceding the file. IOCS locates the correct file header (HDR1) label by checking the file sequence number.

After checking the standard label (if user standard labels UHL1-UHL8 or UTL1-UTL8 for EBCDIC files, UHLa or UTLa for ASCII files), IOCS enters the LABADDR routine (see Tape Input Files) and enters an alphabetic O, V, or F in the low-order byte of register 0.

O indicates header labels.
V indicates end-of-volume labels.
F indicates end-of-file labels.

The user's routine can test this character to determine what labels should be checked. IOCS also loads the address of an 80-byte IOCS label area in register 1 (register 1 must remain positive).

After each label is checked, a LBRET 2 macro can be issued for IOCS to read the next label. However, if a tapemark is read instead, label checking is terminated. If the user wishes to end label checking before all labels are read, he can issue a LBRET 1 macro. After all trailer labels are checked, IOCS initiates EOV or EOF procedures (see Tape Input Files).

Checking Nonstandard Labels on Tape (INPUT)

Any tape labels not conforming to the standard label specifications are considered nonstandard. It is the user's responsibility to check such labels. The MTC job control command can be issued to skip the necessary number of tapemarks or records to position the file. On input, nonstandard labels may or may not be followed by a tapemark. This choice, combined with the user's requirements to check the labels, results in the following possible conditions that can be encountered:

1. One or more labels, followed by a tapemark, are to be checked.

2. One or more labels, not followed by a tapemark, are to be checked.

3. One or more labels, followed by a tapemark, are not to be checked.

4. One or more labels, not followed by a tapemark, are not to be checked.

For conditions 1 and 2, the DTFMT or DTFSR operands FILABL=NSTD and LABADDR=name must be specified in the file definition. For condition 3, the operand FILABL=NSTD must be specified. If LABADDR is omitted, IOCS skips all labels, bypasses the tapemark, and positions the tape at the first data record to be read. For condition 4, the entries FILABL=NSTD and LABADDR=name must be specified. In this case, IOCS cannot distinguish labels from data records because there is no tapemark to indicate the end of the labels. Therefore, the user must read all labels (even though checking is not desired) to position the tape at the first data record.

Each time IOCS opens a file or reads a tapemark it supplies the hexadecimal representation (in the two low-order bytes of register 1) of the symbolic unit currently in use. See the CCB Format, bytes 6 and 7, for these values. IOCS also loads an alphabetic O in the low-order byte of register 0 when the file is opened.

When the user's routine gains control, the tape is not moved by OPEN. Physical IOCS macros must be used to transfer labels from tape to main storage. Therefore, the user must establish a command control block (CCB) and a channel command word(s) (CCW). The macro EXCP is used to initiate the transfer. After all labels are checked, the user returns control to OPEN by use of the LBRET 2 macro.

When IOCS reads a tapemark, it checks to determine if a user's LABADDR routine was supplied. If a LABADDR routine was supplied, IOCS exits to the routine. Otherwise, IOCS skips the labels and branches to the EOFADDR routine. In the LABADDR routine, physical IOCS macro instructions must be used to read the user's label(s). Furthermore, he must determine the EOF and/or EOV condition and indicate which to IOCS by loading either EF (end-of-file) or EV (end-of-volume) in the two low-order bytes of register 0. When this information is passed to IOCS, it initiates the end-of-file or end-of-volume procedures (see Tape Input Files).

Unlabeled Input Files on Tape (INPUT)

The first record for unlabeled tapes (FILABL=NO) may or may not contain a tapemark. Unlabeled tapes with ASCII contain no leading tapemark. If a tapemark is present, the next record is considered to be the first data record. If there is no tapemark, IOCS reads the first record, determines that it is not a tapemark, and backspaces to the beginning of that record. The file can be properly positioned by use of the job control MTC command. When the tapemark following the last data record is read, IOCS branches to the end-of-file address.

This section consists of the sequential declarative macros for particular I/O devices followed by the imperative macro instructions.

After input and output files are defined by the file definition or declarative macros, the imperative macro instructions are issued to operate on those files. Normal operation consists of initializing the declarative macro by the OPEN(R) macro. This initialization begins the label processing function (see Label Processing). After the file is properly initialized, the records for the file are made available for processing. When processing for a file is completed, the file should be deactivated by the CLOSE(R) macro. Figure 7 shows the key to finding the proper macro instructions for a particular I/O file, which is defined by a declarative macro (DTF) and processed accordingly by the imperative macros.

Declarative Macro Instructions

There are two types of declarative macros: DTFs and module-generation macros. Each type of processing is divided by type of file, card, magnetic tape, etc. The DTF used with the file is discussed first, and then (where applicable) the module generation macro.

The examples in Appendix C show that the user need not specify names for his modules. In these cases, the user can skip the discussion on module-naming conventions following each module-generation macro instruction.

Ten DTFs can be used for sequential processing. The DTFCD, DTFCN, DTFDI, DTFMR, DTFMT, DTFOR, DTFPR, DTFPT, and DTFSD macros are subsets of the inclusive declarative macro, DTFSR. DTFSR is included for Basic Programming Support and Basic Operating System users. Program assembly and execution time is substantially improved by specifying the subsets instead of DTFSR.

CARD FILE (DTFCD)

Enter the symbolic name of the file (filename) in the name field and DTFCD in the operation field. The detail entries follow the DTFCD header card in any order. Figure 8 lists the keyword operands contained in the operand field.

```
┌-----------------------------------------┐
|BLKSIZE={n|80}                           |
└-----------------------------------------┘
```

Enter the length of the I/O area (IOAREA1). If the record format is variable or undefined, enter the length of the largest record. If this operand is omitted, the length is assumed to be 80.

```
┌-----------------------------------------┐
|CONTROL=YES                              |
└-----------------------------------------┘
```

This operand is specified if a CNTRL macro is to be issued for a file. If this operand is specified, CTLCHR must be omitted. The CNTRL macro cannot be used on an input file with two I/O areas (IOAREA2= is specified).

```
┌-----------------------------------------┐
|CRDERR=RETRY                             |
└-----------------------------------------┘
```

This operand applies to card output on the IBM 2540 and 2520. It specifies the operation to be performed if an error is detected.

If a punching error occurs, it is usually ignored and operation continues. The error card is stacked in pocket P1 (punch) and correct cards are stacked in the pocket selected by the user. If the CRDERR=RETRY operand is included and an error condition occurs, IOCS also notifies the operator and then enters the wait state. The operator can either terminate the job, ignore the error, or instruct IOCS to repunch the card. From this specification, IOCS generates a retry routine and a save area for the card punch record.

DECLARATIVE MACRO INSTRUCTIONS		1052 Printer-Keyboard	1285/1287/1288 Optical Reader	1403/1404/1443/1445/3211 Printer	1255/1259/1412/1419 Magnetic Character Reader	1442/2501/2520/2540 Reader	1442/2520/2540 Punch	2311 Disk Unit	2314/2319 Disk Unit	2321 Data Cell	2400-Series 3420 Magnetic Tape Unit	2671/1017 Paper Tape Reader	1018 Paper Tape Punch	1270/1275 Optical Reader Sorter
	DTFCD					X	X							
	DTFCN	X												
	DTFDI[1]			X		X	X	X	X		X			
	DTFMR				X									X
	DTFMT										X			
	DTFOR		X											
	DTFPR			X										
	DTFPT											X	X	
	DTFSD							X	X	X				
	DTFSR[2]		X	X		X		X			X	X[18]		
IMPERATIVE MACRO INSTRUCTIONS														
Initialization	OPEN(R)		X[4]	X[4]	X[4]	X[4]	X[4]	X[4]	X[4]	X[4]	X[4]	X[2]	X[17]	X[4]
	LBRET[3]							X	X	X	X			
Processing	CHECK				X			X[6]	X[6]		X[6]			X
	CHNG[2]										X[6]			
	CNTRL		X	X[15]		X[15]	X[15]	X[15]	X[15]	X[15]	X[15]			
	DISEN				X									X
	DSPLY		X[11]											
	ERET							X[16]	X[16]	X[16]	X[16]			
	GET	X	X[10]		X	X[13]		X	X	X	X	X		X
	LITE				X[14]									X[14]
	NOTE							X[6]	X[6]		X[6]			
	POINTR							X[6]	X[6]		X[6]			
	POINTS							X[6]	X[6]		X[6]			
	POINTW							X[6]	X[6]		X[6]			
	PRTOV			X[15]										
	PUT	X		X		X	X[12]	X[5]	X[5]	X[5]	X		X	
	RDLNE		X[10]											
	READ		X[11]		X			X[6]	X[6]		X[6]			X
	RELSE							X[7]	X[7]		X[7]			
	RESCN		X[11]											
	TRUNC							X[8]	X[8]		X[8]			
	WAITF		X[11]		X									X
	WRITE							X[6]	X[6]		X[6]			
Completion	FEOV										X			
	FEOVD							X	X	X				
	LBRET[3]							X	X	X	X			
	CLOSE(R)		X[4]	X[4]	X[4]	X[4]	X[4]	X[4]	X[4]	X[4]	X[4]	X[2]	X[17]	X[4]

Notes:
1 Use only with system logical units.
2 Recommended for compatibility use only.
3 Applies only if LABADDR is specified.
4 Always required for this file.
5 PUT rewrites an input DASD record if UPDATE is specified. GET and PUT cannot be used with workfiles.
6 Work files for DASD and magnetic tape only.
7 Applies only to blocked input records.
8 Applies only to blocked output records.
9 Applies only when 2 selector channels and one or more 2-channel simultaneous-read-while-write tape-controlled units are installed.
10 Journal tape processing only.
11 1287/1288 document processing only.
12 PUT punches an input card with additional information if TYPEFLE=CMBND is specified.
13 In the 2540, GET normally reads cards in the read feed. If TYPEFLE=CMBND is specified, GET reads cards at the punch-feed-read station.
14 For the 1419 or 1275 with the Pocket Light Feature.
15 This macro cannot be used with DTFDI.
16 Applies only if ERREXT is specified.
17 Required if two I/O areas.
18 Valid for 2671 only.

Figure 7. Sequential Input/Output Macro Instructions

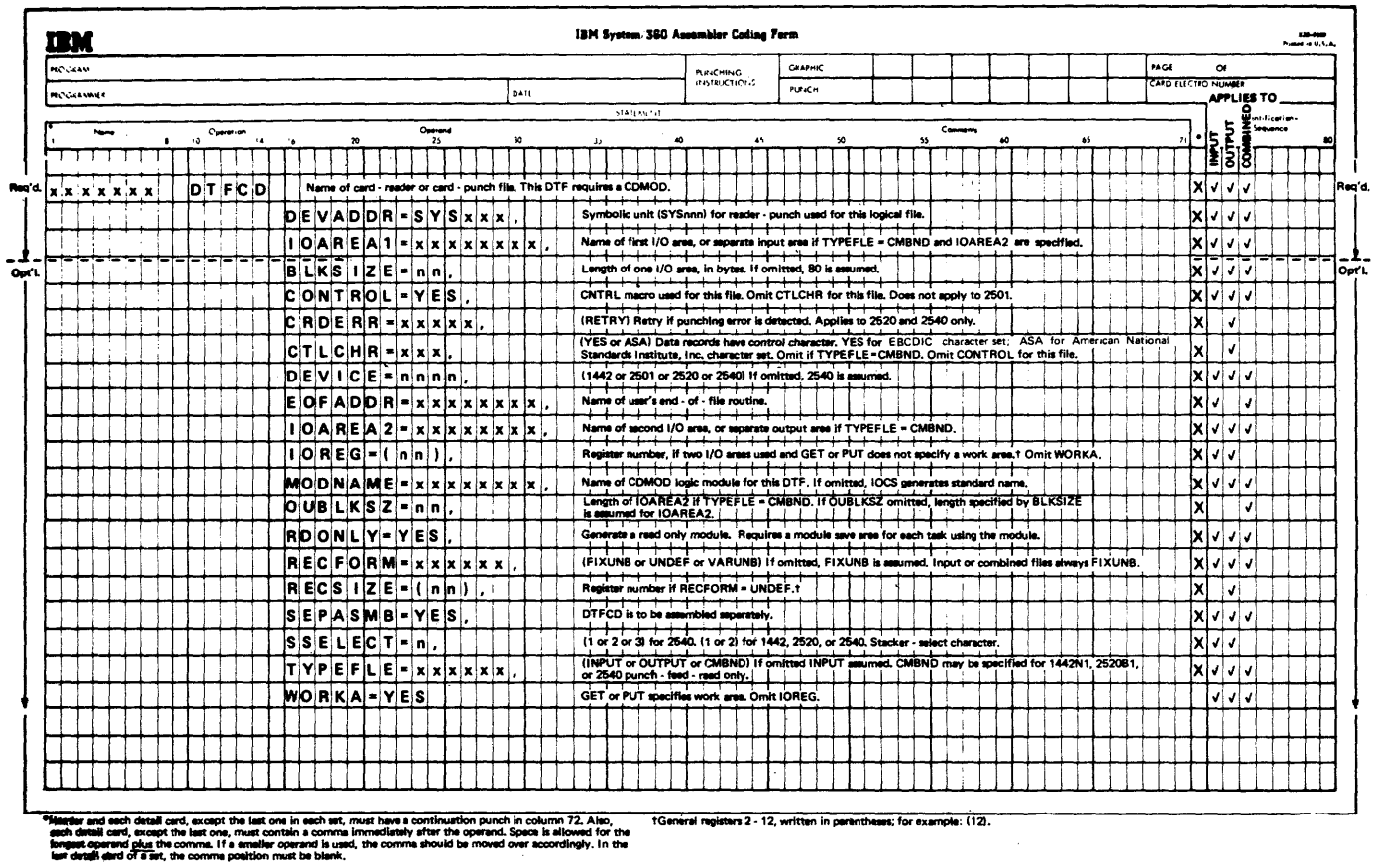


Figure 8. DTFCD Macro

```
| CTLCHR={ASA|YES}                                    |
```

This operand is required if first-character control is to be used on an output file. ASA denotes the American National Standards Institute, Inc. character set. YES denotes the EBCDIC character set. Appendix B contains a complete list of codes. This entry does not apply to combined files. If this operand is specified, CONTROL must be omitted.

```
|DEVADDR={SYSIPT|SYSPCH|SYSRDR|SYSnnn}               |
```

This operand specifies the symbolic unit to be associated with a file. The symbolic unit represents an actual I/O device address and is used in the job control ASSGN statement to assign the actual I/O device address to the file.

```
|DEVICE={1442|2501|2520|2540}                        |
```

This operand specifies the I/O device associated with a logical file. The acceptable entries are 1442, 2501, 2520, or 2540. If this operand is omitted, 2540 is assumed.

```
|EOFADDR=name                                        |
```

This entry must be included for input and combined files and specifies the symbolic name of the user's end-of-file routine. IOCS automatically branches to this routine cn an end-of-file condition. In his routine, the programmer can perform any cperations required for the end of the file, and he generally issues the CLOSE instruction for the file.

IOCS detects end-of-file conditions in the card reader by recognizing /* punched in card columns 1 and 2. If cards are allowed to run out without a /* trailer card (and a /& card if end-of-job) an error condition results.

```
|IOAREA1=name                              |
```

This operand specifies the symbolic name of the input or output area used by this file. An address expression, name, is defined.

If issued for a combined file, this operand specifies the input area. If IOAREA2 is not specified, the area specified in this operand is used for both input and output.

```
|IOAREA2=name                              |
```

This operand specifies a second I/O area. An address expression is defined. If the file is a combined file and the operand is specified, the designated area is an output area.

```
|IOREG=(r)                                 |
```

If work areas are not used but two input or output areas are, this operand specifies the register (2-12) in which IOCS puts the address of the logical record. For output files, IOCS puts the address where the user can build a record. This operand may not be used for combined files.

```
|MODNAME=name                              |
```

This operand may be used to specify the name of the logic module that will be used with the DTF table to process the file. If the logic module is assembled with the program, the MODNAME in the DTF macro instruction must specify the same name as the CDMOD macro instruction. If this operand is omitted, standard names are generated for calling the logic module. If two DTF macro instructions call for different functions that can be handled by a single module, only one module is called.

```
|OUBLKSZ=n                                 |
```

This operand is used in conjunction with IOAREA2 for only a combined file. Enter the maximum number n of characters to be transferred at one time. If this entry is not included and IOAREA2 is specified, the same length as defined by BLKSIZE is assumed.

```
|RDONLY=YES                                |
```

This operand is specified if the DTF is used with a read only module. Each time a read only module is entered, register 13 must contain the address of a 72-byte doubleword aligned save area. Each DTF should have its own uniquely defined save area. Each time an imperative macro (except OPEN(R), LBRET, SETL, or SETFL) is issued using a particular DTF, register 13 must contain the address of the save area associated with that DTF. The fact that the save areas are unique or different for each task makes the module reentrant (that is, capable of being used concurrently by several tasks). For more information see Shared Modules and Files.

If an ERROPT or WLRERR routine issues I/O macro instructions using the same read cnly module that caused control to pass to either error routine, the problem program must provide another save area. One save area is used for the normal I/O operations, and the second for I/O operations in the ERROPT or WLRERR routine. Before returning to the module that entered the ERROPT routine, register 13 must contain the save area address originally specified for that DTF. If this operand is omitted, the module generated is not reenterable, and no save area is required.

```
|RECFORM={FIXUNB|VARUNB|UNDEF}             |
```

This operand specifies the record format of the file. If the record format is FIXUNB, this entry may be omitted. If TYPEFLE=CMBND, this operand must be FIXUNB.

```
|RECSIZE=(r)                               |
```

For undefined records, this operand specifies the register (2-12) that contains the length of the output record. The user must load the length of each record into the specified register before he issues the PUT instruction for the record.

```
|SEPASMB=YES                               |
```

This operand must be specified if the DTF assembled separately. It causes a CATALR card with filename to be punched ahead of the object deck and defines the filename as an entry point in this assembly.

```
|SSELECT=n                                |
```

This operand specifies the valid stacker-select character for a file. If this entry is not specified, cards are selected into NR (normal read) or NP (normal punch) pockets. This entry is not applicable to a combined file. See the CNTRL Macro for further information.

> Note: When this operand is used with a device other than an IBM 1442, the program ignores CONTROL=YES with input files.

```
|TYPEFLE={INPUT|OUTPUT|CMBND}             |
```

This operand specifies if a file is input, output, or combined. A combined file can be specified for an IBM 1442 or 2520 or for a 2540 with the Punch-Feed-Read feature. TYPEFLE=CMBND is applicable if both GETs and PUTs are issued to the same card file. If TYPEFLE=CMBND is specified, the system logical units SYSIPT, SYSPCH, or SYSRDR must not be specified in the DEVADDR operand.

```
|WORKA=YES                                |
```

If I/O records are processed in work areas instead of the I/O area, YES is specified for this entry. The programmer must set up the work area in main storage. The address expression of the work area, or a general purpose register must be specified in each GET and PUT macro.

CARD MODULE (CDMOD)

Listed here are the user-supplied operands for CDMOD. The first card contains CDMOD in the operation field and may contain a module name in the name field.

```
|CONTROL=YES                              |
```

Include this operand if the CNTRL macro instruction is used with the module and its associated DTFs. The module also processes files in which the CNTRL macro is not used. For additional information about CNTRL, see 2540 Card Read Punch Codes in the section CNTRL Macro.

If CONTROL is specified, the CTLCHR operand must not be specified. Also, this operand cannot be specified if IOAREA2 is used for an input file.

```
|CRDERR=RETRY                             |
```

Include this operand if error retry routines for the 2540 and 2520 punch-equipment check are included in the module. Whenever this operand is specified, any DTF used with the module must also specify the CRDERR operand. This operand does not apply to an input or a combined file.

```
|CTLCHR={ASA|YES}                         |
```

Include this operand if first character stacker select control is used. Either YES or ASA may be specified. Whenever this is included, any DTF to be used with the module must also specify the CTLCHR operand with the appropriate YES or ASA parameter. If CTLCHR is included, CONTROL may not be specified. This operand does not apply to a combined file or to an input file.

```
|DEVICE={2540|1442|2520|2501}             |
```

Include this operand to specify the I/O device used by the module. Any DTF associated with the module must have the same operand.

```
|IOAREA2=YES                              |
```

Include this operand if a second I/O area is used. Any DTF used with the module must also include the IOAREA2 operand. This operand is not required for combined files.

```
|RDONLY=YES                               |
```

This operand generates a read only module. RDONLY=YES must be specified in the DTF. For additional programming requirements concerning this operand, see the DTF RDONLY operand.

```
RECFORM={FIXUNB|VARUNB|UNDEF}
```

Specifies the record format: fixed-length, variable-length, or undefined. Any DTF used with the module must include the appropriate parameter in the RECFORM operand. For INPUT and COMBND files, only FIXUNB should be specified.

```
SEPASMB=YES
```

Include this operand if the logic module is assembled separately. A CATALR card with the module name (standard or user) is punched ahead of the object deck.

```
TYPEFLE={INPUT|OUTPUT|CMBND}
```

This operand generates a module for either input, output, or combined file. Any DTF used with the module must include the appropriate parameter in the TYPEFLE operand.

```
WORKA=YES
```

This operand must be included if records are to be processed in work areas instead of I/O areas. Any DTF used with the module must include the appropriate parameter in the WORKA operand.

Recommended Module Name for CDMOD

Each name begins with a 3-character prefix (IJC) and consists of a 5-character field corresponding to the options permitted in the generation of the module.

CDMOD name = IJCabcde

a = F RECFORM=FIXUNB (always for INPUT and CMBND files)
  = V RECFORM=VARUNB
  = U RECFORM=UNDEF

b = A CTLCHR=ASA (not specified if CMBND)
  = Y CTLCHR=YES
  = C CONTROL=YES
  = Z CTLCHR or CONTROL not specified

c = B RDONLY=YES and TYPEFLE=CMBND
  = C TYPEFLE=CMBND
  = H RDONLY=YES and TYPEFLE=INPUT
  = I TYPEFLE=INPUT
  = N RDONLY=YES and TYPEFLE=OUTPUT
  = O TYPEFLE=OUTPUT

d = Z WORKA and IOAREA2 not specified
  = W WORKA=YES
  = I IOAREA2=YES
  = B WORKA and IOAREA2
  = Z WORKA=YES not specified (CMBND file only)

e = 0 DEVICE=2540
  = 1 DEVICE=1442
  = 2 DEVICE=2520
  = 3 DEVICE=2501
  = 4 DEVICE=2540 and CRDERR
  = 5 DEVICE=2520 and CRDERR

Subset/Superset CDMOD Names

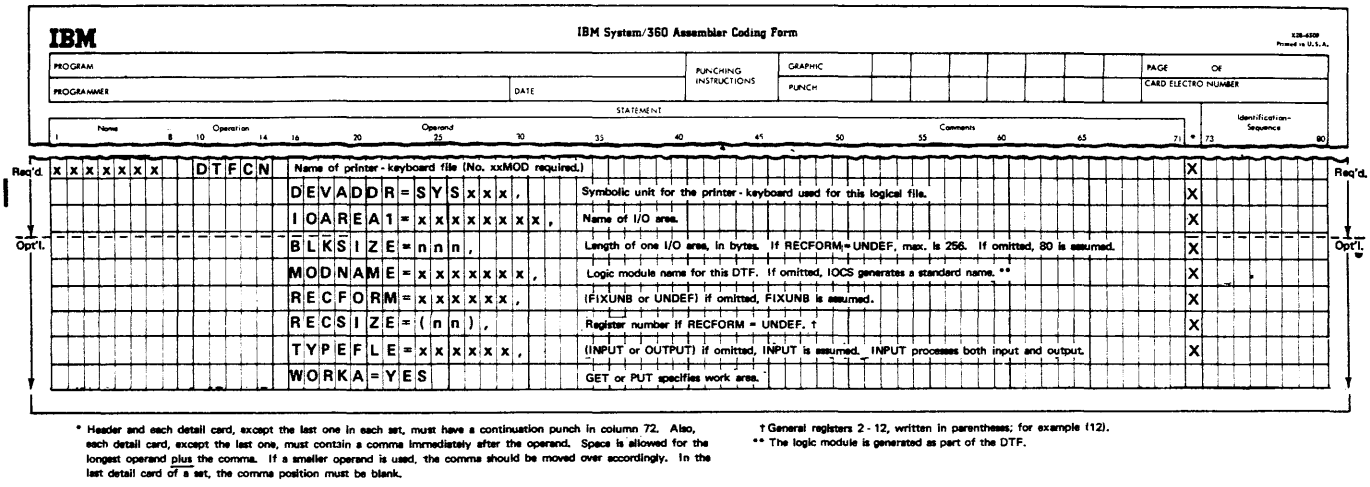
The following chart shows the subsetting and supersetting allowed for CDMOD names. All but one of the parameters are exclusive (that is, do not allow supersetting). A module name specifying C (CONTROL) in the b location is a superset of a module name specifying Z (no control or CTLCHR). A module with the name IJCFCIW0 is a superset of a module with the name IJCFZIW0. See Subset/Superset Module Names.

```
                * * * * *
      I J C F A B B 0
            V Y C I 1
            U + H W 2
              C I Z 3
              Z N   4
                O   5

+ Subsetting/supersetting permitted.
* No subsetting/supersetting permitted.
```

CONSOLE FILE (DTFCN)

DTFCN defines an input or output file that is processed on an IBM 1052 Printer-Keyboard, or a 3210 or 3215 Console Printer-Keyboard. DTFCN provides GET/PUT logic for the printer-keyboard file. The symbolic name of the file is entered in the name field, and DTFCN is entered in the operation field. The detail entries, in any order, follow the DTFCN header entry with keyword operands in the operand field. Figure 9 contains the DTFCN entries.



* Header and each detail card, except the last one in each set, must have a continuation punch in column 72. Also, each detail card, except the last one, must contain a comma immediately after the operand. Space is allowed for the longest operand plus the comma. If a smaller operand is used, the comma should be moved over accordingly. In the last detail card of a set, the comma position must be blank.

† General registers 2 - 12, written in parentheses; for example (12).

** The logic module is generated as part of the DTF.

Figure 9. DTFCN Macro

If a mistake is made when entering information on the printer-keyboard, simultaneously press the ALTERNATE CODE and the CANCEL keys. This issues a new read command, and you can retype the data from the beginning.

```
BLKSIZE=n
```

The number n designates the length of the I/O area. For the undefined record format, BLKSIZE must be as large as the largest record to be processed. The input/output records must not exceed 256 characters.

If the console buffering option is specified at system generation time and the printer-keyboard is assigned to SYSLOG, physical IOCS can increase throughput on each actual output record not exceeding 80 characters. This increase in throughput results from starting the output I/O command and returning to the problem program before output completion. Regardless of whether output records are buffered (queued on an I/O completion basis) or not, they are always printed in a first-in-first-out (FIFO) order.

```
DEVADDR={SYSLOG|SYSnnn}
```

This operand specifies the symbolic unit associated with the logical file. In a multiprogramming environment, DEVADDR=SYSLOG must be specified to obtain Background (BG), Foreground 1 (F1), or Foreground 2 (F2), prefixes for message identification.

```
IOAREA1=name
```

This operand specifies the symbolic name of the I/O area used by the file. The I/O area is not cleared before or after a message is printed, or when a message is canceled and reentered on the console.

```
MODNAME=name
```

This operand can specify the name of the logic module generated by this DTFCN macro.

If this entry is omitted, standard module names are generated for the logic module.

A module name must be given when two phases (each containing a DTFCN macro) are linkage edited into the same program. Under such conditions, omission of this operand results in unresolved address constants.

```
RECFORM={FIXUNB|UNDEF}
```

This operand specifies the record format of the file. FIXUNB is assumed.

```
┌-----------------------------------------┐
|RECSIZE=(r)                              |
└-----------------------------------------┘
```

For undefined records, this operand is required for output files and is optional for input files. It specifies a general register (2-12) that contains the length of the record. On output, the user must load the length of each record into the designated register before he issues a PUT macro. If specified for input files, IOCS provides the length of the record transferred to main storage.

```
┌-----------------------------------------┐
|TYPEFLE={INPUT|OUTPUT}                   |
└-----------------------------------------┘
```

If INPUT is specified, coding is generated for both input and output files. If OUTPUT is specified, coding is provided for output file only.

```
┌-----------------------------------------┐
|WORKA=YES                                |
└-----------------------------------------┘
```

This operand indicates that a work area is used with the file. For a GET or PUT macro IOCS moves the record to or from the work area.

DEVICE INDEPENDENT FILE (DTFDI)

The DTFDI macro is device independent for system logical units. For any number of DTFDI macros, if they are assembled within one program and all of them have the same RDONLY condition, only one logic module (DIMOD) is required. Therefore, DTFDI processing requires fewer parameters and less main storage than multiple LIOCS macros. Also, it allows the programmer to change device assignments without reassembling the logic module.

The restrictions on DTFDI processing are:

- Only fixed unblocked records are supported.
- Only forward reading is allowed.
- Rewind options are not provided.
- Combined file processing is not supported for reader-punches.
- The CNTRL and PRTOV macros cannot be used with this macro.
- Reading, writing, or checking of standard or user-standard labels for tape/disk is not supported.
- If ASA control character code is used in a multitasking environment and more than one DTF is using the same module with RDONLY=YES, overprinting may occur.
- If DTFDI is used with DASD, FOPT SYSFIL must be specified at system generation time.

The symbolic name of the file should be entered in the name field and DTFDI in the operation field. The entries for the DTFDI macro are discussed here and summarized in Figure 10.

```
┌-----------------------------------------┐
|DEVADDR={SYSIPT|SYSLST|SYSPCH|SYSRDR}     |
└-----------------------------------------┘
```

This operand must specify the symbolic unit associated with this system logical file. Only these system names may be specified.

```
┌-----------------------------------------┐
|EOFADDR=name                             |
└-----------------------------------------┘
```

This operand must specify the symbolic name of the user's end-of-file routine. It is required only if SYSIPT or SYSRDR is specified.

IOCS branches to this routine when it detects an end-of-file condition. In this routine, the programmer can perform any operations necessary for the end-of-file condition, although the CLOSE(R) macro instruction is generally issued.

IOCS detects the end-of-file condition by recognizing a /* in positions 1 and 2 of the record for cards, a tapemark for tape and a filemark for disk. If the records are allowed to run out without a /* (and a /&, if end-of-job) an error condition results.

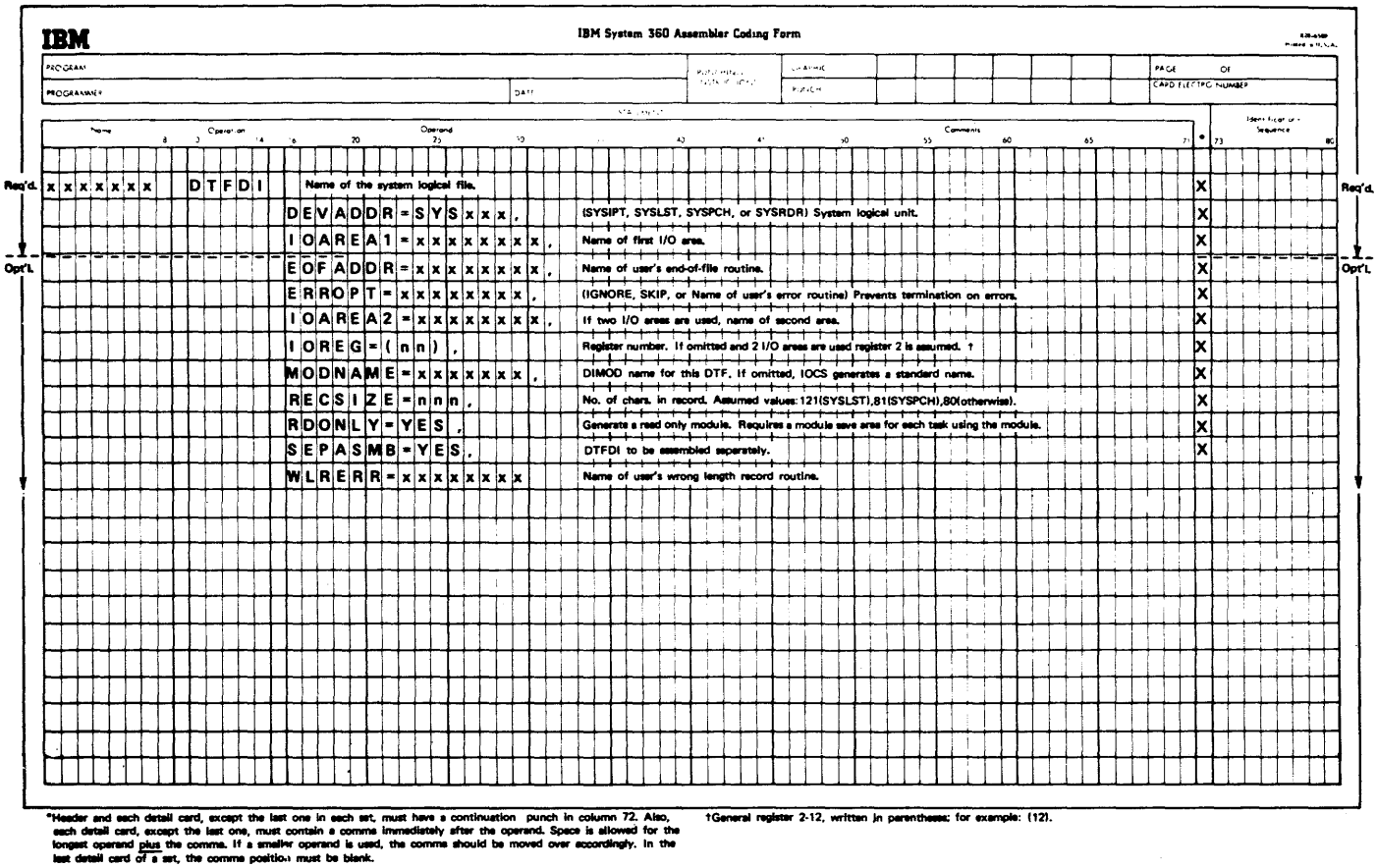


Figure 10. DTFDI Macro

```
|ERROPT={IGNORE|SKIP|name}                |
```

This operand does not apply to output files. For output files, the job is automatically terminated after IOCS has attempted to retry writing the record. It does, however, apply to wrong-length records if WLRERR is omitted. If both ERROPT and WLRERR are omitted and wrong-length records occur, IOCS ignores the error.

ERROPT specifies the function performed for an error block. If an error is detected when reading a magnetic tape or a disk pack, IOCS attempts to recover from the error. If the error is not corrected, the job is terminated unless this operand is included to specify other procedures to be taken. The functions of these three specifications are:

IGNORE: The error condition is ignored and the address of the error record is made available to the user for processing (see CCB).

SKIP: The error block is not made available for processing. The next record is read and processing continues.

name: IOCS branches to the user's routine, where the user may perform whatever functions he desires or note the error condition. The address of the error record is supplied in register 1. The contents of the IOREG register may vary and should not be used for error records. Also, the programmer must not issue any GET instructions in his error routine. If he uses any other IOCS macros, he must save the contents of register 14. If RDONLY=YES is specified, he must also save the contents of register 13. At the end of his error routine, he must return to IOCS by branching to the address in register 14. When control returns to the problem program, the next record is made available to the user for processing.

```
|IOAREA1=name                              |
```

This operand must specify the symbolic name of the input or output area used with the file. The input and/or output routines transfer records to or from this area.

If the DTFDI macro is used to define a printer file, or a card file to be processed on an IBM 2540 Card Read Punch, the first byte of the main-storage output area must contain a control character.

```
|IOAREA2=name                              |
```

Two input or output areas can be allotted for a file to permit overlapped GET or PUT processing. If this operand is included, it specifies the symbolic name of the second I/O area.

```
|IOREG={(r)|(2)}                           |
```

When two I/O areas are used, this operand specifies the general purpose register (2-12) that points to the address of the next record. For input files, it points to the logical record available for processing. For output files, it points to the address of the area where the user can build a record. If omitted, and two I/O areas are used, register 2 is assumed.

```
|MODNAME=name                              |
```

This operand may be used to specify the name of the logic module used with the DTF table to process the file. If the logic module (DIMOD) is assembled with the program, the MODNAME parameter in this DTF must specify the same name as the DIMOD macro instruction.

If this entry is omitted, standard names are generated for calling the logic module. If two different DTF macro instructions call for different functions that can be handled by a single module, only one standard-named module is called.

```
|RDONLY=YES                                |
```

This operand is specified if the DTF is to be used with a read only module. Each time a read only module is entered, register 13 must contain the address of a 72-byte doubleword aligned save area. Each DTF should have its own uniquely defined save area and each time an imperative macro (except OPEN(R), LBRET, SETL, or SETFL) is issued using a particular DTF, register 13 must contain the address of the save area associated with that DTF. The fact that the saveareas are unique for each task makes the module reentrant (that is, capable of being used concurrently by several tasks). For more information see Shared Modules and Files.

If an ERROPT or WLRERR routine issues I/O macro instructions using the same read only module that caused control to pass to either error routine, the problem program must provide another save area. One save area is used for the initial I/O operations, and the second for I/O operations in the ERROPT or WLRERR routine. Before returning to the module that entered the ERROPT routine, register 13 must be set to the save area address originally specified for that DTF. If the operand is omitted, the module generated is not reenterable and no save area need be established.

```
|RECSIZE=n                                 |
```

This operand specifies the length of the record. For input files (SYSIPT and SYSRDR), the maximum allowable record size is 80 bytes. For output files, RECSIZE must include one byte for control characters. The maximum length specification is 121 for SYSLST and 81 for SYSPCH.

For printers and punches, DIMOD assumes a System/360 control character if the character is not a valid ASA character. The program checks ASA control characters before System/360 control characters. Therefore, if it is a valid ASA control character (even though it may also be a System/360 control character), it is used as an ASA control character. Otherwise, it is used as a System/360 control character.

Control character codes are listed in Appendix B with the following exceptions:

- 2520 stacker selection codes must be used for the 1442.

- 2540 stacker selection 3 must not be used if device independence is to be maintained.

If this operand is omitted, the following assumptions are made:

80 bytes for SYSIPT and SYSRDR.
81 bytes for SYSPCH.
121 bytes for SYSLST.

The use of assumed values for the RECSIZE operand assures device independence. For disk files, the assumed values are required to assure device independence.

```
SEPASMB=YES
```

This operand must be included if the DTF is assembled separately from the problem program. It causes the object deck to be preceded by a CATALR filename card.

```
WLRERR=name
```

This entry applies to input files only. It pecifies the symbolic name of a user's routine to which IOCS branches if a wrong-length record is read on a tape or disk device.

Because only fixed-length records are allowed, a wrong-length record error condition results when the length of the record read is not equal to the RECSIZE parameter. If the record is less than the RECSIZE parameter, the first two bytes of the CCB (first 16 bytes of the DTF) contain the number of bytes left to be read (residual count). If the record to be read is larger than the RECSIZE parameter, the residual count is set to zero and there is no way to compute its size. The number of bytes transferred is equal to the RECSIZE parameter and the remainder of the record is truncated.

The address of the record is supplied in register 1. In his routine, the user can perform any operation except issuing another GET for this file. Also if he uses any other IOCS macros in his routine, he must save the contents of register 14. If RDONLY=YES, he must also save the contents of register 13 as well.

At the end of the routine, the user must eturn to IOCS by branching to the address n register 14. When control returns to the problem program, the next record is made available.

If this operand is omitted, but a wrong-length record is detected by IOCS, the action depends on whether the ERROPT operand is included.

- If the ERROPT operand is included, the wrong-length error record is treated as an error record and handled according to the ERROPT parameter.
- If the ERROPT operand is omitted, IOCS ignores wrong-length errors and the record is made available to the user. If, in addition to a wrong-length record error, an unrecoverable parity error occurs, the job is terminated.

DEVICE INDEPENDENT MODULE (DIMOD)

Listed here are the user-supplied operands for DIMOD. The header card contains DIMOD in the operation field and may contain a module name in the name field. If the name field is omitted, a system name is generated in a manner consistent with the recommended module name conventions.

```
IOAREA2=YES
```

Include this operand if a second I/O area is needed. A module with this operand can process DTFDIs with one or two I/O areas. If the operand is omitted or is invalid, one I/O area is assumed.

```
RDONLY=YES
```

This operand causes a read only module to be generated. RDONLY=YES must be specified in the DTF. For the programming requirements of this operand, see the DTF RDONLY operand.

```
SEPASMB=YES
```

This operand must be included if the logic module is assembled separately. This causes a CATALR card with the module name (standard or user) to be punched ahead of the object deck.

```
TYPEFLE={INPUT|OUTPUT}
```

Include this operand to specify whether the module is to process input or output files. If OUTPUT is specified, the generated module can process both input and output files.

Recommended Module Name for DIMOD

Each name begins with a 3-character prefix (IJJ) followed by a 5-character field corresponding to the options permitted in the generation of the module.

DIMOD name = IJJabcde

a = F always

b = C always

c = B TYPEFLE=OUTPUT(processes both input and output)
  = I TYPEFLE=INPUT

d = I IOAREA2=YES
  = Z IOAREA2=YES is not specified

e = C RDONLY YES
  = D RDONLY=YES is not specified

Subset/Superset DIMOD Names

The following diagram illustrates the subsetting and supersetting allowed for DIMOD names. All of the variable entries allow subsetting. A module name specifying B is a superset of the module specifying I; for example, IJJFCBID is a superset of the module IJJFCIID. See Subset/Superset Module Names.

```
                + + *
      I J J F C B I C
                I Z D

+ Subsetting/supersetting permitted.
* No subsetting/supersetting permitted.
```

If two or more modules with the same entry points are included, message 2143I is given. This message can usually be suppressed by including a superset module.

MAGNETIC READER FILE (DTFMR)

DTFMR defines an input file processed on an IBM Magnetic Character Reader (1255, 1259, 1412, 1419) or Optical Reader/Sorter (1270, 1275).

Enter the symbolic name of the file in the name field and DTFMR in the operation field. The entries are discussed here and illustrated in Figure 11.

```
DEVADDR=SYSnnn
```

This operand specifies the symbolic unit (SYSnnn) associated with the primary address for the file. The symbolic unit represents an actual I/O device address used in the job control ASSGN statement to assign the actual I/O device address to the file.

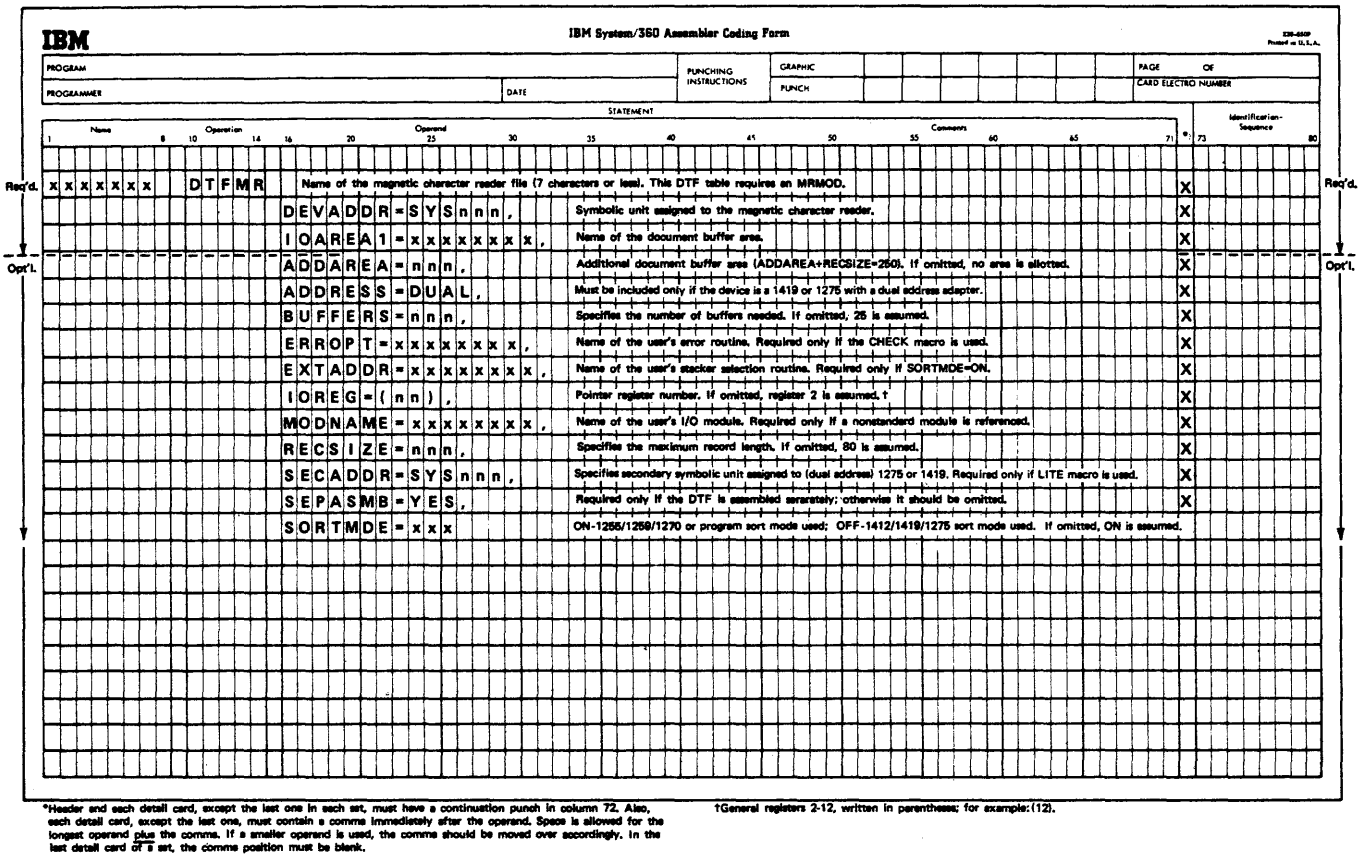
```
IOAREA1=name
```

This operand specifies the symbolic name of the document buffer area used by the file. Figure 12 shows the format of the document buffer area.

```
IOREG={(r)|(2)}
```

This operand specifies the general-purpose register that the input/output and user routines use to indicate which individual document buffer is available for processing. IOCS puts the address of the current document buffer in the specified register each time a GET or READ is issued. Any register parameter from 2 to 12 may be specified, but 2 is assumed if this operand is omitted.

The same register may be specified in the IOREG entry for two or more files in the same program, if desired. In this case, the problem program may need to store the address supplied by IOCS for each record.



IBM System/360 Assembler Coding Form

Name	Operation	Operand	Comments	Identification-Sequence
Req'd. xxxxxx	DTFMR	Name of the magnetic character reader file (7 characters or less). This DTF table requires an MRMOD.		X Req'd.
	DEVADDR=SYSnnn,	Symbolic unit assigned to the magnetic character reader.		X
	IOAREA1=xxxxxxxx,	Name of the document buffer area.		X
Opt'l.	ADDAREA=nnn,	Additional document buffer area (ADDAREA+RECSIZE=250). If omitted, no area is allotted.		X Opt'l.
	ADDRESS=DUAL,	Must be included only if the device is a 1419 or 1275 with a dual address adapter.		X
	BUFFERS=nnn,	Specifies the number of buffers needed. If omitted, 25 is assumed.		X
	ERROPT=xxxxxxxx,	Name of the user's error routine. Required only if the CHECK macro is used.		X
	EXTADDR=xxxxxxxx,	Name of the user's stacker selection routine. Required only if SORTMDE=ON.		X
	IOREG=(nn),	Pointer register number. If omitted, register 2 is assumed. †		X
	MODNAME=xxxxxxxx,	Name of the user's I/O module. Required only if a nonstandard module is referenced.		X
	RECSIZE=nnn,	Specifies the maximum record length. If omitted, 80 is assumed.		X
	SECADDR=SYSnnn,	Specifies secondary symbolic unit assigned to (dual address) 1275 or 1419. Required only if LITE macro is used.		X
	SEPASMB=YES,	Required only if the DTF is assembled separately; otherwise it should be omitted.		X
	SORTMDE=xxx	ON-1255/1259/1270 or program sort mode used; OFF-1412/1419/1275 sort mode used. If omitted, ON is assumed.		

*Header and each detail card, except the last one in each set, must have a continuation punch in column 72. Also, each detail card, except the last one, must contain a comma immediately after the operand. Space is allowed for the longest operand plus the comma. If a smaller operand is used, the comma should be moved over accordingly. In the last detail card of a set, the comma position must be blank.

†General registers 2-12, written in parentheses; for example: (12).

Figure 11. DTFMR Macro

```
|ADDAREA=n                                                    |
```

This operand must be included only if additional buffer work area is needed. The parameter n specifies the number of additional bytes the user desires in each buffer. The ADDAREA and RECSIZE specifications (n) must total less than or equal to 250. This area can be used as a work area and/or output area and is reset to binary zeros when the next GET or READ for a file is executed.

```
|ADDRESS=DUAL                                                 |
```

This operand must be included only if the 1419 or 1275 contains the Dual Address Adapter. If Single Address Adapter is used, this operand must be omitted.

```
|BUFFERS={n|25}                                               |
```

This operand is included to specify the number of buffers in the document buffer area. The minimum number is 12, the maximum is 254, and 25 is the assumed value if this operand is omitted.

```
|ERROPT=name                                                  |
```

This operand may be included only if the CHECK macro instruction is used. The parameter (name) specifies the name of the routine that the CHECK macro branches to if any error condition is posted in byte 0, bits 2-4 (and bit 5, if no control address is specified in the CHECK macro) of the buffer status indicators. It is the user's responsibility to exit from this routine (see the CHECK Macro instruction).

EXTADDR=name

This operand specifies the address of the user stacker selection routine to which control is given when an external interrupt is encountered while reading and sorting the documents internally. The only case when this operand may be omitted is when SORTMDE=OFF is specified.

MODNAME=name

This operand specifies the name of the logic module MRMOD. If omitted, IOCS generates the normal system module name.

RECSIZE={n|80}

This operand specifies the actual length of the data portion of the buffer. The record size specified must be the size of the largest record processed. If this operand is omitted, a record size (n) of 80 is assumed. The ADDAREA and RECSIZE specifications (n) must total less than or equal to 250.

SECADDR=SYSnnn

This operand specifies the symbolic unit to be associated with the secondary control unit address if the 1419 or 1275 with the Dual Address Adapter and LITE macro are utilized. The operand should be omitted if the pocket LITE macro is not being used.

SEPASMB=YES

Include this operand only if the DTFMR is assembled separately. This causes a CATALR card with the filename to be punched ahead of the object deck and defines the filename as an ENTRY point in the assembly. If the operand is omitted, the program assumes that the DTF is being assembled with the user program and no CATALR card is punched.

SORTMDE={ON|OFF}

This operand specifies the method of sorting done on the 1412/1419. SORTMDE=ON indicates that the program sort mode is being used. (For the 1259 or 1270, this is the only mode that can be used.) SORTMDE=OFF indicates that sorting is under control of the magnetic character reader. If omitted, the program sort mode is assumed.

Characteristics of Magnetic Character (MICR) Processing and Optical Reader/Sorter Processing

Logical IOCS allows the user to operate the Magnetic Ink Character Recognition (MICR) or Optical Reader/Sorter devices in either a foreground or background area. The MICR user is supplied with an extension to the DOS supervisor which monitors, by means of external interrupts, the reading of documents into a user supplied I/O area (document buffer area). The user must access all MICR documents through logical IOCS. Logical IOCS gives the user the next sequential document and automatically engages and disengages the devices, as necessary, to provide a continuous stream of input. Detected error conditions and information are passed to the user in each document buffer.

The magnetic character readers and optical reader/sorters are unique in that documents must be read at a rate dictated by the device rather than by the program. To allow time for necessary processing (including determination of pocket selection), the device generates an external interruption at the completion of each read operation for each document. The supervisor gives absolute priority to external interrupt processing.

In a multiprogramming system with MICR document processing, any partition (background or foreground) can utilize MICR devices. For programs with one MICR device, the GET macro instruction is provided. For multiple MICR processing, READ, CHECK, and WAITF macro instructions are provided to allow user processing to continue as long as one file has documents ready for processing.

Before the user can begin any type of MICR processing, he must be aware of the MICR document buffer format (see Appendix E). Each document buffer must n exceed 256 bytes, including the six-byte buffer status indicators, any additional

user work area, and the maximum document data area. The minimum number of document buffers a user may specify is 12 and the maximum number is 254.

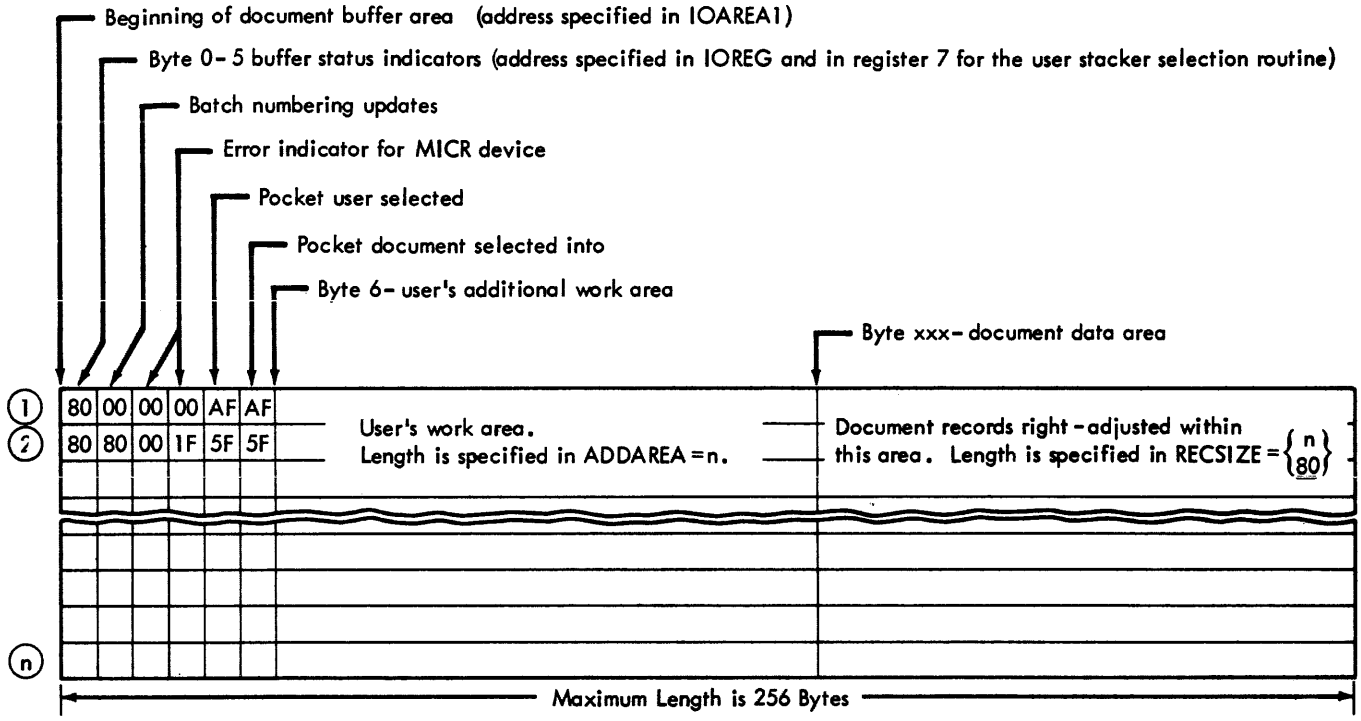
Figure 12 is a storage map of the document buffer area utilized in MICR processing. Visualizing this makes the concept of MICR processing easier to understand. Before any data is read into a document buffer, logical IOCS sets the entire document buffer (including the status indicators) to binary zeros. The GET (READ if multiple MICR device processing) macro instruction then engages the device, and documents are automatically read into the I/O area until the MICR device is out of documents, or the I/O area is filled. The external interrupt routine of the supervisor continually monitors the reading in of data so that processing of other document buffers is never disrupted. Also, at the completion of each read for a MICR document, the external interrupt routine interrupts the user program to give control to the user stacker selection routine to determine pocket selection for that document.

User Stacker Selection Routine for MICR

The user stacker selection routine resides in the user problem-program area and gains control of the system whenever a document is ready to be stacker selected. This routine determines which pocket to select the document into and whether batch numbering update is to be performed (1419 only). The entry point is specified in the DTFMR operand EXTADDR=name. All registers are saved upon exiting from, and restored upon returning to, the problem program. The use of the general registers in this routine is as follows:

Register	Comment
0-4, 6, 8-15	These registers are available to the user stacker selection routine for any purpose. Because the program can be interrupted at any time, the contents of these registers is unpredictable.
5	When the user stacker selection routine is entered, this register contains the address of the user stacker selection routine. Register 5 should be utilized as the base register for the routine.
7	This register always contains the address of the first byte of the buffer for the document being selected. Bytes 2 and 3 of this buffer (see Appendix E) indicate the read status of the document.

Before entering the user stacker selection routine, IOCS aids in stacker selection by setting the entire document buffer to binary zeros, reading the document into the document data area, and posting information in bytes 2 and 3. When the user stacker selection routine has determined which pocket to select the document into, the actual stacker selection command code for this pocket must be placed into byte 4 of the document buffer pointed to by register 7. The final destination of the document is indicated in byte 5 of the buffer. This indication is the same as byte 4 except in the case of a late stacker select, an auto-selected document, a program malfunction, or a device malfunction. Any of these results in an I/O error. Reject code X'CF' is placed into byte 5, indicating the document went to the reject pocket.



(1) Indicates the normal condition (no errors) when the document is being processed and the stacker selection is complete to pocket A (1412).

(2) Indicates the normal condition (no errors - all fields read) when the document is being processed and the stacker selection is complete to pocket 5 and batch numbering update was performed (1419 model 1 or 31).

(n) Number of buffers is limited only by the amount of main storage available (see BUFFERS= {n / 25}).

Figure 12. MICR Document Buffer Area

The command codes to be used to select pockets are:

Pocket	Code	
A	X'AF'	(1270, except Models 1 and 3, 1275, 1412 and 1419 only)
B	X'BF'	(1275, 1412 and 1419 only)
0	X'0F'	
1	X'1F'	
2	X'2F'	
3	X'3F'	
4	X'4F'	
5	X'5F'	(except 1270 Models 1 and 3)
6	X'6F'	
7	X'7F'	
8	X'8F'	
9	X'9F'	
Reject	X'CF'	

An invalid code placed in byte 4 puts the document into the reject pocket and posts bit 1 of byte 0 of the buffer. Byte 0, bit 2 of the next buffer is posted.

Before returning to a 1419 external interrupt routine via the EXIT MR macro instruction (required method), the user can request a batch numbering update. He can do this only within his 1419 stacker selection routine by turning ON byte 1, bit 0 in the current document buffer (OI 1(7), X'80').

For the 1419 (dual address), the user cannot obtain batch numbering update on an auto-selected document (byte 2, bit 6-ON). Such requests are ignored by the external interrupt routine.

TIMINGS FOR STACKER SELECTION: Because the MICR readers continuously feed documents while engaged, it is necessary to reinstruct the readers within a certain time limit after a read completion is signaled by an external interrupt. This period is generally called minimum stacker

selection time. This available time depends on the reader model, the length of documents being read, single or dual address adapter (1419, 1275), and the fields to be read on the 1419 or 1275 (dual address) only. Refer to the MICR publications listed in the Preface for a more complete description of device timings.

The minimum available stacker selection time for the MICR reader for 6-inch documents is:

Device	Time	Serial No. Read Field Key	Transit Routine Read Field Key
1255/1259 (with the use of pocket 0)	24 milliseconds	--	--
1255/1259 (without the use of pocket 0)	40 milliseconds	--	--
1270	24 milliseconds	--	--
1412	7.5 milliseconds	--	--
1419 (Single Address Adapter)	9.5 milliseconds	--	--
1419 (Dual Address Adapter)	15 milliseconds	ON	ON or OFF
	21 milliseconds	OFF	ON
	27 milliseconds	OFF	OFF
1275 (Refer to the IBM 1275 Optical Reader/Sorter publication listed in the Preface.)			

Note: Stacker selection times shown for the 1419 Dual Address Adapter are for the 1419 Models 1 and 31 only. Stacker selection times for the 1419 Model 32 are found in the IBM 1419 Model 32 Systems Reference Library publication listed in the Preface.

Failure to reinstruct the 1255, 1259, 1270, 1275, 1412, or 1419 (single address adapter) within the allotted time causes the document(s) processed after this time to be auto-selected into the reject pocket (late read condition). Failure to reinstruct the 1419 or 1275 (dual address adapter) within the allotted time causes the document being processed to be auto-selected into the reject pocket (late stacker-select condition).

To determine the amount of time available for the user stacker selection routine, and to minimize document rejects, the user should consider:

1. The minimum available stacker selection time. This time depends on the MICR device, the type of adapter (1419), the length of the documents to be read, and the fields to be read (see the applicable MICR device publication(s) listed in the IBM System/360 and System/370 Bibliography).
2. The model(s) of System/360 to be used (see the System/360 Model publication(s) listed in the IBM System/360 and System/370 Bibliography). Also, refer to Figure 13.
3. The maximum time required by the supervisor external interrupt routine. This time can be calculated as follows:

   If the device is a 1255, 1259, 1412, or a 1419 (single address), add 0.1 (2030 Processor 1.5 mu), 0.13 (2030 Processor 2.0 mu), or 0.06 (2040 Processor) milliseconds for each MICR reader.

Number of MICR Readers on Processor	Batched Job Environment			Multiprogramming Environment		
	2030 Processor (1.5 microsecond)	2030 Processor (2.0 microsecond)	2040 Processor	2030 Processor (1.5 microsecond)	2030 Processor (2.0 microsecond)	2040 Processor
1	2.5	3.3	1.5	2.5	3.3	1.5
2	5.3	7.0	3.2	6.2	8.3	3.8
3	7.2	9.6	4.4	8.1	10.8	4.9
4	9.1	12.1	5.5	10.0	13.4	6.0
5	11.0	14.6	6.6	11.9	15.9	7.2
6	12.9	17.2	7.8	13.8	18.4	8.3

Notes:

1. The timings shown are in milliseconds.
2. These timings include the most time consuming cases. That is, if all the MICR devices interrupt simultaneously, the supervisor can successfully process them within the given time.

Figure 13. Stacker Selection Times for IBM 2030 and 2040 Processors

4. Also, another timing consideration is the concurrent operation interference factor. These interference factors are:

   a. Any I/O operation(s) in progress use machine cycles for command execution and data transfer. The user must consider his I/O configuration and probable concurrent I/O operations and consult the appropriate System/360 model publication to calculate this interference value.

   b. The external interruption may occur while the central processing unit is executing either some long-executing instruction such as an MVC (move), or TR (translate), or an I/O interruption. For instance a 256-byte MVC instruction can prevent an interruption for 800 microseconds on a 1.5 microsecond Model 30. The I/O interruption interference may take an additional 380 (500, 250) microseconds on the same model (2.0 Model 30, Model 40).

As an example of the previous steps, assume a hypothetical situation:

- A 2030 Processor (1.5)
- A 1419 (single-address) MICR device
- 6-inch documents to be read
- A MICR batched-job program
- A user stacker selection routine to be written

The minimum available stacker selection time for a 2030 Processor is 9.5 milliseconds. If the external interrupt processing routine time is 2.5 milliseconds (for 6-inch documents), 7 milliseconds is available for the user stacker selection time. But from this 7 milliseconds, the timing consideration of the MICR batched job program must be subtracted.

When writing the stacker selection routine, the functional characteristics publication associated with the user's system can be consulted for the time involved in executing his instructions.

Programming Considerations for 1419 or 1275 Stacker Selection

The user stacker selection routine operates in the program state with the protection key of its problem program and with I/O and external interruptions disabled. If the user's stacker selection routine fails to return to the supervisor (loops indefinitely), there is no possible recovery. If such loop occurs, the system must be re-IPLed to continue operation. Because of this possibility, it is recommended that the user thoroughly debug his stacker selection routine in a dedicated environment.

In the user stacker selection routine, no system macro instruction other than EXIT MR can be used. The routine runs with an all zero program and system mask, but the machine check interruption is enabled and a program check cancels the program.

> Note: Any modification of floating point registers without saving and restoring them may cause errant processing by any concurrent program using floating-point instructions.

When processing with the Dual Address Adapter (1419 or 1275), the user has more time for his user stacker selection routine. The only additional processing he must do within the main line is to check byte 2, bit 0, of the document buffer for stacker selection errors.

> Note: Batch numbering update is not performed with the stacker selection of auto-selected documents, and batch numbering is not available on the 1275 Optical Reader/Sorter.

MICR Document Processing

Processing begins when the user issues a GET (or READ) to the MICR device (Figure 14). The first time this GET (or READ) is executed, the supervisor engages the device for continuous reading. Each time, thereafter, the GET (or READ) merely points (through IOREG) to the next sequential buffer within each document buffer area. When a buffer for a file becomes available, the user's main line processing continues with the instruction after the GET (or READ-CHECK combination).

Each time an end-of-document condition occurs on an MICR device, the user's main line processing routine, or any routine, is interrupted by the supervisor's external interrupt routine. The external interrupt routine branches immediately to the user's stacker selection routine. After the user selects a pocket, he exits from his stacker selection routine so that the supervisor can issue the stacker selection command. At this time, the MICR device(s) should be reading document data into its (their) respective document buffer area(s). The supervisor, in priority order, passes control to the user's main line processing routine, or the routine that was interrupted, at the point of its interruption.

Thus, MICR document processing continues concurrently within the:

1. User's main line processing routine,
2. Supervisor's external interrupt routine, and
3. User's stacker selection routine.

The order for exiting from these routines is the reverse of the indicated order. Processing and monitor operations continue concurrently until the reader is disengaged, either normally or due to an error. End-of-file processing must be detected and handled by the user's main line processing routine.

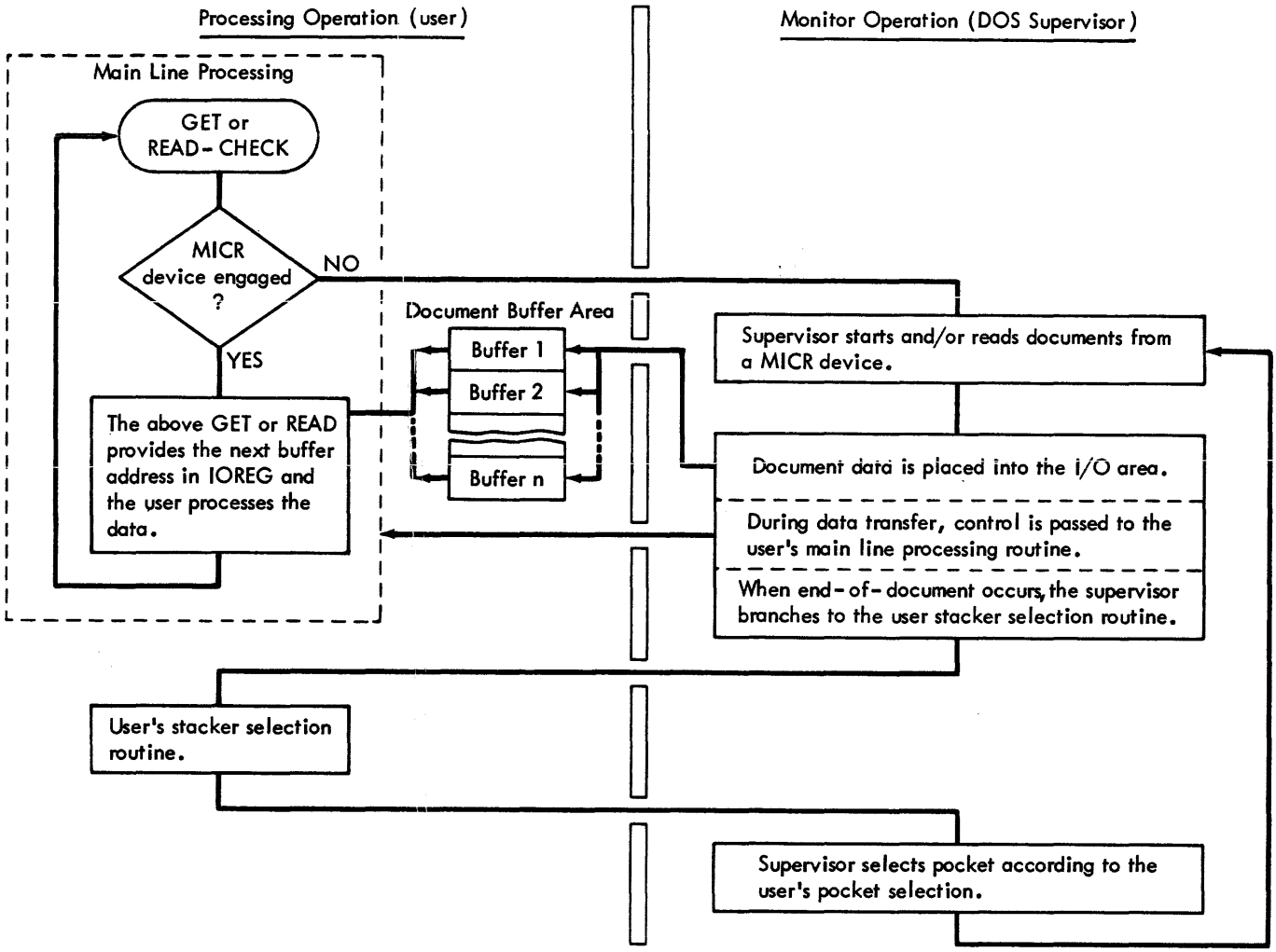


Figure 14. MICR Document Processing

The GET macro performs the functions of a READ except that it waits while the document buffer fills. Instead, the READ posts an indicator in the buffer (byte 0, bit 5) for the user to examine with the CHECK macro. If this indicator bit is ON, the buffer is not ready for processing and a branch is made to the second operand address of the CHECK macro. The user's routine at this operand address can then READ and CHECK another file for document availability. If this buffer is ready for processing, control passes to the next instruction. If a special nondata status exists, the user should analyze the conditions in his ERROPT routine and issue a READ to obtain a document unless an unrecoverable I/O error has occurred. If a second operand is not provided within the CHECK macro, control passes to the ERROPT routine address.

At least one WAITF macro must be issued between two successive executions of any one READ to the same file. The multiple WAITF is essential to the operation of the multiprogramming feature of the system. Its function is to test device operation availability or buffer processing availability. If work can be done on any specified file, control remains in the partition. If not, control passes to a lower priority partition until this partition is ready for processing.

To obtain checkpoint information about MICR files see Notes for DASD and MICR Files under the CHKPT Macro section of this publication. Examples of GET-PUT MICR document processing and multiple 1412/1419 operation (either all single or all dual) are found in the System Generation publication.

MAGNETIC READER MODULE (MRMOD)

Listed here are the user supplied operands for MRMOD. The first card contains MRMOD in the operation field and may contain a user module name in the name field. If a module name is omitted, the system name generated by IOCS of

IJU {S / D} ZZZZ

is assumed. (S = single address adapter, and D = dual address adapter).

ADDRESS=DUAL — Required only if the dual address adapter is utilized for the 1419 or 1275. If omitted, the single address adapter is assumed.

BUFFERS=nnn — A numeric value equal to the maximum number of buffers specified by a DTFMR associated with the module (minimum 12, maximum 254).

SEPASMB=YES — Required only if the module is assembled separately.

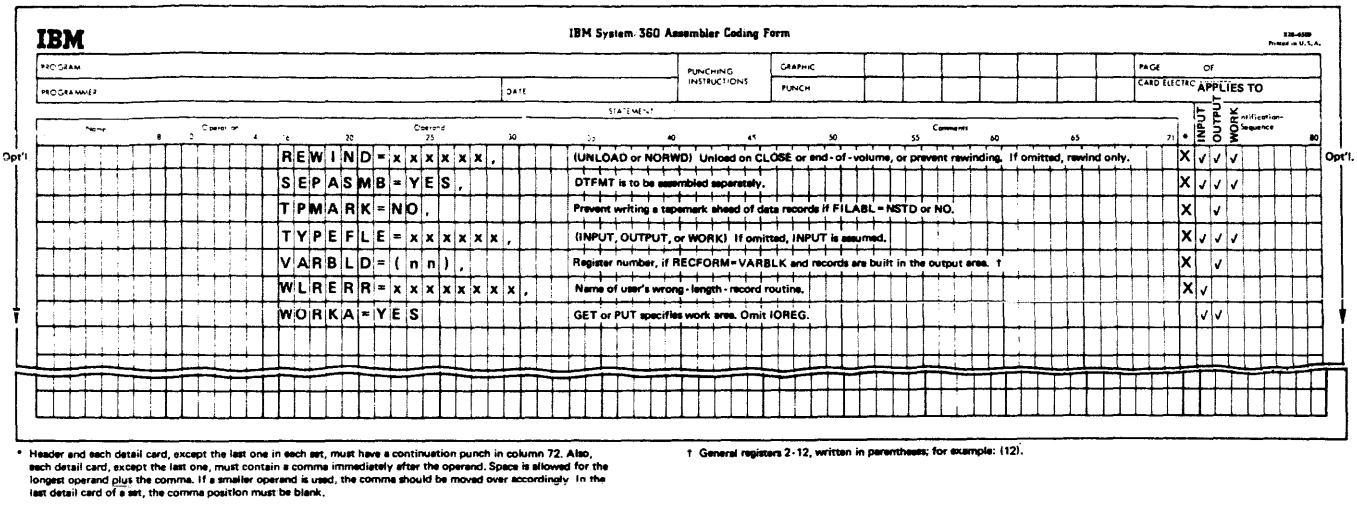
MAGNETIC TAPE FILES (DTFMT)

A DTFMT entry is included for each EBCDIC or ASCII magnetic tape input or output file that is to be processed. The DTFMT header entry is followed by a series of detail entries that describe the file (Figure 15). The detail entries generate the DTF table. Enter the symbolic name of the file in the name field and DTFMT in the operation field. The entries following the header entry may appear in any order.

IBM System/360 Assembler Coding Form

Name	Operation	Operand	Comments	INPUT	OUTPUT	WORK
xxxxxx (Req'd)	DTFMT	Name of logical file on tape. This DTF requries an MTMOD.		X	✓	✓ ✓
		BLKSIZE=nnnnn,	Length of one I/O area in bytes (maximum = 32,767).	X	✓	✓ ✓
		DEVADDR=SYSxxx,	Symbolic unit for tape drive used for this logical file.	X	✓	✓ ✓
		EOFADDR=xxxxxxxx,	Name of user's end-of-file routine.	X	✓	✓
		FILABL=xxxx,	(STD or NSTD or NO). If NSTD specified, include LABADDR. If omitted, NO is assumed.	X	✓	✓ ✓
		IOAREA1=xxxxxxxx,	Name of first I/O area.	X	✓	✓
Opt'l.		ASCII=YES,	ASCII file processing is required.	X	✓	✓ Opt'l.
		BUFOFF=nn,	Length of block prefix if ASCII = YES.	X	✓	✓
		CKPTREC=YES,	Checkpoint records are interspersed with input data records. IOCS bypasses checkpoint records.	X	✓	
		ERREXT=YES,	Additional errors and ERET are desired.	X	✓	✓ ✓
		ERROPT=xxxxxxxx,	(IGNORE or SKIP or Name of error routine) Prevent job termination on error records.	X	✓	✓ ✓
		HDRINFO=YES,	Print header label information if FILABL = STD.	X	✓	✓ ✓
		IOAREA2=xxxxxxxx,	If two I/O areas are used, name of second area.	X	✓	✓
		IOREG=(nn),	Register number. † Use only if GET or PUT does not specify work area or if two I/O areas are used. Omit WORKA.	X	✓	✓
		LABADDR=xxxxxxxx,	Name of user's label routine if FILABL = NSTD, or if FILABL = STD and user-standard labels are processed.	X	✓	✓
		LENCHK=YES,	Length check of physical records if ASCII = Yes and BUFOFF = 4.	X	✓	
		MODNAME=xxxxxxxx,	Name of MTMOD logic module for this DTF. If omitted, IOCS generates standard name.	X	✓	✓ ✓
		NOTEPNT=xxxxxx,	(YES or POINTS) YES if NOTE, POINTW, POINTR, or POINTS macro used. POINTS if only POINTS macro used.	X		✓
		RDONLY=YES,	Generate read only module. Requires a module save area for each task using the module.	X	✓	✓ ✓
		READ=xxxxxxx,	(FORWARD or BACK) If omitted, FORWARD is assumed.	X	✓	✓
		RECFORM=xxxxxx,	(FIXUNB, FIXBLK, VARUNB, VARBLK, SPNUNB, SPNBLK, or UNDEF) For work files, use FIXUNB or UNDEF. If omitted, FIXUNB is assumed.	X	✓	✓ ✓
		RECSIZE=nnnn,	If RECFORM = FIXBLK, no. of characters in record. If RECFORM = UNDEF, register number. † Not required for other records.	X	✓	✓

Figure 15. DTFMT Macro (Part 1 of 2)



* Header and each detail card, except the last one in each set, must have a continuation punch in column 72. Also, each detail card, except the last one, must contain a comma immediately after the operand. Space is allowed for the longest operand plus the comma. If a smaller operand is used, the comma should be moved over accordingly. In the last detail card of a set, the comma position must be blank.

† General registers 2-12, written in parentheses; for example: (12).

Figure 15. DTFMT Macro (Part 2 of 2)

ASCII=YES

This operand specifies that processing of ASCII tape files is required. If this operand is omitted, EBCDIC file processing is assumed. ASCII=YES is not permitted for work files.

BLKSIZE=n

Enter the length n of the I/O area. If the record format is variable or undefined, enter the length of the largest block of records. If a READ or WRITE macro specifies a length greater than n for workfiles, the record length is greater than BLKSIZE. The maximum block size is 32,767 bytes (32K minus one). The minimum size physical tape record (gap to gap) is 12 bytes. Eleven bytes or less are considered a noise record.

For output processing of spanned records, the minimum physical record length is 18 bytes. If SPNBLK or SPNUNB and TYPEFLE=OUTPUT are specified in the DTFMT and the BLKSIZE is invalid or less than 18 bytes, a new MNOTE is generated and BLKSIZE=18 is assumed.

For ASCII files, the BLKSIZE includes the length of any block prefix or padding characters present. If ASCII=YES and BLKSIZE is less than 18 bytes (for fixed-length records only) or greater than 2048 bytes, an MNOTE is generated because this length violates the limits specified by American National Standards Institute, Inc.

BUFOFF={n|0}

This is the buffer offset operand indicating the length of the block prefix. Enter the length (n) of the block prefix if processing of the block prefix is required. The contents of this field are not passed on to the user.

This operand can only be included when ASCII=YES is specified. If the BUFOFF operand is omitted, a value of 0 (no block prefix) is assumed.

The values for n can be:

Value	Condition
0-99	If TYPEFLE=INPUT
0	If TYPEFLE=OUTPUT
4	If TYPEFLE=OUTPUT and RECFORM=VARUNB or VARBLK. In this case, the program automatically inserts the physical record length in the block prefix

CKPTREC=YES

This operand is necessary if a tape input file has checkpoint records interspersed among the data records. IOCS bypasses any checkpoint records encountered. This operand must not be included when ASCII=YES.

```
DEVADDR={SYSRDR|SYSIPT|SYSPCH|SYSnnn|
        SYSLST}
```

This operand specifies the symbolic unit (SYSxxx) to be associated with the logical file. An ASSGN statement assigns an actual channel and unit number to the unit. The ASSGN card contains the same symbolic name as DEVADDR. When processing ASCII tape files, specify DEVADDR=SYSnnn only.

```
EOFADDR=name
```

This operand specifies the symbolic name of the user's end-of-file routine. IOCS automatically branches to this routine on an end-of-file condition. This entry must be specified for input and work files.

In his routine, the programmer can perform any operations required for the end of file, and he generally issues the CLOSE instruction for the file. IOCS detects end-of-file conditions in magnetic tape input by reading a tapemark and EOF when standard labels are specified. If standard labels are not specified, IOCS assumes an end-of-file condition when the tapemark is read or /* if the unit is assigned to SYSRDR or SYSIPT. The user must determine, in his routine, that this actually is the end of the file.

```
ERREXT=YES
```

This operand enables a problem program ERROPT or WLRERR routine to return to MTMOD with the ERET (error return) macro instruction. It also enables unrecoverable I/O errors occurring before data transfer takes place to be indicated to the problem program. To take full advantage of this option, the ERROPT=name operand must be specified.

```
ERROPT={IGNORE|SKIP|name}
```

This operand specifies functions to be performed for an error block.

If a parity error is detected when a block of tape records is read, the tape is backspaced and reread a specified number of times before the tape block is considered an error block. Output parity errors are considered to be an error block if they exist after IOCS attempts to forward erase and write the tape output block a specified number of times.

If ERREXT=YES is specified on output, and ERROPT=IGNORE or SKIP, the error will be ignored.

If either FILABL=STD or CKPTREC, or both, is specified, the error block is included in the block count that is taken. After this the job is automatically terminated unless this ERROPT entry is included to specify other procedures to be followed on an error condition. Either IGNORE, SKIP, or the symbolic name of an error routine can be specified in this card. The functions of these specifications are:

IGNORE

The error condition is completely ignored, and the records are made available to the user for processing.

When reading spanned records, the entire spanned record or a block of spanned records is returned to the user rather than just the one physical record in which the error occurred. On output, the physical record in which the error occurred is ignored as if it were written correctly. The remainder, if any, of the spanned record segments are written, if possible.

SKIP

No records in the error block are made available for processing. The next block is read from tape, and processing continues with the first record of that block. The error block is included in the block count.

When reading spanned records, the entire spanned record or a block of spanned records is skipped rather than just one physical record. On output, the physical record on which the error occurred is ignored as if it were written correctly. The remainder, if any, of the spanned record segments are written, if possible.

name

IOCS branches to the problem program error routine named by this parameter regardless of whether ERREXT=YES is specified. In this routine, the problem program processes or makes note of the error condition as desired.

If ERREXT is not specified, register 1 contains the address of the physical record in error. When spanned records are processed, register 1 contains the address

of the whole unblocked or blocked spanned record. Register 14 contains the return address. When processing in the ERROPT routine, the problem program references the error block, or records in the error block, by referring to the address supplied in register 1. The contents of the IOREG register or workarea (if either is specified) are variable and therefore should not be used for error processing. Also, the problem program must not issue any GET instructions for records in the error block. If any other IOCS macros (excluding ERET if ERREXT=YES) are used in this routine, the contents of registers 13 (with RDONLY) and 14 must be saved and restored after their use. At the end of the routine, the problem program must return control to IOCS by branching to the address in register 14. IOCS skips the physical record in error and it makes the next logical record available for processing in the main problem program.

A sequence error may occur if LIOCS is searching for a first segment of a logical spanned record and fails to find it. If WLRERR or ERROPT=name was specified, the error recovery procedure is the same as for wrong-length record errors. If neither WLRERR nor ERROPT=name was specified, LIOCS ignores the sequence error and searches for the next first segment.

If ERREXT is specified, register 1 contains the address of a two part parameter list containing the 4-byte DTFMT address and the 4-byte address of the physical record in error, respectively.

> Note: If ERREXT is not specified for an output file, no coding is generated and a MNOTE is issued. If an error condition occurs, the job is canceled.

Register 14 contains the return address. Processing is similar to that described when ERREXT is not specified except for addressing the physical record in error. The data transfer bit (byte 2, bit 2) of the DTF should be tested to determine if a nondata transfer error has occurred. If it is ON, the physical record in error has not been read or written. If the bit is OFF, data was transferred and the routine must address the physical record in error to determine the action to be taken. At the end of its input processing, the routine returns to LIOCS by issuing the ERET macro. If any other IOCS macros are used in this routine, the contents of register 13 (with RDONLY) and register 14 must be saved and restored after their use. At the end of the ERROPT output routine, the problem program must consider the device inoperative and must not attempt further processing on it. Another attempt to return to MTMOD results in job termination.

The ERET macro can specify one of two actions to the MTMOD logic module. The error condition can be ignored with an ERET IGNORE, or the physical record in error can be skipped to process the next physical record with an ERET SKIP. ERET RETRY is invalid and if issued results in job termination.

Figure 16 shows the DTFMT error options. This figure is divided into two parts. The upper part lists the error Conditions Specified by the User in the DTF, and the lower part shows the Actions Resulting From these specifications when an error occurs. Refer to the shaded column in the figure. The user has specified WLRERR=name and also the RETRY option in his ERET macro. If the error occurring is either a wrong length record or other than a wrong length record, the job is terminated. Refer to the remaining columns of the figure for other specifications and their resulting actions.

The job is automatically terminated if a parity error still exists after IOCS attempts to write a tape output block a specified number of times. This includes erasing forward.

This entry applies to wrong length records if the entry WLRERR is not included. If both ERROPT and WLRERR are omitted and wrong-length records occur, IOCS assumes the IGNORE option.

> Note: For ASCII tape files, the pointer to the block in error points to the first logical record following the block prefix.

```
|FILABL={STD|NO|NSTD}                    |
```

The parameter STD indicates that standard labels will be processed. Enter NO if no labels are contained on the file. If nonstandard labels are contained on the file, enter NSTD. The user must furnish a routine to check or create the nonstandard labels by using his own I/O area and EXCP to read or write the labels. The entry point of this routine is the operand of LABADDR.

The specification FILABL=NSTD is not permitted for ASCII files (that is, when ASCII=YES). Labels and tape data are assumed to be in the same mode.

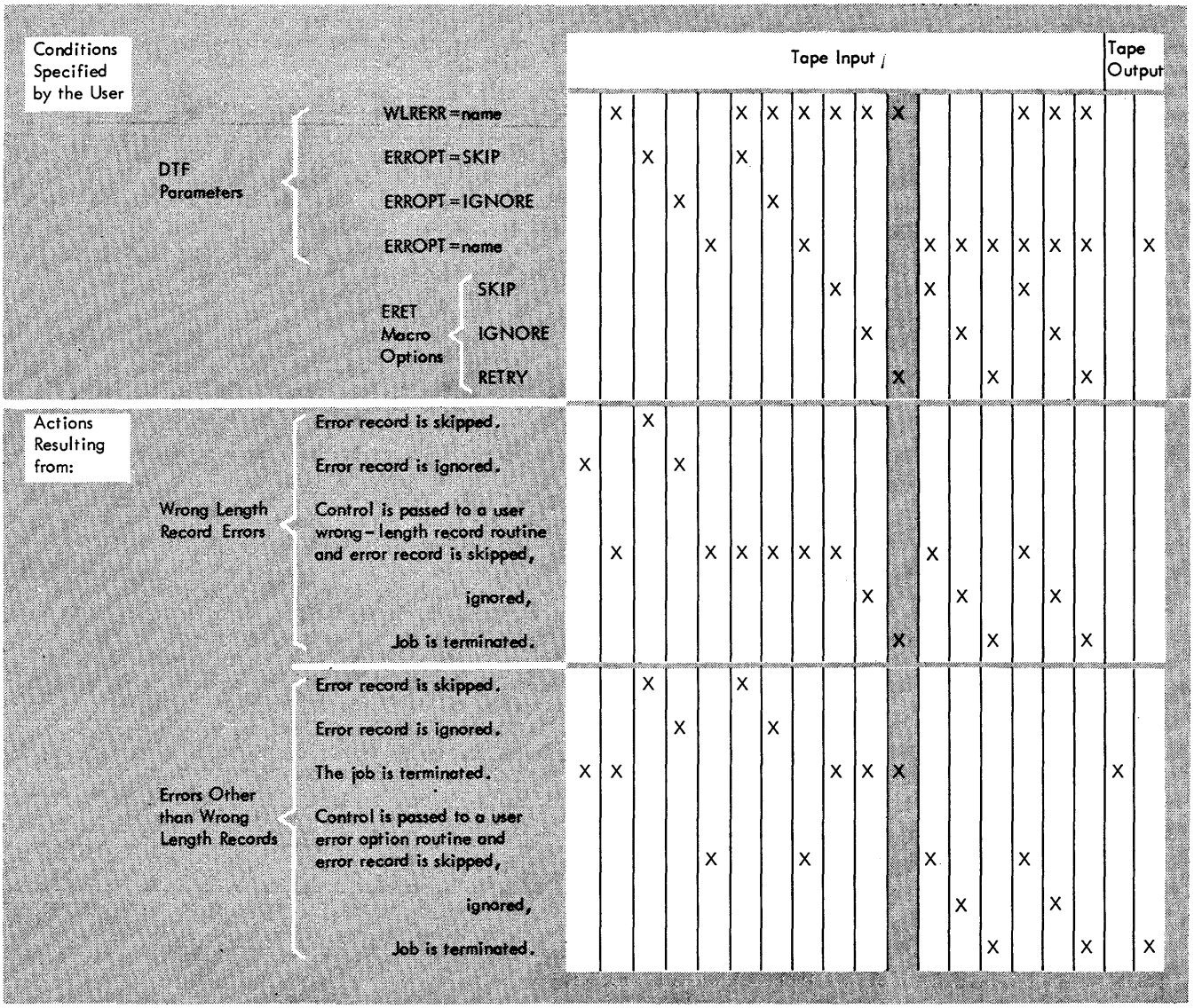


Figure 16. DTFMT Error Options

HDRINFO=YES

This operand, if specified with FILABL=STD, causes IOCS to print standard header label information (fields 3-10) on SYSLOG each time a standard label file is opened. It also prints the filename, symbolic unit, and device address each time an end-of-volume condition is detected. Both FILABL=STD and HDRINFO=YES must be specified for header label information to be printed.

IOAREA1=name

This operand specifies the I/O area. Enter an address expression (name), which specifies the I/O area. When variable-length records are processed, the size of the I/O area must include four bytes for the block size. This operand does not apply to work files.

IOAREA2=name

This operand specifies a second I/O area. Enter an address expression (name), which specifies the I/O area. When variable-length records are processed, the size of the I/O area must include four bytes for the blocksize. This operand does not apply to work files.

IOREG=(r)

This operand specifies the register in which IOCS places the address of the logical record that is available for processing under the following conditions:

- If two input or output areas are used.
- If blocked input or output records are processed in the I/O area.
- If variable unblocked records are read.
- If undefined records are read backwards.
- If neither BUFOFF=0 nor WORKA=YES is specified for ASCII files.

For output files, IOCS places in the specified register the address of the area where the user can build a record. Any register (2-12) may be specified.

Note: This operand cannot be used if WORKA=YES.

LABADDR=Name

Enter the symbolic name of the user routine to process user-standard or nonstandard labels. See sections Writing and Checking User Standard Labels and Writing and Checking Nonstandard Labels.

For ASCII files, this operand may only be used for writing and checking American National Standards Institute, Inc. user standard labels. The user must process these labels in EBCDIC. User nonstandard labels are not permitted.

LENCHK=YES

This operand applies only to ASCII tape input files if BUFOFF=4 and RECFORM=VARUNB or VARBLK. It must be included if the block length (specified in the block prefix) is to be checked against the physical record length. If an inequality is detected, the action taken is the same as described under the WLRERR operand, but the WLR bit (byte 5, bit 1) in the DTF is not set.

MODNAME=name

This operand specifies the name of the logic module used with the DTF table to process the file. If the logic module was assembled with the program, the MODNAME in the DTF macro instruction must specify the same name as the MTMOD macro instruction. If this entry is omitted, standard names are generated for calling the logic module. If two DTF macro instructions call for different functions that can be handled by a single module, only one module is called. For example, if one DTF specifies READ=FORWARD and another specifies READ=BACK, only one logic module capable of handling both functions is called.

NOTEPNT={POINTS|YES}

If the parameter YES is specified, the NOTE, POINTW, POINTR, or POINTS macro instructions are issued to a tape work file. If POINTS is specified, only POINTS macro instructions can be issued tc tape work files. The NOTEPNT operand must not be specified for ASCII tape files because work files are not supported.

RDONLY=YES

This operand is specified if the DTF is used with a read only module. Each time a read only module is entered, register 13 must contain the address of a 72-byte doubleword aligned save area. Each DTF should have its own uniquely defined save area and each time an imperative macro (except OPEN(R), LBRET, SETL, or SETFL) is issued using a particular DTF, register 13 must contain the address of the save area associated with that DTF. The fact that the save areas are unique for each task makes the module reentrant (that is,

capable of being used concurrently by several tasks). For more information see Shared Modules and Files.

If an ERROPT or WLRERR routine issues I/O macro instructions using the same read only module that caused control to pass to either error routine, the problem program must provide another save area. One save area is used for the normal I/O operations and the second for I/O operations in the ERROPT or WLRERR routine. Before returning to the module that entered the ERROPT routine, register 13 must be set to the save area address originally specified for that DTF.

If the operand is omitted, the module generated is not reenterable and no save area is required.

```
|READ={FORWARD|BACK}                    |
```

This operand specifies the direction the tape is read. READ=FORWARD may be omitted. If READ=BACK is specified and a smaller wrong length record is encountered, the record is read into the I/O area right justified.

```
|RECFORM={FIXUNB|FIXBLK|VARUNB|VARBLK|  |
|        SPNBLK|SPNUNB|UNDEF}           |
```

This operand specifies the type of EBCDIC or ASCII records (fixed or variable length, blocked or unblocked, or undefined) in the input or output file. One of the following may be entered immediately following the = sign.

- FIXUNB — For fixed-length unblocked records
- FIXBLK — For fixed-length blocked records
- VARUNB — For variable-length unblocked records
- VARBLK — For variable-length blocked records
- SPNBLK — For spanned variable-length blocked records (EBCDIC only)
- SPNUNB — For spanned variable-length unblocked records (EBCDIC only)
- UNDEF — For undefined records.

If the record format is fixed-length unblocked, this entry may be omitted. Work files may use only FIXUNB or UNDEF.

```
|RECSIZE={n|(r)}                        |
```

For fixed-length blocked records, this operand is required. It specifies the number of characters, n, in each record.

When processing spanned records, the user must specify RECSIZE=(r) where r is a register.

For undefined records, this entry is required for output files and optional for input files. It specifies a general register (2-12) that contains the length of the record. On output, the user must load the length of each record into the register before he issues a PUT macro. Spanned-record output requires a minimum record length of 18 bytes. A physical record less than 18 bytes is padded with binary zeros to complete the 18-byte requirement. This applies to both blocked and unblocked records. If specified for input, IOCS provides the length of the record transferred to main storage.

```
|REWIND={UNLOAD|NORWD}                  |
```

If this specification is not included, tape files are automatically rewound to load point, but not unloaded, on an OPEN or CLOSE instruction or on an end-of-volume condition. If other operations are desired for a tape input or output file, this entry specifies:

- UNLOAD — To rewind the tape on OPEN or to rewind and unload on a CLOSE or end-of-volume condition.
- NORWD — To prevent rewinding the tape at any time. This option positions the read/write head between the two tapemarks on the EOF condition.

```
|SEPASMB=YES                            |
```

This operand must be included if the DTF is assembled separately. This causes a CATALR card with the filename to be punched ahead of the object deck and defines the filename as an entry point in the assembly.

```
|TPMARK=NO                                |
```

This operand is included if the user does not want a tapemark written as the first record on a tape output file when no labels are specified. This entry is also included if no tapemark is to be written following nonstandard labels. If this entry is omitted for a tape output file, a tapemark will precede the first record if no labels are specified. If this entry is omitted a tapemark is written following nonstandard labels. For unlabeled tapes in ASCII, TPMARK=NO is the default. This parameter is ignored if FILABL=STD.

```
|TYPEFLE={INPUT|OUTPUT|WORK}              |
```

Use this operand to indicate whether the file is an input or output file. If INPUT is specified, the GET macro is used. If OUTPUT is specified, the PUT macro is used. If WORK is specified, the READ/WRITE, NOTE/POINT, and CHECK macros are used. See Work Files for DTFMT and DTFSD (2321). The specification of WORK in this operand is not permitted for ASCII files.

```
|VARBLD=(r)                               |
```

Whenever variable-length blocked records are built directly in the output area (no work area specified), this entry must be included. It specifies the number (r) of a general-purpose register that always contains the length of the available space remaining in the output area. Any register (2-12) may be specified.

IOCS calculates the space still available in the output area, and supplies it to the programmer in the VARBLD register, only after the PUT instruction is issued for a variable-length record. The programmer then compares the length of his next variable-length record with the available space to determine if the record will fit in the remaining area. This check must be made before the record is built. If the record does not fit, the programmer issues a TRUNC instruction to transfer the completed block of records to the tape file. Then, the current record is built as the first record of the next block.

```
|WLRERR=name                              |
```

This operand applies only to tape input files. It specifies the symbolic name of a problem program routine to receive control if a wrong-length record is read.

If ERREXT is not specified, the address of the physical record in error is supplied by IOCS in register 1. If ERREXT is specified, register 1 contains the address of a two-part parameter list. The first four bytes of the list are the DTF address and the second four bytes are the address of the physical record in error. If the block read is less than the BLKSIZE parameter, the first two bytes of the DTF contain the number of bytes left to be read (residual count). Therefore, the size of the actual block is equal to the block size minus the residual count. If the block to be read is larger than the BLKSIZE parameter, the residual count is zero, and there is no way to compute the record size. The number of bytes transferred is equal to the BLKSIZE parameter, and the remainder of the original block is truncated.

The problem program WLRERR routine can perform any processing desirable for wrong length records. However, it must not issue GET macro instructions to this file. If the routine issues any other IOCS macros (excluding ERET if ERREXT=YES) the contents of registers 13 (with RDONLY) and 14 must be saved before and restored after their use. At the end of the routine, control must be returned to IOCS by branching to the address in register 14, or, if ERREXT is specified, the ERET IGNORE or SKIP option can be taken.

When fixed-length unblocked records are specified (RECFORM=FIXUNB), a wrong-length record error condition is given when the length of the record read is not equal to that specified in the BLKSIZE parameter. For EBCDIC fixed-length blocked records, record length is considered incorrect if the physical tape record (gap to gap) that is read is not a multiple of the logical-record length (specified in DTF RECSIZE), up to the maximum length of the block (specified in DTFMT BLKSIZE). This permits the reading of short blocks of logical records without a wrong-length record indication.

For EBCDIC variable-length records blocked and unblocked, record length is considered incorrect if the length of the tape record is not the same as the block length specified in the 4-byte block-length field. The residual count can be obtained by addressing the halfword at filename+98.

For ASCII variable length records, blocked and unblocked, a check on the physical record length is performed if LENCHK=YES is specified. The physical record length is considered incorrect if the tape record is not the same as the block length that is specified in the 4-byte block prefix. In this case, the WLR bit (byte 5, bit 1) in the DTF is set off.

The WLRERR option is taken for undefined records if the record read is greater than the size specified by the BLKSIZE parameter.

If the WLRERR entry is omitted but a wrong-length record is detected by IOCS, one of the following condition results:

- If the ERROPT entry is included for this file, the wrong-length record is treated as a error block, and handled according to the user's specifications for an error (IGNORE, SKIP, or name-of-error routine).
- If the ERROPT entry is not included, IOCS assumes the IGNORE option of ERROPT.

```
|WORKA=YES                                |
```

If I/O records are processed in work areas instead of the I/O area, specify YES with this operand. The user must set up the work area in main storage. The address expression of the work area (or general register containing the address) must be specified in each GET or PUT. Omit IOREG if this operand is included. WORKA=YES is required for spanned record processing.

MAGNETIC TAPE MODULE (MTMOD)

Listed here are the user-supplied operands for MTMOD. The first card contains MTMOD in the operation field and may contain a user module name in the name field.

```
|ASCII=YES                                |
```

Include the operand if processing ASCII input or output files is required. This entry is not permitted for workfiles. If omitted, EBCDIC file processing is assumed.

```
|CKPTREC=YES                              |
```

Include this operand if tape input files processed by the module contain checkpoint records interspersed among the data records. The module also processes files that do not have checkpoint records; that is, those whose DTFs do not specify CKPTREC=YES.

This entry is not needed for work files, and is not valid for ASCII files.

```
|ERREXT=YES                               |
```

Include this operand if additional I/O errors are to be indicated and/or the ERET macro is used with this DTF and module. ERROPT=YES should be specified in this module for workfiles, but is not needed for input or output files.

```
|ERROPT=YES                               |
```

Include this operand if the module is to handle any of the error options for an error block. Logic is generated to handle any of the three options (IGNCRE, SKIP, or name). The module processes any files in which the ERROPT operand is not specified in the DTF. This entry is needed for work files, but it is not needed for input or output files.

```
|NOTEPNT={YES|POINTS}                     |
```

Include this operand if NOTE/POINT logic is used with the module. If YES, the module processes any NOTE, POINTR, POINTW, or POINTS macro instruction. If POINTS is specified, only the POINTS macro instruction is processed.

Modules specifying either one of the two options also process work files for which the NOTE/POINT operand is not specified. Modules specifying YES also process work files specifying only POINTS. This entry does not apply to input or output files. The NOTEPNT operand is not used for ASCII files.

```
RDONLY=YES
```

This operand generates a read only module. RDONLY=YES must be specified in the DTF. For the programming requirements of this operand, see the DTF RDONLY operand.

```
READ={FORWARD|BACK}
```

This operand generates a module that reads tape files forward or backward. If forward is specified, only logic to read tape forward is generated. Any DTF used with the module may not specify BACK in the READ parameter statement.

If the parameter is BACK, logic to read tape both forward and backward is generated, and any DTF used with the module may specify either FORWARD or BACK as its READ parameter. READ=BACK does not handle multivolume files.

This entry is not needed for work files.

```
RECFORM={FIXUNB|FIXBLK|VARUNB|VARBLK|
        SPNBLK|SPNUNB|UNDEF}

Note: FIXUNB and FIXBLK use identical
table formats and logic modules.
```

This operand generates an input/output module that processes either EBCDIC or ASCII fixed-length, variable-length or undefined records. If either FIXUNB or FIXBLK is specified, a logic module is generated that allows processing of both fixed-length record types. Similarly, a logic module is generated that allows processing of both types of variable and spanned records. ASCII files are not permitted in spanned record format. If UNDEF is specified, a logic module for processing undefined record types is generated. Any DTF used with the module must specify the same record format type as the module. For example, if the module has the entry RECFORM=FIXUNB, the DTF may have either the entry RECFORM=FIXUNB or RECFORM=FIXBLK. This entry is not needed for work files.

```
SEPASMB=YES
```

This operand must be included if the logic module is assembled separately. This causes a CATALR card with the module name (standard or user) to be punched ahead of the object deck.

```
TYPEFLE={INPUT|OUTPUT|WORK}
```

This operand generates a logic module that processes either GET/PUT macro instructions or READ/WRITE, NOTE/POINT and CHECK macro instructions for work files. If the parameter of the operand specifies WORK, logic to process work files is generated. Otherwise, a module to handle both input and output file types is assumed. Only DTFs for work files may be used with work file modules. Only DTFs for input or output files may be used with an input/output module. Work files are not supported with ASCII tapes.

Note: INPUT and OUTPUT have the same table format and logic modules.

```
WORKA=YES
```

This operand is to be included if records are to be processed in work areas instead of I/O areas for the GET/PUT macros. This operand must be included if spanned records are processed. The module also processes files that do not use a work area. This entry is not needed for work files.

Recommended Module Name for MTMOD

Each name begins with a 3-character prefix (IJF) and consists of a 5-character field corresponding to the options permitted in module generation.

In MTMOD there are two module classes: the module class for handling GET/PUT functions and the module class for handling READ/WRITE, NOTE/POINT, and CHECK functions (work files). Modules handling fixed length (F,X) and undefined (U,N) records are mutually exclusive of each other and of all forms of the variable length module (V,R,S).

Name list for GET/PUT type modules:

MTMOD name = IJFabcde

a = F RECFORM=FIXUNB (or FIXBLK)
(EBCDIC mode)

= X RECFORM=FIXUNB (or FIXBLK)
(ASCII mode)

= V RECFORM=VARUNB (or VARBLK)
(EBCDIC mode)

= R RECFORM=VARUNB (or VARBLK)
(ASCII mode)

= S RECFORM=SPNUNB (or SPNBLK)
(spanned records)

= U RECFORM=UNDEF (EBCDIC mode)

= N RECFORM=UNDEF (ASCII mode)

b = B READ=BACK
= Z READ=FORWARD, or if READ is not specified

c = C CKPTREC=YES
= Z CKPTREC=YES is not specified

d = W WORKA=YES
= Z WORKA=YES is not specified

e = M ERREXT=YES and RDONLY=YES
= N ERREXT=YES
= Y RDONLY=YES
= Z ERREXT and RDONLY not specified

Name list for work file type modules (TYPEFLE=WORK):

MTMOD name = IJFabcde

a = W always

b = E ERROPT=YES
= Z ERROPT is not specified

c = N NOTEPNT=YES
= S NOTEPNT=POINTS
= Z NOTEPNT is not specified

d = Z always

e = M ERREXT=YES and RDONLY=YES
= N ERREXT=YES
= Y RDONLY=YES
= Z ERREXT and RDONLY not specified

Subset/Superset MTMOD Names

The following charts illustrate the subsetting and supersetting allowed for MTMOD names. Four of the GET/PUT parameters allow subsetting. For example, the module name IJFFBCWZ is a superset of IJFFBZWZ specifying fixed-length records. See Subset/Superset: (Module Names).

For GET/PUT Type Modules:

			*	+	+	+	+
	I	J	F	F	B	C	W	M
				N	Z	Z	Z	Y
				R				+
				U				N
				X				Z
				+				
				S				
				V				

\+ Subsetting/supersetting permitted.
* No subsetting/supersetting permitted.

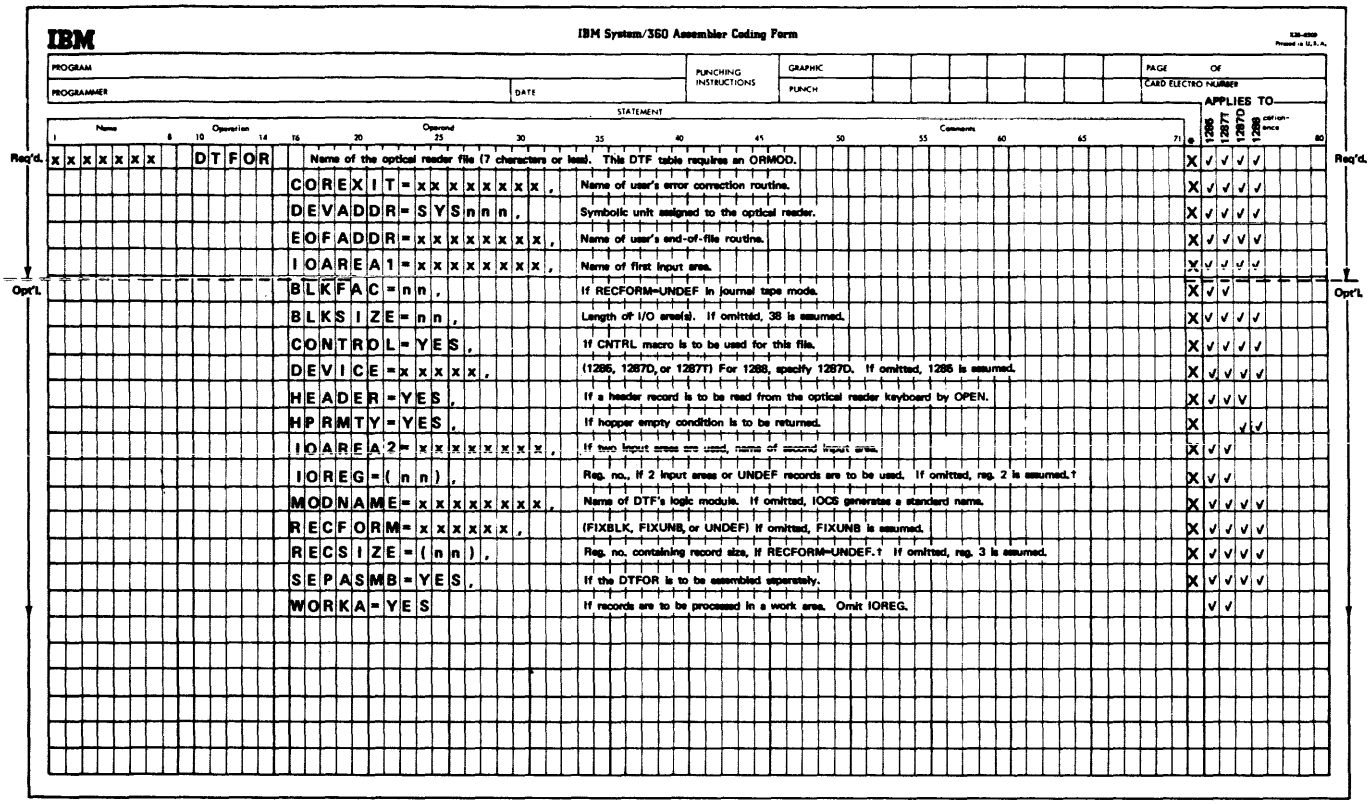
For Workfile Type Modules:

				+	+		+
I	J	F	W	E	N	Z	M
				Z	S		Y
					Z		+
							N
							Z

\+ Subsetting/supersetting permitted.
* No subsetting/supersetting permitted.

OPTICAL READER FILE (DTFOR)

DTFOR is used to define an input file to be processed on an IBM 1285, 1287 Optical Reader or 1288 Optical Page Reader. Enter the symbolic name of the file in the name field and DTFOR in the operation field. The operands for DTFOR follow and are illustrated in Figure 17.



* Header and each detail card, except the last one in each set, must have a continuation punch in column 72. Also, each detail card, except the last one, must contain a comma immediately after the operand. Space is allowed for the longest operand plus the comma. If a smaller operand is used, the comma should be moved over accordingly. In the last detail card of a set, the comma position must be blank.

† General registers 2-12, written with parentheses; for example: (12).

Figure 17. DTFOR Macro

```
|BLKFAC=n                                                    |
```

Undefined journal tape records are processed with greater throughput speeds when this operand is included. This is accomplished by reading groups of lines as blocked records. When undefined records are processed, BLKFAC specifies the blocking factor (n) that determines the number of lines read (through CCW chaining) as a block of data by one physical read. Deblocking is accomplished automatically by IOCS when the GET macro is used. The BLKFAC parameter is not used with RECFORM=FIXBLK, because the blocking factor is determined from the BLKSIZE and RECSIZE parameters. If the operand is included for FIXBLK, FIXUNB, or document processing, the operand is noted (MNOTE) and ignored.

```
|BLKSIZE={n|38}                                              |
```

This operand indicates the size of the input area specified by IOAREA1. For journal tape processing, BLKSIZE specifies the maximum number (n) of characters that may be transferred to the area at any one time.

When undefined journal tape records are read, the area must be large enough to accommodate the longest record to be read if the BLKFAC parameter is not specified. If the BLKFAC parameter is specified, the BLKSIZE value must be determined by multiplying the maximum length that must be accommodated for an undefined record by the blocking factor desired. A BLKSIZE value smaller than this results in truncated data.

If two input areas are used for journal tape processing (IOAREA1 and IOAREA2), the size specified in this entry is the size of each I/O area.

```
CONTROL=YES
```

This entry must be included if a CNTRL macro instruction is issued for a file. A control command issues orders to the optical reader to perform nondata operations such as line marking, stacker selecting, document incrementing, etc.

```
COREXIT=name
```

COREXIT provides an exit to the user's error correction routine for the IBM 1285, 1287 Optical Reader, or 1288 Optical Page Reader. After GET, WAITF, or CNTRL macro (to increment or eject and/or stacker select a document) is executed, an error condition results in an error correction routine with an indication provided in filename+80. Filename+80 contains the following hexadecimal bits indicating the conditions that occurred during the last line or field read. Filename+80 should also be tested after issuing the optical reader macros DSPLY, RESCN, RDLNE, CNTRL READKB, and CNTRL MARK. More than one error condition may be present.

X'20' For the 1288, reading in unformatted mode, the end-of-page (EOP) condition has been detected. Normally, on an EOP indication, the problem program ejects and stacker selects the document.

Filename+80 should also be tested after issuing the optical reader macros CNTRL ESD, CNTRL SSD, CNTRL EJD in user's COREXIT routine. These should only be tested for nonrecovery (X'10') and (X'20') late stacker select.

For the 1287, a stacker select was given after the allotted elapsed time and the document was put in the reject pocket (1287 only).

X'01' A data check has occurred. Five read attempts for journal tape processing or three read attempts for document processing were made.

X'02' The operator corrected one or more characters from the keyboard (1285 or 1287T) or a hopper empty condition (see HPRMTY=YES operand) has occurred (1287D).

X'04' A wrong-length record condition has occurred (five read attempts were made for journal tapes or three for documents). Not applicable for undefined records.

X'08' An equipment check resulted in an incomplete read (ten read attempts were made for journal tapes or three for documents).

If an equipment check occurs on the first character in the record, when processing undefined journal tape records, the RECSIZE register contains zero, and the IOREG (if used) points to the rightmost position of the record in the I/O area. The user should test the RECSIZE register before moving records from the work area or the I/O area.

X'10'. A nonrecovery error occurred.

X'40' The 1287D scanner was unable to locate the reference mark (ten read attempts were made for journal tapes or three for documents).

Filename+80 can be interrogated by the user to determine the reason for entering the error correction routine. Choice of action in the user's error correction routine is determined by the particular application.

If the user issues I/O macro instructions to any device other than the 1285, 1287 and/or 1288 in the COREXIT routine, he must save registers 0, 1, 14, and 15 upon entering the routine, and restore these registers before exiting. Also, if I/O macro instructions (other than the GET, WAITF, and/or READ, which cannot be used in COREXIT) are issued to the 1285, 1287, and/or 1288 in this routine, the user must also save, and later restore registers 14 and 15 before exiting. All exits from COREXIT should be to the address specified in register 14. This provides a return to the point from which the branch to COREXIT occurred. If the command chain bit is ON in the READ CCW for which the error occurred, IOCS completes the chain upon return from the COREXIT routine.

Note: Do not issue a GET, READ, or WAITF macro to the 1285, 1287, or 1288 in the error correction routine. Do not process records in the error correction routine. The record that caused the exit to the error routine is available for processing upon return to the user's mainline program. Any processing included in the error routine would be duplicated after return to the mainline program.

When processing journal tapes, a nonrecovery error (torn tape, tape jam,

etc) normally requires that the tape be completely reprocessed.

Restriction: In this case, the user routine must not branch to the address in register 14 from the COREXIT routine or a program loop will occur.

Following a nonrecovery error:

- The optical reader file must be closed:
- The condition causing the nonrecovery must be cleared:
- The file must be reopened before processing can continue.

If a nonrecoverable error occurs while processing documents (indicating that a jam occurred during a document incrementation operation, or a scanner control failure has occurred, or an end-of-page condition, etc), the document should be removed either manually or by nonprocess runout.

Restriction: In such cases, the user program should branch to read the next document. If the 1287 or 1288 scanner is unable to locate the document reference mark, the document cannot be processed. In this case, the document must be ejected and stacker selected before attempting to read the following document or a program loop will result. In any case, the user routine must not branch to the address in register 14 from the COREXIT routine. If a nonrecoverable error occurs, the user routine should ignore any output resulting from the document.

Eight binary counters are used to accumulate totals of certain 1285, 1287, and 1288 error conditions. These counters each occupy four bytes, starting at filename+48. Filename is the name specified in the DTF header entry. The error counters are:

Counter and Address	Contents
1 filename+48	Equipment check (see Note).
2 filename+52	Equipment check uncorrectable after ten attempts for journal tapes or three for documents (see Note).
3 filename+56	Wrong-length records (not applicable for undefined records).
4 filename+60	Wrong-length records uncorrectable after five read attempts for journal tapes or three for documents (not applicable for undefined records).
5 filename+64	Keyboard corrections (journal tape only).
6 filename+68	Journal tape lines (including retried lines) or document fields (including retried fields) in which data checks are present.
7 filename+72	Lines marked (journal tape only).
8 filename+76	Count of total lines read from journal tape or the number of CCW chains executed during document processing.

Note: Counters 1 and 2 apply to equipment checks that result from incomplete reads or from the inability of the 1287 or 1288 scanner to locate a reference mark (when processing documents only).

All the previous counters contain binary zeros at the start of each job step and are never cleared. The user may list the contents of these counters for analysis at end of file, or at end of job, or he may ignore the counters. Binary contents of the counters should be converted to a printable format.

```
DEVADDR=SYSnnn
```

This operand specifies the symbolic unit (SYSnnn) to be associated with the logical file. The symbolic unit represents an actual I/O device address used in the job control ASSGN statement to assign the actual I/O device address to this file.

```
|DEVICE={1285|1287D|1287T}
```

This operand must be included to specify the I/O device associated with this logical file. One of the following specifications must be entered immediately after the = sign:

1285 For a 1285 journal tape file

1287D For a 1287 or 1288 document file

1287T For a 1287 journal tape file

From this specification, IOCS sets up the device-dependent routines for this file. For document processing on the IBM 1287 Optical Reader or IBM 1288 Optical Page Reader, the user codes CCWs.

```
|EOFADDR=name
```

This operand specifies the symbolic name of the user's end-of-file routine. IOCS automatically branches to this routine on an end-of-file condition.

When reading data from documents, you can recognize an end-of-file condition by pressing the end-of-file key on the console when the hopper is empty. When processing journal tapes on a 1285 or 1287, you can detect an end-of-file by pressing the end-of-file key after the end of the tape is sensed.

When IOCS detects an end-of-file condition, it branches to the user's routine specified by EOFADDR. The user must determine if the current roll is the last roll to be processed when handling journal tapes. For 1285, do this by keying in header information at the beginning of each roll. This header information can then be interrogated to determine whether it is the last roll. Regardless of the situation, the tape file must be closed for each roll within the user's EOF routine. If the current roll is not the last, OPEN must be issued. The OPEN macro instruction allows header (identifying) information to be entered at the reader keyboard and read by the processor when using logical IOCS.

The same procedure can be used for 1287 processing of multiple journal tape rolls, as well as the method described under OPEN(R) Macros.

```
|HEADER=YES
```

This operand is required if the operator is to key in header (identifying) information from the 1285 or 1287 keyboard. The OPEN routine reads the header information only when this entry is present. If the entry is not included, OPEN assumes no header information is to be read. The header record size can be as large as the BLKSIZE entry, and it is read into the high-order positions or IOAREA1. This operand cannot be used for 1288 files.

```
|HPRMTY=YES
```

This operand is included if the hopper empty indication is to be passed to the user. This condition occurs when a READ is issued and no document is present, and is recognized at WAITF time. When a hopper empty condition is detected, the user's COREXIT routine is entered with the condition indicated as X'02' in filename+80.

This operand should be used when processing documents in the time dependent mode of operation, which allows complete overlapping of processing with reading. (See Method 2 under Programming the 1287 in the IBM 1287 Optical Reader Component Description and Operating Procedures publication listed in the IBM S/360 and S/370 Bibliography.) With this method of processing, the HPRMTY parameter, allows the user to check for a hopper empty condition in his COREXIT routine. He can then stacker select properly the previously ejected document before return from COREXIT (via register 14).

```
|IOAREA1=name
```

This operand is included to specify the symbolic name of the input area used by the file. When opening a file and before each journal tape input operation to this area, the designated area is set to binary zeros and the input routines then transfer records to this area. For document processing, the area is cleared only when the file is opened.

```
|IOAREA2=name                                |
```

A second input area can be allotted only for a journal tape file. This permits an overlap of data transfer and processing operations and the IOAREA2 entry must be included. The specified second I/O area is set to binary zeros before each input operation to this area occurs.

```
|IOREG={(r)|(2)}                            |
```

This operand specifies a general purpose register (r) that the input routines use to indicate the beginning of records for journal tape file only. Any register number 2-12 may be specified, but if the entry is omitted, register 2 is assumed. The same register may be specified in the IOREG entry for two or more files in the same program, if desired. In this case, the problem program may need to store the address supplied by IOCS for each record. Whenever this entry is included for a file, the DTFOR entry WORKA must be omitted, and the GET instruction must not specify a work area.

A read by an optical reader is accomplished by a backward scan. This places the rightmost character in the record in the rightmost position in the I/O area and subsequent characters in sequence from right to left. The register defined by IOREG indicates to the user the leftmost position of the record.

```
|MODNAME=name                                |
```

This operand is used if a nonstandard, or a more inclusive module is referenced. It specifies a user-named I/O module.

```
|RECFORM={FIXUNB|FIXBLK|UNDEF}              |
```

This operand specifies the type of records in an optical reader file. One of the following specifications may be entered immediately after the = sign:

FIXUNB For fixed-length unblocked records

FIXBLK For fixed-blocked records in journal tape mode

UNDEF For undefined records

```
|RECSIZE={n|(r)|(3)}                        |
```

For fixed-length unblocked records, this operand should be omitted and no register is assumed.

For fixed-length blocked records (journal tape mode), this entry must be included to specify the number n of characters in an individual record. The input routines use this factor to deblock records, and to check the length of input records. If this operand is omitted, an MNOTE is flagged in the macro assembly and fixed-length unblocked records are assumed.

For undefined journal tape records, this entry specifies the number (r) of the general-purpose register in which IOCS provides the length of each input record. For undefined document records, RECSIZE contains only the length of the last field of a document read by the user-supplied channel command word chain. Any register 2-12 may be specified, but if the entry is omitted, register 3 is assumed.

> Note: When processing undefined records in document mode, the user gains complete usage of the register normally used in the RECSIZE parameter. He can do this by ensuring that the suppress-length-indication (SLI) flag is always ON when processing undefined records.

```
|SEPASMB=YES                                 |
```

This operand must be included if the DTF is assembled separately. This causes a CATALR card with the filename to be punched before the object deck, and defines the filename as an entry point in the assembly.

```
|WORKA=YES                                   |
```

Input records (journal tape only) can be processed in a work area instead of the input area. If this is planned, the operand WORKA=YES must be included, and the programmer must set up the work area in main storage. The symbolic name of the work area (or a general register containing the address of the work area) must be specified in each GET macro. When GET is issued, IOCS left-justifies the record in the specified work area. Whenever this entry is included for a file, the DTFOR entry IOREG must be omitted.

IBM 1288 Optical Page Reader Programming Considerations

After an IBM 1288 Optical Reader file is defined, the OPEN(R) macro makes it available for input. Processing is then accomplished by the CNTRL, READ, RESCN, and WAITF macro instructions. When processing is completed, the CLOSE(R) macro deactivates the file. 1288 processing adheres closely to the macros and DTF specifications used for 1287 document processing.

OPTICAL READER MODULE (ORMCD)

The following list contains the parameters for a user-coded I/O module (ORMOD). The first card contains the module name in the name field and ORMOD in the operation field.

```
|BLKFAC=YES                              |
```

Include this operand if RECFORM=UNDEF and groups of undefined journal tape records are to be processed as blocks of data. (See Optical Reader File (DTFOR): BLKFAC=n.) The DTFOR used with this module must also include RECFORM=UNDEF and BLKFAC=n.

```
|CONTROL=YES                             |
```

Include this operand if CNTRL macro instructions are to be used with the associated DTFs. The module also processes files that do not use the CNTRL macro instruction.

```
|DEVICE={1285|1287D|1287T}               |
```

This operand must be specified to indicate that either the IBM 1285, 1287 or 1288 (document mode), or 1287 (tape mode) is used as the input device.

```
|IOAREA2=YES                             |
```

Include this operand (journal tape only) if a second I/O area is used. The DTFOR used with this module must also include the IOAREA2 parameter.

```
|RECFORM={FIXUNB|FIXBLK|UNDEF}          |
```

This operand generates a module that processes the specified record format. The DTFOR used with this module must also include the appropriate operand in the RECFORM parameter.

```
|SEPASMB=YES                             |
```

This operand must be included if the module is assembled separately from the DTF(s). This entry causes a CATALR card to be punched preceding the module.

```
|WORKA=YES                               |
```

Include this operand (journal tape only) if records are to be processed in work area(s) instead of I/O areas. The DTFOR used with this module must include the appropriate operand in the WORKA parameter.

Recommended Module Name for ORMOD

Each name begins with a 3-character prefix (IJM) followed by a 5-character field corresponding to the options permitted in the generation of the module.

ORMOD name = IJMabcde

a = F RECFORM=FIXUNB
  = X RECFORM=FIXBLK
  = U RECFORM=UNDEF
  = D RECFORM=UNDEF and BLKFAC=YES

b = C CONTROL=YES
  = Z CONTROL=YES is not specified

c = I IOAREA2=YES
  = W WORKA=YES
  = B both are specified
  = Z neither is specified

d = T device is in tape mode
  = D device is in document mode

e = Z always

Subset/Superset ORMOD Names

The following chart shows the subsetting and supersetting allowed for ORMOD names.

One of the parameters allows subsetting. For example, the module IJMFCITZ is a superset of the module IJMFZITZ. See Subset/Superset: (Module Names).

```
                * + * *
        I J M D C B D Z
              F Z I T
              U   W
              X   Z

 + Supersetting/subsetting permitted.
 * No subsetting/supersetting permitted.
```

PRINTER FILE (DTFPR)

A DTFPR entry is included for each printer file processed in the program. The first entry is the DTFPR header entry. The name field contains the symbolic file name, filename. The operation field contains DTFPR. The detail entries, in any order, follow the DTFPR header entry with keyword operands in the operand field. Figure 18 contains DTFPR operands.

```
|BLKSIZE={n|121}                          |
```

This operand specifies the length of IOAREA1. If the record format is variable or undefined, enter the length of the longest record. If this entry is omitted, 121 is assumed.

Note: A maximum BLKSIZE of 151 may be specified for the 3211.

```
|CONTROL=YES                              |
```

This operand is specified if the CNTRL macro will be issued to the file. If this operand is specified, omit CTLCHR.

```
|CTLCHR={YES|ASA}                         |
```

This operand is specified if first character control is used. The parameter ASA specifies the American National Standards Institute, Inc. character set. The entry CTLCHR=YES specifies the EBCDIC character set. Appendix B contains the control character codes. If this parameter is specified, omit CONTROL.

```
|DEVADDR={SYSLOG|SYSLST|SYSnnn)}          |
```

This operand specifies the symbolic unit to be associated with this printer.

```
|DEVICE={1403|1404|1443|1445|3211}        |
```

This operand specifies that one of the following printers is used for the file: 1403, 1404 (continuous forms only), 1443, 1445, or 3211. If this entry is omitted, 1403 is the assumed device.

```
|ERROPT={RETRY|name}                      |
```

If RETRY is specified and an equipment check with command retry is encountered, the command is retried once. If retry is unsuccessful, an informational message is issued and the job is canceled.

If the name of a user error routine is specified and an equipment check with command retry is encountered, the command is retried once. If the retry is unsuccessful, an informational message is issued, and the job is canceled. When certain other errors are encountered, an informational message is issued, error information posted in the CCB, and control given to the user error routine. If any other IOCS macros are used in this routine, the contents of register 13 (with RDONLY) and register 14 must be saved and restored after their use.

Note: If ERROPT is specified, then DEVICE=3211 must be specified.

```
|IOAREA1=name                             |
```

This operand specifies the output area. An address expression (name) is specified.

```
|IOAREA2=name                             |
```

This operand specifies a second output area. An address expression (name) is specified.

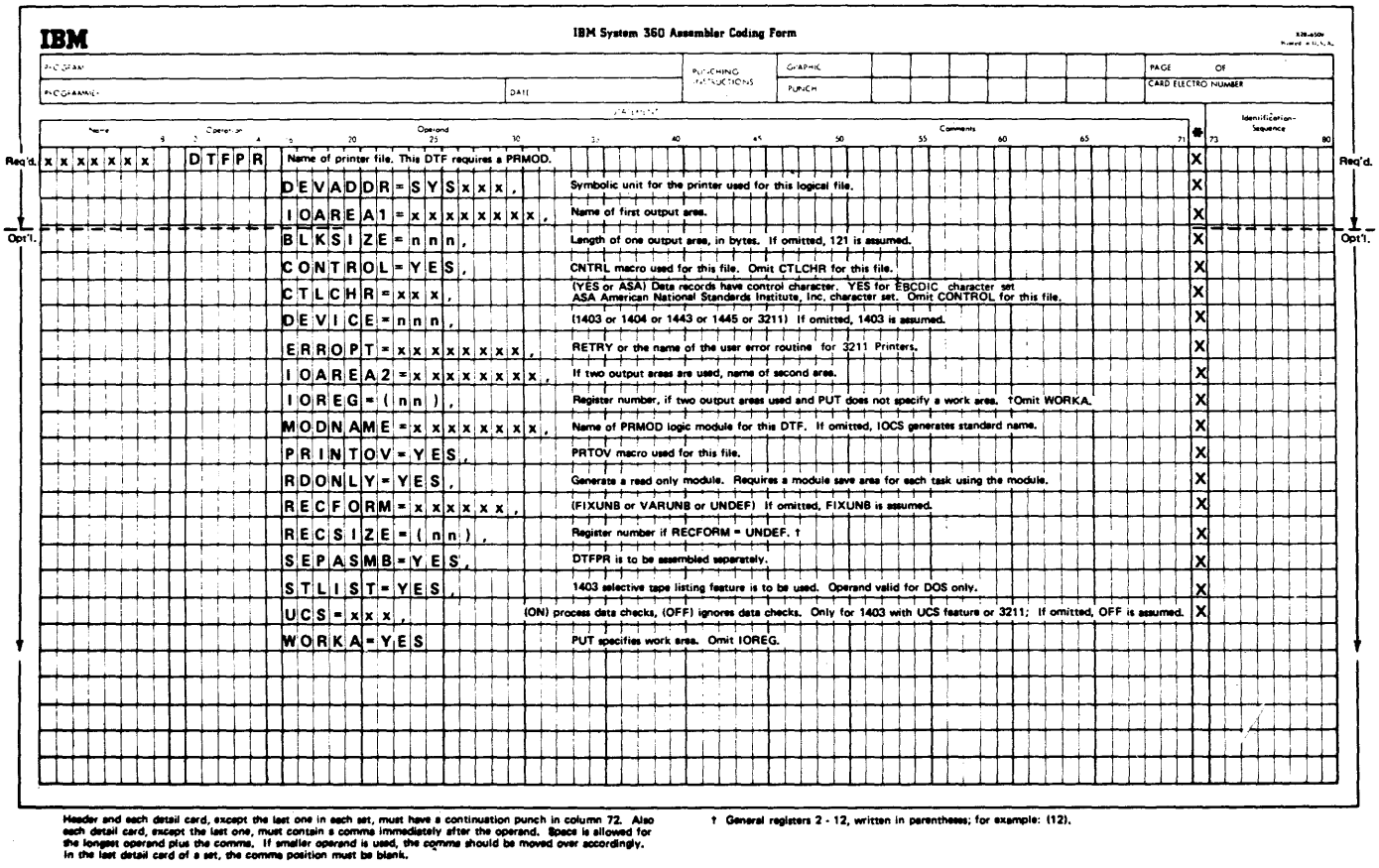


Figure 18. DTFPR Macro

```
|IOREG=(r)                                |
```

If two output areas and no work areas are used, the operand IOREG=(r) specifies the address of the area where the user can build a record. The (r) represents a register 2-12.

```
|MODNAME=name                             |
```

This operand may be used to specify the name of the logic module that is used with the DTF table to process the file. If the logic module is assembled with the program, the MODNAME in the DTF macro instruction must specify the same name as the PRMOD macro instruction. If this operand is omitted, standard names are generated for calling the logic module. If two DTF macro instructions call for different functions that can be handled by a single module, only one module is called.

```
|PRINTOV=YES                              |
```

This operand is specified if the PRTOV macro instruction is included in the problem program.

```
|RDONLY=YES                               |
```

This operand is specified if the DTF is used with a read only module. Each time a read only module is entered, register 13 must contain the address of a 72-byte doubleword aligned save area. Each DTF requires its own uniquely defined save area. Each time an imperative macro (except OPEN(R), LBRET, SETL, or SETFL) is issued for a particular DTF, register 13 must contain the address of the save area associated with that DTF. The fact that the save areas are unique for each task makes the module reentrant (that is, capable of being used concurrently by several tasks). For more information see Shared Modules and Files.

If an ERROPT or WLRERR routine issues I/O macro instructions using the same read only module that caused control to pass to either error routine, the problem program must provide another save area. One save area is used for the normal I/O, and the second for I/O operations in the ERROPT or WLRERR routine. Before returning to the module that entered the ERROPT routine, register 13 must be set to the save area address originally specified for that DTF. If this operand is omitted, the module generated is not reenterable and nc save area need be established.

```
|RECFORM={FIXUNB|UNDEF|VARUNB}
```

The operand RECFORM=FIXUNB is specified whenever the record format is fixed. When the record format is FIXUNB, this entry may be omitted. The entry RECFORM=UNDEF is specified whenever the record format is undefined. If the output is variable and unblocked, enter VARUNB.

```
|RECSIZE=(r)
```

This operand specifies the general register (2-12) that will contain the length of the output record with an undefined format. The length of each record must be loaded into the register before issuing the PUT instruction.

```
|SEPASMB=YES
```

This operand must be included if the DTF is assembled separately. This causes a CATALR card with the filename to be punched ahead of the object deck and defines the filename as an entry point in the assembly.

```
|STLIST=YES
```

Include this operand if the selective tape listing feature (1403 only) is used. If this entry is specified, the CONTRCL, CTLCHR, and PRINTOV entries are not valid and will be ignored if specified. Alsc, the RECFORM entry is forced to FIXUNB and records are handled accordingly.

```
|UCS={ON|OFF}
```

For a 1403 printer with the Universal Character Set feature or a 3211, this cperand determines whether data checks occurring on unprintable characters are indicated to the operator, printed as blanks, or ignored. The entry is especially useful to programmers who are using first-character forms control and who have modules that cannot process the CNTRL macro.

ON Data checks are processed with an operator indication.

OFF Data checks are ignored and blanks are printed for the unprintable character.

```
|WORKA=YES
```

If output records are processed in workareas instead of the output area, the entry WORKA=YES is specified. The user must set up the workarea(s) in main storage. The address expression of the workarea (or a general register containing the address) must be specified for each GET or PUT macro.

PRINTER MODULE (PRMOD)

Listed here are the user-supplied operands for PRMOD. The first card contains PRMOD in the operation field and may contain a user module name in the name field.

```
|CONTROL=YES
```

Include this operand if CNTRL macro instructions are used with the associated DTFs. The module also processes files that do nct use the CNTRL macro instruction. If CONTROL is specified, the CTLCHR operand should not be specified.

```
|CTLCHR={YES|ASA}                          |
```

Include this operand if first-character carriage-control is used. Whenever this operand is included, any DTF used with the module must also specify CTLCHR with the appropriate YES or ASA parameter. If CTLCHR is included, CONTROL should not be specified.

> Note: If more than one DTF uses the same module in a multitasking environment with CTLCHR=ASA and RDONLY=YES, overprinting may occur.

```
|DEVICE={1403|1404|1443|1445|3211}         |
```

This operand specifies that one of the following printers is used for the file: 1403, 1404 (continuous forms only), 1443, 1445, or 3211. Enter one of these numbers. If this entry is omitted, 1403 is the assumed device.

```
|ERROPT=YES                                |
```

ERROPT is valid only for the 3211, and ERROPT=YES must be specified if ERROPT=name is specified in the DTFPR. If ERROPT was not specified or was specified as ERROPT=RETRY, then this operand must not be used in PRMOD.

```
|IOAREA2=YES                               |
```

Include this operand if a second I/O area is used. Any DTF used with the module must also include the IOAREA2 operand.

```
|PRINTOV=YES                               |
```

Include this operand if PRTOV macro instructions are used with the associated DTFs. The module also processes any files that do not use the PRTOV macro instruction.

```
|RDONLY=YES                                |
```

This operand generates a read only module. RDONLY=YES must be specified in the DTF. For the programming requirements of this operand, see the DTF RDONLY operand.

```
|RECFORM={FIXUNB|VARUNB|UNDEF}             |
```

This operand generates a module that processes the specified record format: fixed-length, variable-length, or undefined. Any DTF used with the module must include the appropriate parameter in the RECFORM operand.

```
|SEPASMB=YES                               |
```

This operand must be included if the logic module is assembled separately. This causes a CATALR card with the module name (standard or user) to be punched ahead of the object deck.

```
|STLIST=YES                                |
```

Include this operand if the selective tape listing feature (1403 only) is used. If this entry is specified, the CONTROL, CTLCHR, and PRINTOV entries are not valid, and are ignored if supplied. Also, the RECFORM entry is forced to FIXUNB and records are handled accordingly.

```
|WORKA=YES                                 |
```

Include this operand if records are processed in work areas instead of I/O areas. Any DTF used with the module must include the appropriate parameter in the WORKA operand.

Recommended Module Name for PRMOD

Each name begins with a 3-character prefix (IJD) followed by a 5-character field corresponding to the options permitted in the generation of the module.

PRMOD name = IJDabcde

a = F RECFORM=FIXUNB
  = V RECFORM=VARUNB
  = U RECFORM=UNDEF

b = A CTLCHR=ASA
  = Y CTLCHR=YES
  = C CONTROL=YES
  = S STLIST=YES
  = Z neither is specified

c = B ERROPT=YES (ERROPT=name in DTFPR) and PRINTOV=YES.
  = P PRINTOV=YES and ERROPT is not specified (ERROPT=RETRY or is omitted in DTFPR).
  = E ERROPT=YES (ERROPT=name in DTFPR).
  = Z neither ERROPT (ERROPT=RETRY or is omitted in DTFPR) nor PRINTOV is specified.

d = I IOAREA2=YES
  = Z IOAREA2=YES is not specified

e = V RDONLY=YES and WORKA=YES
  = W WORKA=YES
  = Y RDONLY=YES
  = Z neither is specified

Subset/Superset PRMOD Names

The following chart shows the subsetting and supersetting allowed for PRMOD names. Two of the parameters allow subsetting. For example, the module name IJDFCPIW is a superset of the module names IJDFCZIW and IJDFZZIW. See Subset/Superset: (Module Names).

			*	*	+	*	*	
		I	J	D	F	A	B	I	V
					V	Y	E	Z	W
					U	S	+		Y
						+	P		Z
						C	Z		
						Z			

+ Subsetting/supersetting permitted.
* No subsetting/supersetting permitted.

PAPER TAPE FILE (DTFPT)

A DTF entry is included for every paper tape input or output file that is processed by the program. The first entry must be the DTFPT header entry. Enter the symbolic name of the file, filename, in the name field, and DTFPT in the operation field. The detail entries follow the DTFPT header card in any order. Keyword operands are contained in the operand field (see Figure 19).

```
BLKSIZE=n
```

This operand specifies the length of the input or output area. The maximum block size is 32,767 bytes (32K minus one).

Input: For undefined records, this area must be at least one position larger than the longest record including all shift and delete characters included in the record. If the record is larger than the block size and the wrong-length record option was specified, the remainder of the record is read as the next record. For fixed-length records, this area must be the same size as the record. If shift and delete characters are included in the record (the SCAN entry is specified), BLKSIZE indicates the number of characters required by the problem program after translation and compression. OVBLKSZ contains the number of characters to be read in to produce the BLKSIZE number.

Output: For undefined records, the area must be at least equal to the longest record, including all shift characters that are to be included in the record. For fixed-length records, the area must be the same size as the record. For shifted codes (when the FSCAN and LSCAN entries are specified), BLKSIZE must contain the number of characters before translation and insertion of shift characters. OVBLKSZ must contain the number of characters after translation and insertion of shift characters.

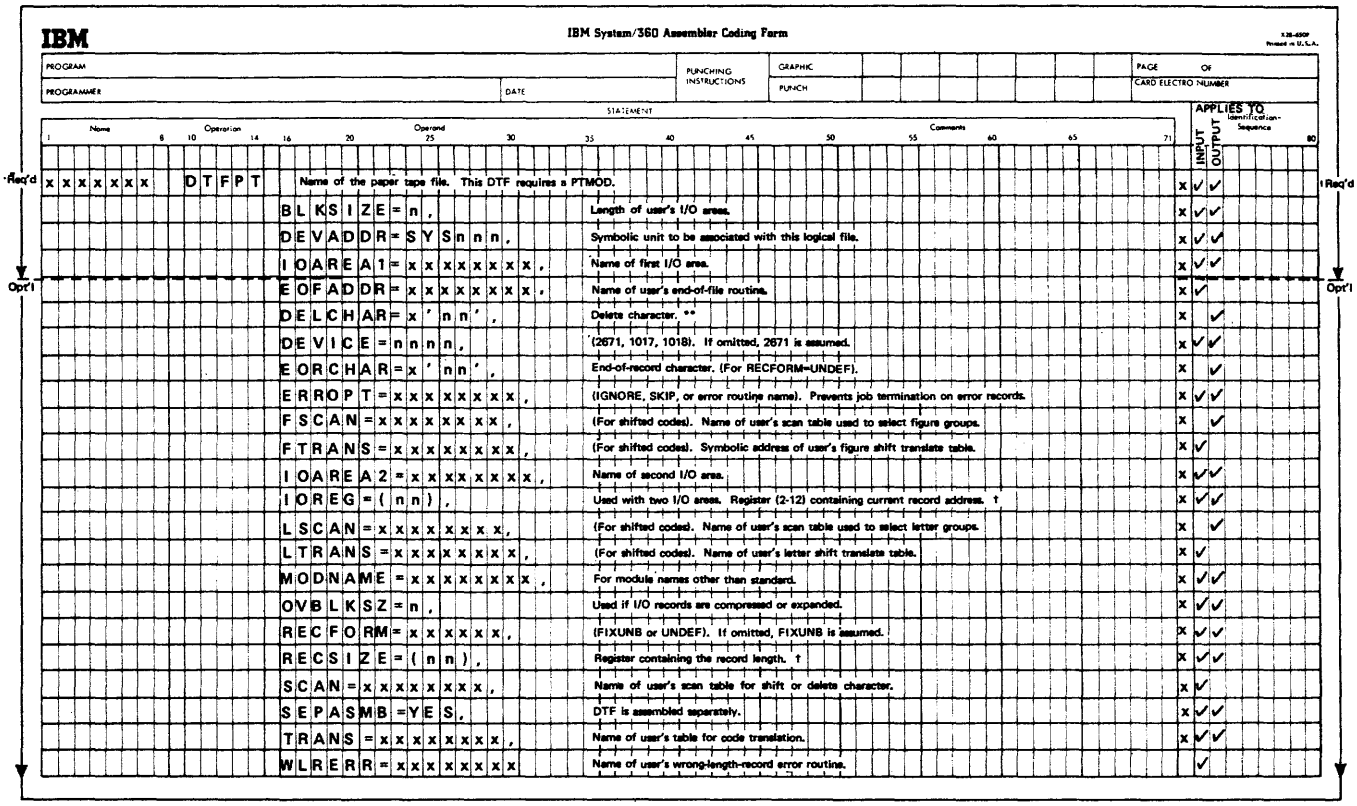


Figure 19. DTFPT Macro

```
DELCHAR=X'nn'
```

This operand specifies the configuration of the delete character and must be used for output files only, that is, when DEVICE=1018 is specified. The constant X'nn' consists of two hexadecimal digits. The delete character is used in the error recovery procedure, and the user must specify the correct configuration in accordance with the number of tracks of the output tape, as follows:

- X'1F' for five tracks.
- X'3F' for six tracks.
- X'7F' for seven tracks.
- X'FF' for eight tracks.

Note: The delete character is required only if the IBM 1018 Paper Tape Punch has the Error Correction feature.

```
DEVADDR=SYSnnn
```

This operand specifies the symbolic unit (SYSnnn) associated with this logical file. An actual channel and unit are assigned to the unit by an ASSGN card in the job control statement. The ASSGN statement contains the same symbolic name as DEVADDR.

```
DEVICE={2671|1017|1018}
```

This operand is required only to specify an I/O device other than 2671. If this entry is omitted, 2671 is assumed.

```
EOFADDR=name
```

This operand specifies the symbolic name of the problem program end-of-file routine. IOCS automatically branches to this routine on an end-of-file condition if the end-of-file switch is set ON. The problem

program routine can execute any operation required for the end-of-file, issue the CLOSE instruction for the file, or return to IOCS by branching to the address in register 14. In the latter case, IOCS reads in the next record. The end-of-file condition cannot occur on the IBM 1018 Paper Tape Punch.

```
|EORCHAR=X'nn'                          |
```

This operand specifies the user-defined end-of-record (EOR) character, where nn is two hexadecimal digits, and must be used for output files with undefined record format only. IOCS writes this character after the last character of the undefined record.

```
|ERROPT={IGNORE|SKIP|name}              |
```

This operand is specified if the user does not want a job terminated when standard recovery procedure cannot recover from a read or write error. If the ERROPT entry is omitted and a read or write error occurs, IOCS terminates the job.

For input files, the IGNORE entry allows IOCS to handle the record as if no errors were detected. If the SKIP entry is specified, IOCS skips the record in error and reads the next record. For output files with shifted codes, no ERROPT can be specified. For unshifted codes, the options ERROPT=IGNORE and ERROPT=name can be specified. The entry IGNORE allows IOCS to handle the record as if no errors were detected. The ERROPT=SKIP option is ignored and causes IOCS to terminate the job. If two I/O areas are used, the CLOSE macro instruction checks the last record, and the ERROPT=name option is treated as the ERROPT=IGNORE option.

If IGNORE and SKIP are not specified, the symbolic name of the problem program error routine must be supplied to process errors. On an error condition, IOCS reads or writes the complete record, including the error character(s), and then branches to the problem program error routine. At the end of the error routine, the problem program must return to IOCS by branching to the address in register 14. The next record is then read or written. The problem program must not issue any GET or PUT instructions for records in the error block. If the error routine contains any other IOCS macros, the contents of register 14 must be saved and restored.

```
|FSCAN=name                             |
```

This operand must be included for every output file using a shifted code. It specifies the symbolic name of a problem program scan table used to select groups of figures. This table must conform to the specifications of the machine instruction TRANSLATE AND TEST. The entry in the table for each letter character must be the letter shift character, and all other entries must be hexadecimal zero. Any deviation from this results in incorrect translation. For an input file, omit FSCAN=name.

```
|FTRANS=name                            |
```

This operand must be included for every input file using a shifted code. It specifies the symbolic name of a problem program figure shift table. This table must conform to the specifications of the machine instruction TRANSLATE. For an output file, omit FTRANS=name and TRANS=name entries.

```
|IOAREA1=name                           |
```

This operand specifies the input or output area. Enter an address expression (name) that specifies the area.

```
|IOAREA2=name                           |
```

This operand specifies a second input or output area. Enter an address expression (name) that specifies the input or output area. When two I/O areas are specified, IOCS overlaps the I/O operation in one area with the processing of the record in the other.

```
|IOREG=(r)                              |
```

This operand must be included if two input or output areas are used. For input, it specifies the register into which IOCS puts the address of the logical record available for processing. For output, it specifies the address of the area into which the problem program can build a record. Any register from 2 to 12 may be specified.

```
|LSCAN=name                                |
```

This operand must be included for every output file using a shifted code. It specifies the symbolic name of a problem program scan table used to select groups of letters. This table must conform to the specifications of the machine instruction TRANSLATE AND TEST. The entry in the table for each figure character must be the figure shift character, and all other entries must be hexadecimal zero. Any deviation from this results in incorrect translation. For an input file, omit LSCAN=name.

```
|LTRANS=name                               |
```

This operand must be included for every input file using a shifted code. It specifies the symbolic name of a problem program letter shift table. This table must conform to the specifications of the machine instruction TRANSLATE. For an output file, omit the LTRANS=name and TRANS=name entries.

```
|MODNAME=name                              |
```

This operand may specify the name of the logic module used with the DTF table to process the file. If the logic module is assembled with the program, the MODNAME operand in the DTFPT macro instruction must specify the same name as the PTMOD macro instruction. If MODNAME=name is omitted, IOCS generates standard names for calling the logic module.

```
|OVBLKSZ=n                                 |
```

For input files, this operand specifies the number of characters to be read in, (before translation and compression) to produce the number of characters specified in the BLKSIZE entry. OVBLKSZ is used only when SCAN and RECFORM=FIXUNB are both specified. If OVBLKSZ is omitted, IOCS assumes the number of characters to be read is equal to the number specified in the BLKSIZE entry. The maximum value is 32,767 bytes (32K minus one).

For output files, OVBLKSZ specifies the number of characters indicated in the BLKSIZE entry, plus the number of shift characters to be inserted. If the size of OVBLKSZ is large enough to allow the insertion of all the shift characters required to build the output record, a single WRITE operation results from a PUT macro. On the other hand, if the size of OVBLKSZ (which must be at least one position larger than BLKSIZE) does not permit the insertion of all the shift characters, several WRITE operations result. OVBLKSZ is used only when LSCAN and FSCAN are specified with the FIXUNB format. If OVBLKSZ is coded with UNDEF format, it is ignored.

```
|RECFORM={FIXUNB|UNDEF}                    |
```

This operand specifies the record format for the file. Specify either format for shifted or unshifted codes, but if the record format is FIXUNB, this entry may be omitted.

```
|RECSIZE=(r)                              |
```

This operand specifies the number of a register (2-12) that contains the length of the input or output record. This entry is optional for input file and, if present, IOCS loads the length of each record read into the specified register. If input files contain shift codes or other characters requiring deletion, IOCS loads the compressed record length in the specified register. For output files, this entry must be included for undefined records. Before translation, the problem program must load each record length into the designated register before issuing the PUT instruction for the record.

```
|SCAN=name                                 |
```

This operand must be included for all input files using shifted codes. It may also be included if the user wishes to delete certain characters from each record. The SCAN entry specifies the symbolic name of a table provided by the problem program. This table must conform to the specifications of the machine instruction TRANSLATE AND TEST. It must contains nonzero entries for all delete characters and, where appropriate, for the figure and letter shift characters. The table entry for the figure shift character must be hexadecimal 04, and for the letter shift character, hexadecimal 08. Delete entries must be hexadecimal 0C. All other entries in the table must be hexadecimal 00.

Otherwise, incorrect translation results and may produce a program check.

The table must be large enough to hold the maximum binary value of coding for the tape being processed; that is, 255 bytes for 8-track tape. This prohibits erroneous coding on the tape from causing a SCAN function beyond the limits of the SCAN table.

```
┌-----------------------------------------┐
|SEPASMB=YES                              |
└-----------------------------------------┘
```

This operand must be included if the DTF is assembled separately. This causes a CATALR card with the filename to be punched before the object deck and defines the filename as an entry point in this assembly.

```
┌-----------------------------------------┐
|TRANS=name                               |
└-----------------------------------------┘
```

The TRANS operand specifies the symbolic name of a table provided within the problem program. This table must conform to the specifications of the machine instruction TRANSLATE. For input files, include this entry if a nonshifted code is to be translated into internal EBCDIC code. Omit the FTRANS and LTRANS entries if this entry is present. If none of these three entries is present, no translation takes place. For output files, include this entry if the internal EBCDIC code is translated into a shifted or nonshifted code, depending on whether the FSCAN and LSCAN entries are present or omitted.

```
┌-----------------------------------------┐
|WLRERR=name                              |
└-----------------------------------------┘
```

This operand applies only to paper tape input files when RECFORM=UNDEF is specified.

When IOCS finds a wrong-length record, it branches to the symbolic name specified in the WLRERR entry. If this entry is not included and the ERROPT entry is included, IOCS considers the error uncorrectable and uses the ERROPT option specified. Absence of both ERROPT and WLRERR entries causes the wrong-length record to be accepted as a normal record. Wrong-length checking is not performed for fixed-length records because a fixed number of characters is read in each time. IOCS detects overlength undefined records when the incoming record fills the input area. The input area must, therefore, be at least one position longer than the longest record anticipated.

At the end of the WLRERR routine, the problem program must return to IOCS by branching to the address in register 14. IOCS causes the next record to be read. If any other IOCS macros are included in the record-length error routines, the problem program must save and restore the contents of register 14 in the error routine.

> Note: A wrong-length condition appears during the first read operation (IBM 1017) if the combined length of the tape leader and of the first record is greater than the length of the longest record anticipated (length specified in BLKSIZE).

CHARACTERISTICS OF A PAPER TAPE FILE

Record Formats

The paper tape reader and punch accept only fixed unblocked records and undefined records. Shifted and nonshifted codes are acceptable in both formats.

Fixed Unblocked Records

Input File: Fixed unblocked records should not be followed by an end-of-record (EOR) character since it enters main storage as a normal character and does not stop the data transfer. The problem program must define the number of characters contained in each record in the BLKSIZE entry. A count controlled read causes the number of characters specified (or fewer if an EOF condition occurs) to be read following a GET instruction. If shift codes and delete characters are included in the record, the record is translated and compressed. IOCS automatically reads until it obtains a translated record of the specified length. Control then returns to the problem program.

When using fixed unblocked records, the problem program must not clear the input area beyond the length specified in the BLKSIZE entry because the next record has already been read, and remains until the next GET instruction is issued.

Output File: The problem program must define the number of characters contained in each record in the BLKSIZE entry, and IOCS translates it if needed. A count-controlled write causes the specified number of records to be written as the

result of a PUT instruction. For shifted code files, IOCS translates and expands the record by adding the letter and figure shift characters. IOCS performs additional writes until the specified record is written. Control then returns to the problem program.

Undefined Records

Input File: Each undefined record must end with an EOR character. The problem program must define an input area and a BLKSIZE entry that is at least one position longer than the longest record anticipated. A modified read (under control of count and the EOR character) causes the number of characters specified (up to the next EOR character) to be read. This record is translated and then compressed, if SCAN is specified. There are no rereads. However, consecutive EOR characters without intervening data are skipped, no zero length record indication is passed back to the problem program, and the next data record is read. If the record fills the input area, it is assumed to be overlength, and IOCS takes the wrong-length record option if specified by the user.

Output File: In an output file, undefined and fixed unblocked records have the same format with one exception. For undefined records, the problem program must define an output area and a BLKSIZE entry that is at least equal to the longest record anticipated. The end-of-record (EOR) character specified by the problem program is added by IOCS.

> Note: All processing is carried out in the problem program's input or output area(s). Workareas are not utilized unless the problem program moves the record to or from the work area. The filename and workarea form of the GET/PUT macro instructions cannot be used for a paper tape file. Only the filename form is acceptable. If there are two input or output areas, the problem program must reference fields within the records by using displacements relative to the general register specified in the IOREG entry, or by using a DSECT.

Code Translation

Input File: The TRANS entry is for records containing nonshifted codes. Translation is performed directly into internal System/360 code. If the input tape is punched in EBCDIC code, no translation is required and all translation entries may be omitted.

If the input tape contains shifted codes, the FTRANS, LTRANS, and SCAN entries must be included. IOCS assumes that the first record read from the input tape starts in figure shift. Therefore, if the first record starts with letter shift coding, the problem program must ensure that the first character of the first record is a letter shift character. The shift status is carried from one record to the next and remains unchanged until another shift character is encountered.

Translation of shifted codes is accomplished by IOCS as follows:

1. The record is first scanned for shift characters. The segment between the shift characters is translated, using the appropriate shift table.
2. The translated segment is moved to the left to remove the shift character.
3. Steps 1 and 2 are repeated for each segment until the complete record has been translated and compressed.

These steps result in a translated and compressed record left-justified in the input area. The record length is communicated to the user in the register designated in the RECSIZE entry, if present.

If the record format is fixed unblocked, the number of characters specified in the OVBLKSZ entry is read, translated, and compressed. If the resulting record is shorter than that specified in the BLKSIZE entry, additional reads are performed until the record length is equal to, or greater than, the BLKSIZE specification. The record is then available to the problem program. If the final read results in a record length that exceeds the BLKSIZE specification, the remaining characters are moved into the beginning of the input area when the next GET is issued.

SCAN may be used alone or in conjunction with TRANS to delete characters from records that do not contain shifted code. There must not be any 04 or 08 entries in the scan table.

The EOR character must be independent of shift status. That is, it must be effective whether the coding is in letter or figure shift. If there is valid character coding in either shift that corresponds to the EOR coding established for a particular job, the corresponding code must not be included in the input record.

Output File: The TRANS entry is used for both shifted and nonshifted codes, and translation is performed from internal System/360 code into a shifted or nonshifted code. If the tape is punched in EBCDIC code, no translation is required and the TRANS entry may be omitted.

For shifted codes, the following three entries must be included: TRANS, LSCAN, and FSCAN.

If the OVBLKSZ entry is omitted, translation into shifted codes follows the procedure explained in Step 1 if the first character of the record is a figure character. Translation with shifted codes follows the procedure explained in Step 2 if the first character of the record is a letter character.

Step 1. The record is scanned for a letter character, using the scan table FSCAN (this table contains nonzero entries for letter characters). The segment so defined that contains figure characters only is then translated by the TRANS table and written out. IOCS places the figure shift character before the segment.

Step 2. The record is scanned for a figure character, using the scan table LSCAN (this table contains nonzero entries for figure characters). The segment is so defined that it contains letter characters only, and is then translated by the TRANS table, and written out. IOCS adds the letter shift character in front of the segment.

In both cases, IOCS then moves the rest of the record to the left and to the beginning of the output area, and proceeds as if an OVBLKSZ entry with the same contents as BLKSIZE were specified. That is, IOCS performs one of these two operations:

- If the OVBLKSZ entry is used, translation into shifted code follows the procedure explained in Step 3, if the first character of the record is a figure character.
- Translation into shifted code follows the procedure in Step 4, if the first character of the record is a letter character.

Step 3. The record is scanned as in Step 1. Instead of writing out the segment of figure characters, IOCS moves it with the rest of the record one position to the right, and places the figure shift character before the segment.

Step 4. The record is scanned as in Step 2. The segment of letter characters and the rest of the record are moved one position to the right. IOCS places the letter shift character in front of the segment.

Step 3 and Step 4 are repeated until the entire record is processed, unless the number specified in OVBLKSZ is insufficient. The result is a translated and expanded record or part of a record, left-justified in the output area. IOCS causes this record to be written out. If the entire record is punched, control returns to the user. If not, any characters remaining to the right of the output area move to the beginning of this output area, and Step 3 and Step 4 resume.

EOF Condition (Input Only)

The EOF condition occurs with an end-of-tape condition when the ECF switch is ON. When IOCS detects this EOF condition (unit exception flag ON in first CSW status byte), it automatically branches to the user's end-of-file routine. However, at the end of his routine, the user can choose to return to IOCS to read a new tape by branching to the address in register 14. If any IOCS macro is contained in this routine, the contents of register 14 must be saved and restored.

If an end-of-tape condition is detected while reading characters other than blanks or deletes (all punched holes), the unit check bit in the first CSW status byte is set ON. This applies only to the IBM 1017 Paper Tape Reader and causes the broken tape bit (bit 7) to appear in the sense byte. The broken tape condition may occur in addition to the ECF condition if the EOF switch is ON.

Trailer Length (Input Only)

To avoid a broken tape condition that would result if the tape trailer is too short, the length of the trailer must be greater than:

- For undefined records (read-EOR command): 2 inches.
- For fixed unblocked records (read command):

$$\left(\frac{\text{Byte Count}}{10} + 2\right) \text{ inches}$$

Note: Byte count is either the count specified in BLKSIZE (record without shifted codes or records with shifted codes but without using the OVBLKSZ operand in the DTFPT), or the count specified in OVBLKSZ (records with shifted codes using the OVBLKSZ operand).

However, when processing undefined records, if a trailer greater than

$$\left(\frac{\text{BLKSIZE}}{10} + 2\right) \text{inches}$$

is read, this trailer will be mistaken for a wrong-length record.

Error Conditions

The paper tape reader or punch stops immediately on an error condition. If the error cannot be corrected and the job is not terminated, IOCS causes the entire record containing the error to be:

- Translated and compressed, if needed, if an input record
- Translated, expanded, and punched, before taking the error option specified by the problem program if an output record.

Wrong Length

For input files, the only wrong-length condition that can be detected is an overlength undefined record. This should be reflected in the BLKSIZE entry. If an undefined record is larger than the block size and the wrong-length record option was specified, the remainder of the record is read as the next record. Wrong-length record indication is impossible with fixed unblocked records, because each record is a sequence of a specified number of characters. Use the FIXUNB record format carefully, because specifying one character too few or too many in any record causes all subsequent records to be out of phase. The problem program should use the RECSIZE entry to check for the correct length of the last record of any file. A record must be entirely on one reel of input tape.

Data Check

The following shows the decision taken by logical IOCS, or possible operator actions, after an unrecoverable data check occurs:

Type of Record Processed	Input Operation	Output Operation
Fixed unblocked record in shifted code	Action 1	Action 1
Fixed unblocked record in nonshifted code	Action 2	Action 2
Undefined record in nonshifted code	Action 2	Action 2
Undefined record in shifted code	Action 2	Action 1

Action 1: The system automatically cancels the job.

Action 2: The operator may choose to:
CANCEL the job
IGNORE the error
or RETRY the operation (For 2671 only)

Following an IGNORE decision, logical IOCS takes action in accordance with the user's option specified in ERROPT.

ERROPT=IGNORE	The record is handled as if no errors were detected.
ERROPT=SKIP (Input file only)	The erroneous record is skipped and the next record is read in.
ERROPT=name	The record is handled as if no errors were detected, and control is given to the user's error routine. At the end of this routine, return must be made to IOCS by branching to the address in register 14, and the next record is read in or written out.
ERROPT omitted	The job is canceled.

Note 1: The character in error is repunched preceded by its corresponding shift character.

- For output records expressed in a paper tape code where the delete character and one of the shift characters have the same configuration.
- Following a data check.

Note 2: The entire erroneous record is repunched as if no errors were detected:

- If an unrecoverable error occurs and the ERROPT operand's name or IGNORE was coded in the DTFPT.
- In the case of output records with two I/O areas, the CLOSE macro instruction checks the successful completion of the last operation.

Note 3: No error condition occurs on the IBM 1018 Paper Tape Punch if the setting of the Tape Width Selector does not match the tape code specified in the problem program.

Note 4: When reading paper tape with physical IOCS, the programmer must restore the CCW address in the CCB before using the EXCP instruction.

Programming Considerations

For information about special equipment considerations for paper tape devices, refer to the IBM 2671 Paper Tape Reader and the IBM System/360 Component Descriptions: 2826 Paper Tape Control Unit, 1017 Paper Tape Reader, 1018 Paper Tape Punch publications listed in the Preface.

PAPER TAPE MODULE (PTMOD)

Listed here are the user-supplied operands for PTMOD. The first card contains PTMOD in the operation field and may contain a user module name in the name field.

```
DEVICE={2671|1017|1018}
```

Required only to specify an I/O device, other than 2671, used by the module. Any DTF used with the module must have the same operand. If this entry is omitted, 2671 is assumed.

```
RECFORM={FIXUNB|UNDEF}
```

Required only if the entry SCAN=YES is present. If records of undefined format using the SCAN option are translated, specify the UNDEF parameter. If records of fixed unblocked format are translated, the FIXUNB parameter may be specified or omitted.

```
SCAN=YES
```

Required for records containing shift characters and/or characters that are automatically deleted by IOCS.

```
SEPASMB=YES
```

This parameter must be included if the logic module is assembled separately. This causes a CATALR card with the module name (standard or user) to be punched before the object deck.

TRANS=YES

Required only if records using an unshifted code are translated and if the entry SCAN=YES is not present.

Summary of PTMOD

The following are the only possible combinations of entries (with or without SEPASMB=YES):

1. No parameters specified, or DEVICE=2671. Module does not provide routines for translation or for shift or delete characters.

2. TRANS=YES[,DEVICE=2671]. Module handles translation of unshifted codes, but not delete characters.

3. SCAN=YES,RECFORM=FIXUNB[DEVICE=2671]. Module handles shift and delete characters for records of fixed unblocked format.

4. SCAN=YES,RECFORM=UNDEF[,DEVICE=2671]. Module handles shift and delete characters for records of undefined format.

5. DEVICE=1017. Module handles neither translation nor shift or delete characters.

6. TRANS=YES,DEVICE=1017. Module handles translation of unshifted codes, but not delete characters.

7. SCAN=YES,RECFORM=FIXUNB,DEVICE=1017. Module handles shift and delete characters for records of fixed unblocked format.

8. SCAN=YES,RECFORM=UNDEF,DEVICE=1017. Module handles shift and delete characters for records of undefined format.

9. DEVICE=1018[,TRANS=YES] [,RECFORM=UNDEF]. Module handles translation of unshifted code, if specified in DTFPT, for records of fixed unblocked or undefined format.

10. SCAN=YES[,RECFORM=UNDEF],DEVICE=1018. Module handles shift characters, for records of fixed unblocked or undefined format.

Recommended Module Name for PTMOD

Each name consists of a 3-character prefix (IJE) followed by a 5-character field corresponding to the options permitted in the generation of the module, as follows:

PTMOD name =IJEabcde

a = S SCAN=YES
  = Z SCAN=YES is not specified

b = T TRANS=YES (SCAN=YES is not specified)
  = Z TRANS=YES is not specified

c = F RECFORM=FIXUNB, and SCAN=YES
  = U RECFORM=UNDEF, and SCAN=YES
  = Z SCAN=YES is not specified, and/or DEVICE=1018

d = 1 DEVICE=1017
  = 2 DEVICE=1018
  = Z DEVICE=2671, or if this entry is omitted

e = Z always

Subsetting and Supersetting of PTMOD Names

The following chart shows the PTMOD names. No subsetting or supersetting is allowed. See Subset/Superset: (Module Names).

			*	*	*	*	
I	J	E	Z	Z	Z	Z	Z
			Z	T	Z	Z	
			S	Z	F	Z	
			S	Z	U	Z	
			Z	Z	Z	1	
			Z	T	Z	1	
			S	Z	F	1	
			S	Z	U	1	
			S	Z	Z	2	
			Z	T	Z	2	

* No subsetting/supersetting permitted.

SEQUENTIAL DASD FILES (DTFSD)

The DTFSD macro instruction defines sequential (consecutive) processing for a file contained in a DASD. Only IBM standard label formats are processed. The DTFSD macro instruction can be used with the 2311, 2314, 2319, and 2321 DASDs.

A DTFSD entry is included for each sequential input or output DASD file that

is processed in the program (Figure 20). The DTFSD header entry and a series of detail entries describe the file. Symbolic addresses of routines and areas are specified in the detail entries. Enter the symbolic name of the file in the name field and DTFSD in the operation field. The detail entries can follow in any order. Keyword operands are contained in the operand field.

Device	Length
2311 Disk Drive	3625 decimal or BLKSIZE
2314, 2319 Disk Drive	7294 decimal or BLKSIZE
2321 Data Cell	2000 decimal or BLKSIZE

BLKSIZE=n

Enter the length (n) of the I/O area. If the record format is variable or undefined, enter the length of the I/O area needed for the largest block of records.

When processing spanned records, the length of the user's define storage statements for his I/O areas must be at least as large as the smaller of the following values:

CONTROL=YES

This operand is specified if a CNTRL macro is to be issued to the file. A CCW is generated for control commands.

DELETFL=NO

Specify this operand if the CLOSE macro is not to delete the Format-1 and Format-3 label for a work file. The operand applies to work files only.

IBM System/360 Assembler Coding Form

Name	Operation	Operand	Comments	INPUT	OUTPUT	WORK
Req'd. xxxxxx	DTFSD	Name of sequential DASD file. This DTF requires an SDMODxx.		X	✓	✓ ✓
	BLKSIZE=nnnn,	Length of one I/O area, in bytes.		X	✓	✓ ✓
	EOFADDR=xxxxxxxx,	Name of user's end-of-file routine.		X	✓	✓
	IOAREA1=xxxxxxxx,	†Name of first I/O area.		X	✓	✓
Opt'l.	CONTROL=YES,	CNTRL macro used for this file.		X	✓	✓ ✓ Opt'l.
	DELETFL=NO,	CLOSE macro is not to delete Format-1 and Format-3 labels for work files.		X		✓
	DEVADDR=SYSxxx,	Symbolic unit required only when not provided on an extent statement.		X	✓	✓ ✓
	DEVICE=nnnn,	(2311, 2314, or 2321) If omitted, 2311 is assumed.		X	✓	✓ ✓
	ERREXT=YES,	Additional errors and ERET are desired. Specify ERROPT.		X	✓	✓ ✓
	ERROPT=xxxxxxxx,	(IGNORE or SKIP or Name of error routine) Prevent job termination on error records. Do not use SKIP for output files.		X	✓	✓ ✓
	FEOVD=YES	Forced end of volume for disk is desired.		X	✓	✓
	HOLD=YES,	Employ the track hold function.		X	✓	✓
	IOAREA2=xxxxxxxx,	If two I/O areas are used, name of second area.		X	✓	✓
	IOREG=(nn),	Register number. † Use only if GET or PUT does not specify work area or if two I/O areas are used. Omit WORKA.		X	✓	✓
	LABADDR=xxxxxxxx,	Name of user's routine to check/write user-standard labels.		X	✓	✓
	MODNAME=xxxxxxxx,	Name of SDMODxx logic module for this DTF. If omitted, IOCS generates standard name.		X	✓	✓
	NOTEPNT=xxxxxxx,	(YES or POINTRW) YES if NOTE/POINTR/POINTW/POINTS used. POINTRW if only NOTE/POINTR/POINTW used.		X		✓
	RDONLY=YES,	Generate a read only module. Requires a module save area for each task using the module.		X	✓	✓ ✓
	RECFORM=xxxxxx,	(FIXUNB, FIXBLK, VARUNB, VARBLK, SPNUNB, SPNBLK, or UNDEF) For work files, use FIXUNB or UNDEF. If omitted, FIXUNB assumed.		X	✓	✓ ✓
	RECSIZE=nnnnnn,	If RECFORM = FIXBLK, number of characters in record. If RECFORM = SPNUNB, SPNBLK, or UNDEF, register number. † Not required for other records.		X	✓	✓
	SEPASMB=YES,	DTFSD is to be assembled separately.		X	✓	✓
	TRUNCS=YES,	RECFORM = FIXBLK or TRUNC macro used for this file.		X	✓	✓
	TYPEFLE=xxxxxx,	(INPUT, OUTPUT, or WORK) If omitted, INPUT is assumed.		X	✓	✓ ✓
	UPDATE=YES,	Input file or work file is to be updated.		X	✓	✓

Figure 20. DTFSD Macro (Part 1 of 2)

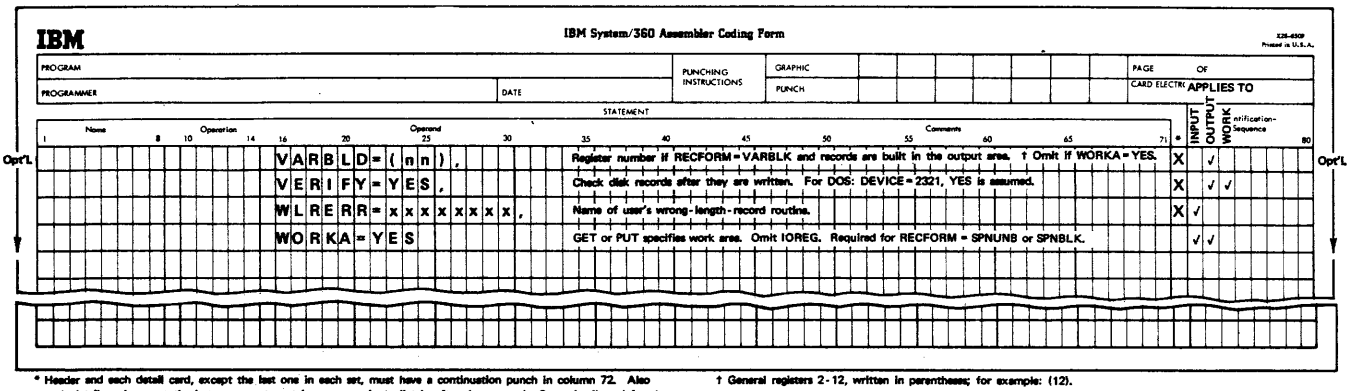


Figure 20. DTFSD Macro (Part 2 of 2)

```
DEVADDR=SYSnnn
```

This operand must specify the symbolic unit (SYSnnn) associated with the file if an extent statement specification is not provided. An extent statement is not required for single-volume input files. If an extent statement is provided, its specification overrides any DEVADDR specification. This specification, or symbolic unit, represents an actual I/O address, and is used in the job control ASSGN statement to assign the actual I/O device address to this file.

A list of symbolic units applying to DTFSD can be found in the Symbolic Unit Addresses section. The only symbolic unit within this list not applicable is SYSLOG.

```
DEVICE={2311|2314|2321}
```

This operand is included to specify whether the data file is located on an IBM 2311, 2314, or 2321. If the location is unspecified, 2311 is assumed.

Note: Specify 2314 for 2319.

```
EOFADDR=name
```

This operand specifies the symbolic name of the user's end-of-file routine. IOCS automatically branches to this routine on an end-of-file condition. The user can perform any operations required for the end of the file in this routine. However, he generally issues the CLOSE macro.

```
ERREXT=YES
```

This operand enables a problem program ERROPT or WLRERR routine to return to SDMOD with the ERET macro. It also enables unrecoverable I/O errors (occuring before a data transfer takes place) to be indicated to the problem program. For ERREXT facilities, the ERROPT operand must be specified. However, to take full advantage of this option give the ERROPT=name operand.

```
┌-----------------------------------------┐
|ERROPT={IGNORE|SKIP|name}                 |
└-----------------------------------------┘
```

This operand is specified if the user does not want a job terminated when a read or write error cannot be corrected in the disk error routines. If a parity error is detected when a block of records is read, the block is reread 256 times before it is considered an error block. After unsuccessfully reading 256 times, the job is terminated unless the ERROPT entry is included. Enter one of the following parameters after the = sign if the ERROPT entry is desired.

IGNORE

The error condition is ignored. The records are made available to the user for processing. When reading spanned records, the whole spanned record or block of spanned records is returned to the user, rather than just the one physical record in which the error occurred. On output, the physical record in which the error occurred is ignored as if it were written correctly. If possible, any remaining spanned record segments are written.

SKIP

No records in the error block are made available for processing. The next block is read from the disk, and processing continues with the first record of that block. When reading spanned records, the whole spanned record or block of spanned records is skipped, rather than just one physical record. On output, the physical record in which the error occurred is ignored as if it were written correctly. If possible, any remaining spanned record segments are written.

name

IOCS branches to the problem program error routine named by this parameter regardless of whether ERREXT=YES is specified. In this routine, the problem program can process or make note of the error condition as desired.

If ERREXT is not specified, register 1 contains the address of the block in error. When spanned records are processed, register 1 contains the address of the whole unblocked or blocked spanned record. Register 14 contains the return address. When processing in the ERROPT routine, the problem program must reference the error block (or records within the error block) by referring to the address supplied in register 1. The contents of the IOREG register or workarea (if either is specified) are variable and therefore should not be used for error block processing. Also, the problem program must not issue GET instructions for records in the error block. If any other IOCS macros (excluding ERET if ERREXT=YES) are used in this routine, the contents of register 13 (with RDONLY) and 14 must be saved and restored after their use. At the end of the routine, the problem program must return control to IOCS by branching to the address in register 14. For a read error IOCS skips that error block and makes the first record of the next block available for processing in the main problem program.

A sequence error may occur if LIOCS is searching for the first segment of a logical spanned record and fails tc find it. If WLRERR or ERROPT=name is specified, the error recovery procedure is the same as for wrong-length record errors. If neither WLRERR nor ERROPT=name is specified, LIOCS ignores the sequence error, and searches for the next first segment. Write errors are ignored.

If ERREXT is specified, register 1 contains the address of a two part parameter list containing the 4-byte DTFSD address and the 4-byte address of the error block respectively. Register 14 contains the return address. Processing is similar to that described above except for addressing the error block and for the following considerations. The data transfer bit (byte 2, bit 2) of the DTF is tested to determine if a nondata transfer error has occurred. If this bit is ON, the block in error was not read or written. If the bit is OFF, data was transferred and the routine must address the block in error to determine the necessary action. At the end of its processing, the routine returns to LIOCS by issuing the ERET macro.

For an input file:

- The problem program skips the block in error and reads the next block with an ERET SKIP.
- Or, it ignores the error with an ERET IGNORE.
- Or it makes another attempt to read the block with an ERET RETRY.

For an output file:

- The problem program ignores the error condition ERET IGNORE or ERET SKIP.

- Or, attempts to write the block with an ERET RETRY macro.

Also, for an output file, the only acceptable parameters are IGNORE or name. On an UPDATE=YES file, the parameter SKIP ignores write errors.

If an error occurs while rereading the physical block while updating spanned records, and neither WLRERR nor ERROPT is specified, the entire logical record is skipped. Likewise, if an error occurs when rereading the physical block that contains the last segment for blocked spanned records, the next entire logical record is skipped. If WLRERR and/or ERROPT were specified, the error recovery procedure is the same as for nonspanned records.

This operand applies to wrong-length records if the WLRERR operand is not included. If the ERROPT routine is used to process wrong-length records, the ERET RETRY option cannot successfully retry the option. ERET RETRY for this condition results in job termination. If both ERROPT and WLRERR are omitted and wrong length records occur, IOCS assumes the IGNORE option.

The DTFSD error options are shown in Figure 21. The following description shows how to use Figure 21. The figure is divided into two parts. The upper part lists the error conditions Specified by the User in the DTF, and the lower part shows the Action Resulting From these specifications when an error occurs. Refer to the shaded column in the figure. The user has specified WLRERR=name and also the RETRY option in his ERET macro. If the error occurring is either a wrong length record, or other than a wrong length record, the job is terminated. Refer to the remaining columns of the figure for other specifications and their resulting actions.

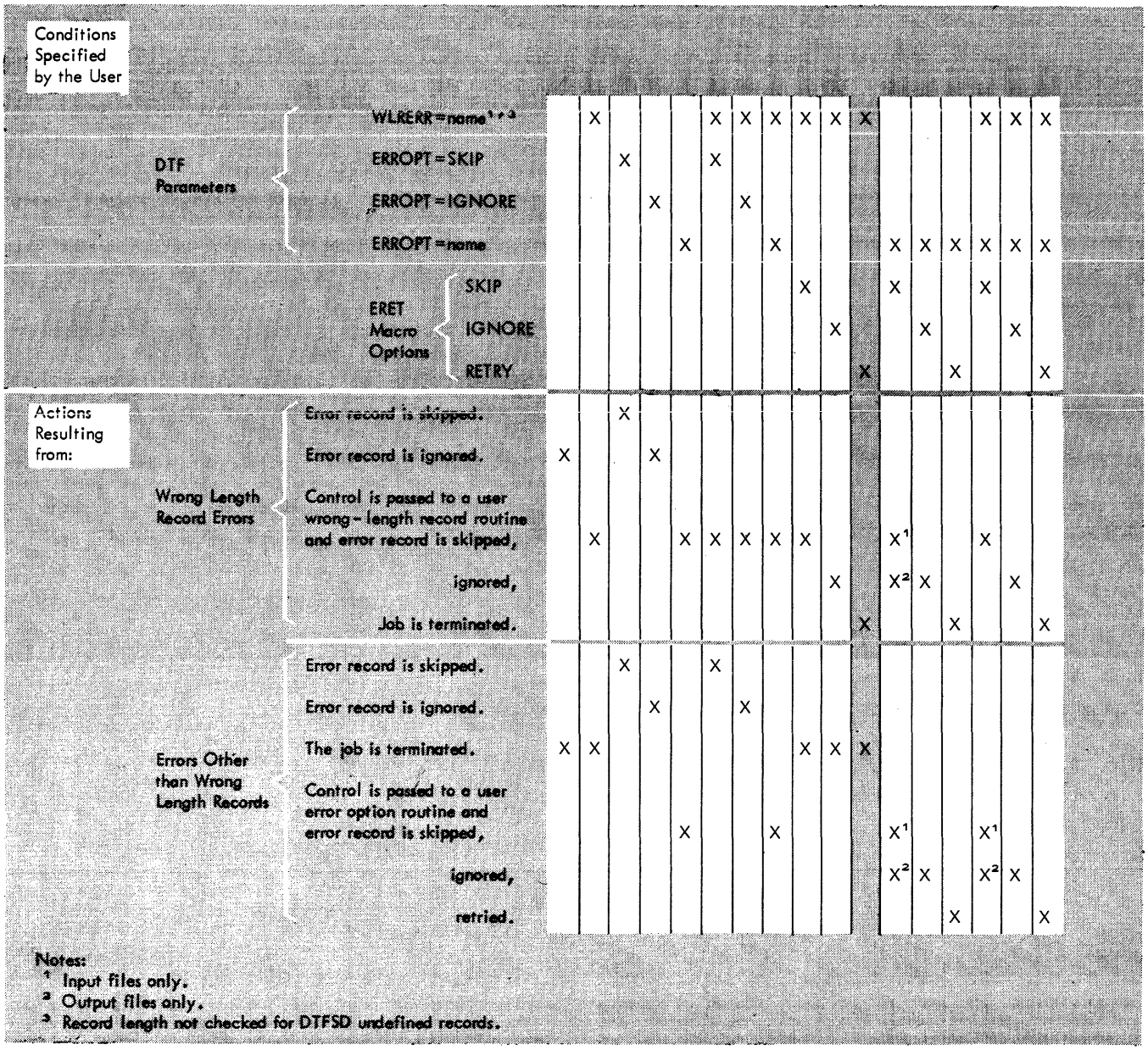


Figure 21. DTFSD Error Options

FEOVD=YES

This operand is specified if the forced end of volume for disk feature is desired. It forces the end-of-volume condition before physical end of volume occurs. When the FEOVD macro instruction is issued, the current volume is closed, and I/O processing continues on the next volume.

HOLD=YES

This operand is specified only if the track hold function is specified at system generation time and if it is employed when a data file or a work file is referenced for updating. SDMOD must also contain this operand. See the Track Protection Macros for more information.

```
IOAREA1=name
```

This operand specifies the symbolic name of the I/O area used by the file. IOCS either reads or writes records using this area. For output records, the first 8 bytes of IOAREA1 must be allotted for IOCS to construct a count field. When variable-length records are processed, the size of the I/O area must include four bytes for the block size. The I/O area must begin on a half-word boundary.

Device	Length
2311 Disk Drive	3625[1] decimal or BLKSIZE
2314, 2319 Disk Drive	7294[1] decimal or BLKSIZE
2321 Data Cell	2000[1] decimal or BLKSIZE
[1]Add 8 decimal for output files	

Figure 22. I/O Area Requirements when Processing Spanned Records

When processing spanned records, the length of the user's define storage statements for his I/O areas must be at least as large as the smaller of the values given in Figure 22.

```
IOAREA2=name
```

If two I/O areas are used by GET or PUT, this operand is specified. When variable length records are processed, the size of the I/O area must include four bytes for the block size. Also, the I/O area must include eight bytes to build a count field for output files.

```
IOREG=(r)
```

This operand specifies the general purpose register (2-12) in which IOCS puts the address of the logical record that is available for processing. At OPEN time, for output files, IOCS puts the address of the area where the user can build a record. The same register may be used for two or more files in the same program, if desired. If this is done, the problem program must store the address supplied by IOCS for each record.

This entry must be specified if blocked input or output records are processed in one I/O area, or if two I/O areas are used and the records are processed in both I/O areas.

```
LABADDR=name
```

Enter the symbolic name of the routine that enables the user to process his own labels. See the sections Writing and Checking User Standard DASD Labels for a discussion of what the LABADDR should do.

```
MODNAME=name
```

This operand may be used to specify the name of the logic module to process the file. If the logic module is assembled with the program, the MODNAME in the DTF macro instruction must specify the same name as the SDMODxx macro instruction. If this entry is omitted, standard names are generated for calling the logic module. If two DTF macro instructions call for different functions that can be handled by a single module, only one module need be called.

```
NOTEPNT={POINTRW|YES}
```

The parameter POINTRW is specified if a NOTE, POINTR, or POINTW macro is issued to the file. If the parameter YES is specified, NOTE, POINTR, POINTW, and POINTS macros may be issued to the file.

```
RDONLY=YES
```

This operand is specified if the DTF is used with a read-only module. Each time a read only module is entered, register 13 must contain the address of a 72-byte doubleword aligned save area. Each DTF should have its own uniquely defined save area. When an imperative macro (except OPEN(R), LBRET, SETL, or SETFL) is issued using a particular DTF, register 13 must contain the address of the save area associated with that DTF. The fact that the save areas are unique for each task makes the module reentrant (that is, capable of being used concurrently by

several tasks). For more information see Shared Modules and Files.

If an ERROPT or WLRERR routine issues I/O macro instructions using the same read only module that caused control to pass to either error routine, the problem program must provide another save area. One save area is used for the normal I/O operations, and the second for I/O in the ERROPT or WLRERR routine. Before returning to the module that entered the ERROPT routine, register 13 must be set to the save area address originally specified for that DTF. If the operand is omitted, the module generated is not reenterable and no save area need be established.

```
RECFORM={FIXUNB|FIXBLK|VARUNB|VARBLK|
        SPNUNB|SPNBLK|UNDEF}
```

This operand specifies the type of records (fixed or variable length, blocked or unblocked, spanned, or undefined) in the input or output file. One of the following parameters may be entered immediately following the = sign.

FIXUNB For fixed-length unblocked records

FIXBLK For fixed-length blocked records

VARUNB For variable-length unblocked records

VARBLK For variable-length blocked records

SPNUNB For spanned variable-length unblocked records.

SPNBLK For spanned variable-length blocked records.

UNDEF For undefined records.

If RECFORM=SPNUNB or RECFORM=SPNBLK was specified and RECSIZE=(r) was not specified, an assembler diagnostic (MNOTE) is issued, and register 2 is assumed. If WORKA=YES was omitted, an MNOTE is issued and WORKA=YES is assumed. If RECFORM is omitted, FIXUNB is assumed.

```
RECSIZE={n|(r)}
```

For fixed-length blocked records, this operand is required. It specifies the number of characters, n, in each record.

When processing spanned records, the user must specify RECSIZE=(r) where r is a register.

For undefined records and variable length spanned records, this entry is required for output files, is optional for input files, and is invalid for work files. It specifies a general register (2-12) that contains the length of the record. On output, the user must load the length of each record into the designated register before he issues a PUT instruction. If specified for input, IOCS provides the length of the record transferred to main storage.

```
SEPASMB=YES
```

This operand must be included if the DTF is assembled separately. This causes a CATALR card with the filename to be punched ahead of the object deck, and defines the filename as an entry point in the assembly.

```
TRUNCS=YES
```

This operand is specified if FIXBLK DASD files contain short blocks embedded within an input file or if the input file was created with a module that specified TRUNCS. This entry is also specified if the TRUNC macro is issued for a FIXBLK output file.

```
TYPEFLE={INPUT|OUTPUT|WORK}
```

Use this operand to indicate whether the file is an input or output file. If WORK is specified, a work file is used. (See work file macros.) If INPUT/OUTPUT is specified, the GET/PUT macro is used. If WORK is specified, the READ/WRITE, NOTE/POINT, and CHECK macros are used.

```
|UPDATE=YES                              |
```

This operand must be included if the DASD input or work file is updated. That is, if disk records are read, processed, and then transferred back (PUT) to the same disk record locations from which they were read, this operand is required. CLOSE writes any remaining records in sequence onto the disk.

```
|VARBLD=(r)                              |
```

Whenever variable-length blocked records are built directly in the output area (no work area specified), this entry must be included. It specifies the number (r) of a general-purpose register (2-12), which will always contain the length of the available space remaining in the output area.

IOCS calculates the space still available in the output area, and supplies it to the programmer in the designated register. Only after the PUT instruction is issued for a variable-length record. The programmer then compares the length of his next variable-length record with the available space to determine if the record fits in the area. This check must be made before the record is built. If the record does not fit, the programmer issues a TRUNC instruction to transfer the completed block of records to the file. Then, the present record is built at the beginning of the output area in the next block.

```
|VERIFY=YES                              |
```

This operand is included if the user wants to check the parity of IBM 2311, 2314, or 2319 records after they are written. If this entry is omitted, any records written on these devices are not verified. VERIFY is assumed when 2321 records are written.

```
|WLRERR=name                             |
```

This operand applies only to disk input files. It specifies the symbolic name of a problem program routine to receive control if a wrong-length record is read.

If ERREXT is not specified, the address of the error block is supplied by IOCS in register 1. If ERREXT is specified, register 1 contains the address of a two part parameter list. The first four bytes of the list are the DTF address, and the second four bytes are the address of the error block. If the block read is less than the BLKSIZE parameter, the first two bytes of the DTF contain the number of bytes left to be read (residual count). Therefore, the size of the actual block is equal to the block size minus the residual count. If the block to be read is larger than the BLKSIZE parameter, the residual count is zero, and there is no way to compute its size. In this case, the number of bytes transferred is equal to the BLKSIZE parameter and the remainder of the original block is truncated.

The problem program WLRERR routine performs any processing desirable for wrong-length records. However, GET macro instructions must not be issued in this routine. If the routine issues any other IOCS macros (excluding ERET if ERREXT=YES) the contents of registers 13 (with RDONLY) and 14 must be saved before and restored after their use. At the end of the routine, control returns to IOCS by branching to the address in register 14. If ERREXT is specified, the ERET IGNORE or SKIP options can be taken. The ERET RETRY terminates the job.

If the WLRERR entry is omitted from the set of DTFSD entries but a wrong-length record is detected by IOCS, one of the following conditions result:

- If the ERROPT entry is included for this file, the wrong-length record is treated as an error block and handled according to the user's specifications for an error (IGNORE, SKIP, or name of error routine).
- If the ERROPT entry is not included, the error is ignored.

The WLRERR entry does not apply to undefined records. Undefined records are not checked for incorrect record length. The record is truncated when the BLKSIZE specification is exceeded.

```
|WORKA=YES                               |
```

Input/output records can be processed, or built, in workareas instead of the input/output areas. If this is planned, WORKA=YES must be included, and the programmer must set up the workarea(s) in main storage. In this case, the symbolic name (or a general register containing the address), must be specified in each GET or PUT instruction. For a GET or PUT instruction, IOCS moves the record to, or

from, the specified work area. WORKA=YES is required for SPNUNB and SPNBLK. Whenever this entry is included for a file, the DTF entry IOREG must be omitted.

SEQUENTIAL DASD MODULE (SDMODXX)

Sequential DASD module generation macros differ from other IOCS module generation macros. The file characteristics are separated into ten categories, and each category has a unique macro instruction associated with it.

Macro	Module Generated
SDMODFI	Sequential DASD Module, Fixed length records[1], Input file
SDMODFO	Sequential DASD Module, Fixed length records[1], Output file
SDMODFU	Sequential DASD Module, Fixed length records[1], Update file
SDMODVI	Sequential DASD Module, Variable length records (including spanned records)[2], Input file
SDMODVO	Sequential DASD Module, Variable length records (including spanned records)[2], Output file
SDMODVU	Sequential DASD Module, Variable length records (including spanned records)[2], Update file
SDMODUI	Sequential DASD Module, Undefined records[3], Input file
SDMODUO	Sequential DASD Module, Undefined records[3], Output file
SDMODUU	Sequential DASD Module, Undefined records[3], Update file
SDMODW	Sequential DASD Module, Work file[4]

[1]RECFORM=FIXUNB or FIXBLK in DTFSD

[2]RECFORM=VARUNB, VARBLK, SPNUNB, or SPNBLK in DTFSD

[3]RECFORM=UNDEF in DTFSD

[4]RECFORM=FIXUNB or UNDEF in DTFSD

The macro operation and the keyword operands define the characteristics of the module. The advantages to generating modules for sequential DASD files this way are:

1. Maintenance changes can be made to the module more easily.
2. A module can be generated to handle a specific file more quickly than if there were only one macro.

The operands for the ten macro instructions are shown in Figure 23 and explained in the following section.

SDMODxx Operands

A module name may be contained in the name field of the macro instruction. The macro operation is contained in the operation field (SDMODFI, for example). The operands are contained in the operand field.

```
CONTROL=YES
```

This operand is specified if a CNTRL macro is issued to the file. This entry applies to all SDMODxx macro instructions. The module also processes any DTF in which the CONTROL parameter is not specified.

```
ERREXT=YES
```

Include this operand if nondata transfer errors are returned to a problem program ERROPT routine or if the ERET macro is used with the DTF and module. The ERROPT operand must be specified for this module.

Operand	Required	Comments
CONTROL=YES	If the CNTRL macro is to be issued to the file.	Applies to all SDMODs.
ERREXT=YES	If the module returns nondata transfer errors or is used with the ERET macro.	Applies to all SDMODs.
ERROPT=YES	If the module is to handle error options for an error block.	Applies to all SDMODs.
FEOVD=YES	If the FEOVD macro is to be issued to the file.	Applies to all SDMODs except SDMODW.
HOLD=YES	If the track hold function is to be employed.	Applies to update and work file logic modules.
NOTEPNT={POINTRW|YES}	If NOTE, POINTR, POINTS, or POINTW macros are to be issued to the file.	This parameter applies to SDMODW only. The operand POINTRW generates logic for NOTE, POINTR, and POINTW. The operand YES generates logic for all macros.
RDONLY=YES	If a read only module is to be generated.	Applies to all SDMODs.
RECFORM={SPNUNB|SPNBLK}	If unblocked or blocked spanned records are to be processed.	Applies to SDMODVI, SDMODVO, and SDMODVU only.
SEPASMB=YES	If the module is assembled separately from the DTF.	Applies to all SDMODs.
TRUNCS=YES	If the TRUNC macro is to be issued to the file. Assumed for variable length blocked records.	Applies to all SDMODs for fixed length records.
UPDATE=YES	If SDMODW is to process the WRITE UPDATE macro instruction.	Applies to SDMODW only.

Figure 23. SDMODxx Operands

ERROPT=YES

This operand applies to all SDMODxx macro instructions. This operand is included if the module handles any of the error options for an error block. Logic is generated to handle any of the three options (IGNORE, SKIP, or fname) regardless of which option is specified. The module processes any DTF in which the ERROPT operand is not specified.

If this operand is not included, the user's program is canceled whenever any uncorrectable error except wrong-length record error (which LIOCS ignores) is encountered.

HOLD=YES

This operand applies to update (SDMODFU, SDMODVU, and SDMODUU) and to work files (SDMODW) only. The operand is included if the track hold function is employed. (See the HOLD operand under DTFSD.)

FEOVD=YES

This operand is specified if the forced end of volume for disk feature is desired. It forces the end of volume condition before physical end of volume occurs. When the FEOVD macro instruction is issued, the current volume is closed, and I/O processing continues on the next volume.

```
|NOTEPNT={POINTRW|YES}                    |
```

This operand applies to SDMODW (work files) only. This entry is included if any NOTE, POINTR, POINTS, or POINTW macro instructions are used within the module. If the operand specifies POINTRW, logic to handle only NOTE, POINTR, and POINTW is generated.

If YES is specified, the routines to handle NOTE, POINTR, POINTS, and POINTW are generated and any files that specify NOTEPNT=POINTRW in the DTF are processed.

In any case, any files that do not specify the NOTEPNT parameter in the DTF are processed.

```
|RDONLY=YES                               |
```

This operand generates a read only module. RDONLY=YES must be specified in the DTF. For the programming requirements of this operand, see the DTFSD RDONLY operand.

```
|RECFORM={SPNUNB|SPNBLK}                  |
```

This operand is required only for SDMODVI (input files), SDMODVO (output files), and SDMODVU (update files) if RECFORM=SPNUNB or SPNBLK is specified in the DTF. If RECFORM is specified incorrectly, an assembler diagnostic (MNOTE) is issued, and the module generation is terminated.

```
|SEPASMB=YES                              |
```

This operand must be included if the logic module is assembled separately. This causes a CATALR card with the module name (standard or user) to be punched ahead of the object deck.

```
|TRUNCS=YES                               |
```

This operand applies to all SDMOD macro instructions for fixed-length records. It generates a logic module that supports the TRUNC macro instruction. Also, this operand is assumed for VARBLK output files. It must be specified if any FIXBLK DASD files (processed by the module) contain short blocks embedded within them or if the input file was created with a module that specified TRUNCS. The module cannot process any DTF, for fixed-length records, in which the TRUNCS operand is not specified.

```
|UPDATE=YES                               |
```

This operand applies to the SDMODW only. It is assumed for SDMODFU, SDMODUU, and SDMODVU and generates a logic module that supports the WRITE UPDATE macro instruction with work files.

Recommended Module Name List for SDMODxx

Each name begins with a 3-character prefix (IJG) and consists of a 5-character field corresponding to the options permitted in the generation of the module.

In SDMOD there are two module classes:

- For handling GET/PUT functions
- For handling READ/WRITE, NOTE/POINT, and CHECK functions (work files).

Name List for GET/PUT Type Modules

SDMODxx name = IJGabcde

```
a = C SDMODFx specifies HOLD=YES
  = F SDMODFx does not specify HOLD=YES
  = R SDMODUx specifies HOLD=YES
  = U SDMODUx does not specify HOLD=YES
  = P SDMODVx specifies HOLD=YES (spanned
      records)
  = Q SDMODVx does not specify HOLD=YES
      (spanned records)
  = S SDMODVx specifies HOLD=YES
  = V SDMODVx does not specify HOLD=YES

b = U SDMODxU
  = I SDMODxI
  = O SDMODxO

c = C ERROPT=YES and ERREXT=YES
  = E ERROPT=YES
  = Z neither is specified
```

d = M TRUNCS=YES and FEOVD=YES
  = T TRUNCS=YES
  = W FEOVD=YES
  = Z neither is specified

e = B CONTROL=YES and RDONLY=YES
  = C CONTROL=YES
  = Y RDONLY=YES
  = Z neither is specified

Name List for Workfile Type Modules (TYPEFLE=WORK)

SDMODxx name = IJGabcde

a = T HOLD=YES
  = W HOLD=YES not specified

b = C ERROPT=YES and ERREXT=YES
  = E ERROPT=YES
  = Z neither is specified

c = N NOTEPNT=YES
  = R NOTEPNT=POINTRW
  = Z NOTEPNT is not specified

d = C CONTROL=YES
  = Z CONTROL=YES is not specified

e = T RDONLY=YES and UPDATE=YES
  = U UPDATE=YES
  = Y RDONLY=YES
  = Z neither is specified

Subset/Superset SDMOD Names

The following diagrams illustrate the subsetting and supersetting allowed for SDMOD names. For the GET/PUT type modules, four parameters allow supersetting. For example, in the GET/PUT type module, the module IJGFUETC is a superset of a module with the name of IJGFUZTZ. See Subset/Superset Module Names.

For GET/PUT Type Modules:

```
                    + * + + +
          I J G C U C M B
                F I E T Y
                + O Z + +
                R     W C
                U     Z Z
                +
                P
                Q
                V
                +
                P
                S
                V

+ Subsetting/supersetting permitted.
* No subsetting/supersetting permitted.
```

For Workfile Type Modules:

```
                    + + + + +
          I J G T C N C T
                W E R Z Y
                  Z Z   +
                        U
                        Z

+ Subsetting/supersetting permitted.
* No subsetting/supersetting permitted.
```

SERIAL DEVICE FILE (DTFSR) FOR BOS/360 USERS

The DTFSR macro instruction is provided as a compatibility aid to users of the Basic Operating System/360. For Disk Operating System/360 users, the DTFCD, DTFPT, DTFMT, DTFOR, DTFPR, DTFCN, and DTFSD macros are recommended instead of the DTFSR macro. Enter the symbolic name of the file in the name field and DTFSR in the operation field. The entries for DTFSR are discussed here and illustrated in Figure 24.

The begin-definition card must be punched with DTFBG in the operation field and DISK in the operand field. The name field is blank. It is included in DOS to provide compatibility with the BOS DTFSR macro instruction.

ALTTAPE=SYSnnn

This operand is provided for BPS and BOS compatibility.

BLKFAC=n

Undefined journal tape records are processed faster when this operand is included because it reads groups of lines as blocked records. When undefined records are processed, BLKFAC specifies the blocking factor (n) that determines the number of lines read (through CCW chaining) as a block of data by one physical read. Deblocking is accomplished automatically by IOCS when the GET macro is used. The BLKFAC parameter is not used with RECFORM=FIXBLK, because the blocking factor is determined from the BLKSIZE and RECSIZE parameters. If this operand is included for FIXBLK, FIXUNB, or document processing, an assembler diagnostic (MNOTE) results, and the operand is ignored.

BLKSIZE=n

This operand indicates the size of the input or output area specified by IOAREA1. BLKSIZE specifies the maximum number (n) of characters that may be transferred to or from the area at any one time. When variable-length records are read or written, the area must be large enough to accommodate the largest block of records, or the longest single record if the records are unblocked.

When undefined journal tape records are read, the area must be large enough to accommodate the longest record to be read if the BLKFAC parameter is not specified. If the BLKFAC parameter is specified, the BLKSIZE value must be determined by multiplying the maximum length that must be accommodated for an undefined record by the blocking factor desired. A BLKSIZE value smaller than this results in truncation of data.

If card-punch or printer output records include control characters (that is, DTFSR CTLCHR specified) and/or record-length fields for variable-length records (RECFORM=VARUNB), the BLKSIZE specifications must include the extra bytes allotted in the main-storage output area.

If two input, or output, areas are used for a file (IOAREA1 and IOAREA2), the size specified in this entry is the size of each I/O area.

IOCS uses this block size parameter to:

- Construct the count field of the CCW for an input file.
- Construct the count field of the CCW for an output file of fixed-length records.
- Check physical record length for a file of fixed-length blocked input records.
- Determine if the space remaining in the output area is large enough to accommodate the next variable-length output record.

CHECKPT=n

This operand is for compatibility with BPS and BOS and is ignored by DOS.

CKPTREC=YES

This operand is required if a tape input file contains checkpoint records interspersed among the data records. When this entry is included, IOCS recognizes the checkpoint records and bypasses them.

CONTROL=YES

This operand must be included if a CNTRL macro instruction is issued for a file. A control command issues orders to the I/O device to perform nondata operations such as card-stacker selection, carriage skipping, marking of doubtful 1287 or 1285 tape records, tape rewinding, etc. When CONTROL is included, the DTFSR entry CTLCHR must not be included.

```
┌----------------------------------------┐
|COREXIT=name                            |
└----------------------------------------┘
```

COREXIT provides an exit to the user's error correction routine for the IBM 1285, or 1287 Optical Reader, or 1288 Optical Page Reader. If an error occurs after a GET, WAITF, or CNTRL macro (to increment or eject and/or stacker select a document) is executed, the error correction routine is entered with an indication provided in filename+80. Filename+80 contains the following hexadecimal bits indicating the conditions that occurred during the last line or field read. Filename+80 should also be tested after issuing the optical reader macros DSPLY, RESCN, RDINE, CNTRL READKB, and CNTRL MARK. More than one error condition may be present.

X'20' For the 1288, reading in unformatted mode, the end-of-page (EOP) condition was detected. Normally, on an EOP indication, the problem program ejects and stacker selects the document.

X'01' A data check has occurred. (That is, five read attempts for journal tape processing or three read attempts for document processing were made.)

X'02' The operator corrected one or more characters from the keyboard (1285 or 1287T) or a hopper empty condition (see the HPRMTY=YES operand) has occurred (1287D).

X'04' A wrong-length record condition has occurred after five read attempts were made for journal tapes or three for documents. Not applicable for undefined records.

X'08' An equipment check resulted in an incomplete read after ten read attempts were made for journal tapes or three for documents.

If an equipment check occurs on the first character in the record when processing undefined journal tape records, the RECSIZE register contains zero, and the IOREG (if used) points to the rightmost position of the record in the I/O area. The user should test the RECSIZE register before moving records from the I/O or workarea(s). The test conditions are:

X'10' A nonrecovery error occurred.

X'20' A stacker-select command was given after the allotted time had elapsed and the document is sent to the reject pocket.

X'40' The 1287D scanner was unable to locate the reference mark after ten read attempts were made for journal tapes or three for documents.

Filename+80 can be interrogated by the user to determine the reason for entry into the error correction routine. The choice of action in the user's error correction routine is determined by the particular application.

If the user issues I/O macro instructions to any device other than the 1285, 1287, and/or 1288, in the COREXIT routine, he must save registers 0, 1, 14, and 15 upon entering the routine and restore these registers before exiting. If I/O macro instructions, other than the GET and/or READ, are issued to the 1285, 1287, and/or 1288 in this routine, the user must first save, and later restore registers 14 and 15. All exits, except as noted, from COREXIT should be to the address in register 14. This returns control to the point from which the branch to COREXIT occurred. If a READ document command chain is broken, IOCS completes the chain upon return from the COREXIT routine.

Note: The user cannot issue a GET or a READ macro to the 1285, 1287, or 1288 in his error correction routine. Also, the user should not process records in this routine. The record that caused the exit to the error routine is available for processing upon return to the user's mainline program. Any processing attempted in the error routine is duplicated after return to the mainline program.

When processing journal tapes, a nonrecovery error (torn tape, tape jam, etc) normally requires the tape to be completely reprocessed. In this case, the user must not branch to the address in register 14 from the COREXIT routine or a program loop will result. Following a nonrecovery error, the optical reader file must be closed, the condition causing the nonrecoverable error must be cleared, and the file must be reopened before processing can continue.

When processing documents, a nonrecoverable error requires that the document be removed, either manually or by nonprocess runout. Such an error could result from a jammed document or a scanner

control failure. In such cases, the user's program should branch to read the next document. Also, if the 1287 or 1288 scanner is unable to locate the document reference mark, the document cannot be processed. In this case, the document must be ejected and stacker selected before attempting to read the following document or a program loop will result. In any case, the user must not branch to the address in register 14 from the COREXIT routine. The user should ignore any output resulting from the document under any circumstances.

Eight binary counters are used to accumulate totals of certain 1285, 1287, and 1288 error conditions. These counters each occupy four bytes, starting at filename+48. Filename is the name specified in the DTF header entry. The error counters are:

Counter and Address		Contents
1	filename+48	Incomplete read (equipment check)
2	filename+52	Incomplete read uncorrectable after ten read attempts for journal tapes or three for documents
3	filename+56	Wrong-length records (Not applicable for undefined records.)
4	filename+60	Wrong-length records uncorrectable after five read attempts for journal tapes or three for documents (Not applicable for undefined records.)
5	filename+64	Keyboard corrections (journal tape only)
6	filename+68	Journal tape lines, including retried lines, or document fields, including retried fields, in which data checks are present.
7	filename+72	Lines marked (journal tape only)
8	filename+76	Count of total lines read from journal tape or the number of CCW chains executed during document processing.

All of these counters contain binary zero at the start of each job step and are never cleared. The user may list the contents of these counters for analysis at end of file, or at end of job, or he may ignore the counters. (Binary contents of the counters should be converted to a printable format.)

```
CRDERR=RETRY
```

This operand applies only to a card output file for the IBM 2540 or 2520. It specifies the operation performed if an error is detected.

Normally, if a punching error occurs, it is ignored and operation continues. The error card is stacked in pocket P1 (punch). Correct cards are stacked in the pocket specified by the user. If this CRDERR entry is included, however, IOCS also notifies the operator and then enters the wait state whenever an error condition occurs. The operator can then either terminate the job or instruct IOCS to repunch the card. IOCS automatically generates a retry routine and constructs a save area for the card punch record if this entry is included.

```
CTLCHR=YES
```

The CTLCHR (control character) operand applies only to printer and punch output files. It is included if each logical record written or punched contains a control character (carriage control or stacker selection) in the record itself, or in the main storage output area. For fixed-length or undefined records, the control character must be the first character. For variable-length records, it is the first character after the record-length field. The control character codes are the same as the modifier bytes used for a punch or print command.

When this operand is specified, IOCS routines cause the designated control character for printer or card punch order to be issued to the I/O device. Printing or punching begins with the second character in the record. When the CTLCHR entry is not included, any control functions desired must be performed by the CNTRL macro.

```
DEVADDR=SYSxxx
```

This operand specifies the symbolic unit name to be associated with the file. The symbolic unit name represents an actual I/O device address and is used in the job control ASSGN statement to assign the actual I/O device address to this file. For a complete list of symbolic unit names that can be used for particular devices see Symbolic Unit Addresses. SYSOPT, if used, is processed as if SYSPCH were specified.

A reel of tape may be mounted on any tape unit available at the time the job is ready to run. This is done by assigning the device to the specified symbolic unit name. Whenever two devices are used for one logical file (such as an alternate tape unit specified in the ASSGN cards), this DEVADDR entry specifies the symbolic unit name for the first device.

The symbolic unit name must be specified for all units except the 2311 disk drive. For files on this device, DEVADDR may be omitted. If DEVADDR is omitted, the symbolic unit name for a disk drive is supplied by a job control extent card.

```
DEVICE=
```

This operand must be included to state the I/O device associated with this logical file. One of the following parameters must be entered immediately after the = sign.

DISK11 For an input or output file on disk (2311).

TAPE For an input or output file recorded on magnetic tape (3420 or 2400-series).

PRINTER For output printed on a 1403, 1404, 1443, 1445, or 3211.

READ01 For an input card file in a 2501.

READ20 For an input or output card file in a 2520.

READ40 For an input or output card file in a 2540.

READ42 For an input or output card file in a 1442.

CONSOLE For input from and output to the printer-keyboard.

PTAPERD For input from a 2671.

READ87T For a journal tape input file on a 1287.

READ87D For a document input file on a 1287 or 1288.

READ85 For a journal tape input file on a 1285.

This operand causes IOCS to set up the device dependent routines for a file. For document processing on the IBM 1287 or 1288 Optical Reader, or 1288 Optical Page Reader, the user codes CCWs.

```
EOFADDR=name
```

This operand must be included for:

- Card reader files
- Magnetic tape input files
- Paper tape input files
- Sequential disk input files
- Optical reader files

It specifies the symbolic name of the user's end-of-file routine. IOCS automatically branches to this routine on an end-of-file condition.

IOCS detects end-of-file conditions as follows:

- Card Reader. By recognizing /* punched in card columns 1 and 2. If cards are allowed to run out without a /* trailer card (and a /& card if end-of-job), an error condition is signaled to the operator (intervention required).
- Magnetic Tape Input. By reading a tapemark and EOF in the trailer label when standard labels are specified, or by reading /* if the unit is assigned to SYSRDR or SYSIPT. If standard labels are not specified, IOCS assumes an end-of-file condition when the tapemark is read. The user must determine, in his routine, that this actually is the end of the file.
- Paper Tape Reader. By recognizing the end of tape when the end-of-file switch is set ON.
- Sequential Disk Input. By reading an end-of-file record or reaching the end of the last extent supplied by the user.

- Optical Reader Input. When reading data from documents on a 1287 or 1288, end-of-file condition is recognized by pressing the end-of-file key on the console when the input hopper is empty. When processing journal tapes on a 1287 or 1285, end-of-file is detected by pressing the end-of-file key after the end of the tape has been sensed.

When IOCS detects the end of file, it branches to the user's routine specified by EOFADDR. If journal tapes are being processed, it is the user's responsibility to determine if the current roll is the last roll to be processed. For 1285, we suggest that you key in header information at the beginning of each roll. This information can then be interrogated in this routine to determine whether it is the last roll. Regardless of the situation, the tape file must be closed for each roll within this routine. If the current roll is not the last, the OPEN macro must be issued. The OPEN macro instruction allows header (identifying) information to be entered at the reader keyboard and read by the processor when using logical IOCS.

The same procedure can be used for 1287 processing of multiple journal tape rolls as well as the method described under the OPEN(R) operand.

```
|ERROPT={IGNORE|SKIP|name}                |
```

This operand applies to disk or magnetic tape input files, and it specifies functions to be performed for an error block.

If a parity error is detected when a block of sequential disk records is read, the disk block is reread 256 times before it is considered an error block. If a parity error is detected when a block of tape records is read, the tape is backspaced and reread 100 times before the tape block is considered an error block. Unless the ERROPT operand is included to specify other procedures, the job is then automatically terminated. Either IGNORE, SKIP, or the symbolic name of an error routine can be specified in this entry. One of these parameters is entered immediately after the = sign in this keyword operand. The functions of these three parameters are:

IGNORE

The error condition is completely ignored, and the records are made available to the user for processing.

SKIP

No records in the error block are made available for processing. The next block is read from disk or tape, and processing continues with the first record of that block. The error block is included in the block count, however.

name

IOCS branches to the user's routine, where he may perform whatever functions he desires to process or note the error condition. Register 1 contains the address of the block in error, and register 14 contains the return address.

In his error routine, the programmer should address the error block (or records in the error block) by referring to the address supplied in register 1. The contents of the IOREG register or the work area (if either is specified) may vary and, therefore, should not be used. Also, the programmer must not issue any GET instructions for records in the error block. If he uses any other IOCS macros in his routine, he must save and later restore the contents of register 14. At the end of his routine, the programmer must return to IOCS by branching to the address in register 14. When control returns to the problem program, the first record of the next block is available for processing in the main program.

The ERROPT entry does not apply to disk or tape output files. The job is automatically terminated if a parity error still exists after IOCS attempts to write a disk output block ten times, or to write a tape output block 15 times. The tape procedure includes 15 forward erases. This entry applies to wrong-length records if the DTFSR entry WLRERR is not included. If both ERROPT and WLRERR are omitted, IOCS ignores any wrong-length records that occur.

```
|FILABL=NO|STD|NSTD                       |
```

This operand may be included for a tape input or output file. One of the following parameters is entered immediately after the = sign:

NO For a tape file that does not contain labels. The entry FILABL=NO may be omitted, if desired, and IOCS assumes that there are no labels.

STD For a tape input file if standard labels are checked by IOCS, or for a tape output file if standard labels are written by IOCS.

NSTD For a tape input or output file that has nonstandard labels. These labels may be processed by the user (see Writing Checking Nonstandard Labels). NSTD is specified for standard input labels if they are not to be checked by IOCS.

```
HEADER=YES
```

This operand is required if the operator must key in header (identifying) information from the 1285 or 1287 keyboard. The OPEN routine reads the header information only when this entry is present. If the entry is omitted, OPEN assumes no header information is to be read. The header record size can be as large as the BLKSIZE specification and it is read into the high-order positions of IOAREA1. This operand cannot be used for 1288 files.

```
HPRMTY=YES
```

This operand is included if the hopper empty indication is passed to the user. This condition occurs when a READ is issued and no document is present. When hopper empty is detected, the user's COREXIT routine is entered with the condition indicated as X'02' in filename+80.

This operand should be used when processing documents in the time dependent mode of operation. This allows complete overlapping of processing with reading. (See method 2 under Programming the 1287 in the IBM 1287 Optical Reader Component Description and Operating Procedures publication listed in the IBM System/360 and System/370 Bibliography.) Using the HPRMTY parameter is used with this method of processing, the user is able to check for a hopper empty condition in his COREXIT routine. This allows him then to stacker select properly the previously ejected document, before returning from the COREXIT routine (via register 14).

```
INAREA=name
```

This operand applies only to a card file in an IBM 1442 that is updated (TYPEFLE=CMBND) and for which separate input and output areas are required. INAREA specifies the symbolic name of the input area to which the card record is transferred. OUAREA is used in conjunction with INAREA, and both IOAREA1 and IOAREA2 must be omitted.

When the same I/O area is used for both input and output in a combined file for a 2520 or 2540, INAREA and OUAREA are omitted. IOAREA1 specifies the name of the I/O area used for both input and output files. This entry does not apply to combined files in an IBM 2520 or 2540.

```
INBLKSZ=n
```

This operand is used with INAREA for a combined file in the 1442 when separate input and output areas are required. It specifies the maximum number, n, of characters that are transferred to the input area (INAREA) at any one time. Whenever this entry is included, the corresponding entry OUBLKSZ must also be included, and BLKSIZE must be omitted.

```
IOAREA1=name
```

This operand is included to specify the symbolic name of the input, or output, area used by this file. The input/output routines transfer records to or from this area.

For a disk output file, the user must reserve eight bytes at the beginning of his I/O area, ahead of the positions allotted for data records. These eight bytes are necessary to allow IOCS to construct the count area for the disk record. For 1285 and/or 1287 readers, this area is set to binary zeros before each input operation and before each tape input operation to this area. For document processing, the area is cleared only when the file is opened.

This operand must not be included for a 1442 combined file if INAREA and OUAREA are specified for the file. For a 2520 and 2540 combined file, IOAREA1 must be used for both the input and output area.

```
IOAREA2=name
```

Two input, or output, areas can be allotted for a file, to permit an overlapping of data transfer and processing operations. When this is done, this IOAREA2 entry must be included and specify the symbolic name of the second I/O area.

For a disk output file, the user must reserve eight bytes at the beginning of his I/O area, ahead of the positions allotted for data records. These eight bytes are necessary to allow IOCS to construct the count area for the disk record. For 1285 and/or 1287 readers (journal tape only) this area is set to binary zeros before each input operation to this area. This entry must not be included if TYPEFLE=CMBND (1442), if DEVICE=READ87D. For a 2520 or 2540 combined file, IOAREA2 cannot be specified. IOAREA1 in this case must be used for both the input and output areas.

```
IOREG=(r)
```

This operand specifies the general-purpose registers 2-12 (r) that the input/output routines can use to indicate which individual record is available for processing. IOCS puts the address of the current record in the specified register each time a GET or PUT is issued.

The same register may be specified in the IOREG entry for two or more files in the same program, if desired. In this case, the problem program may need to store the address supplied by IOCS for each record.

This operand must be included whenever:

- Blocked input or output records (from disk, magnetic tape, or journal tape) are processed directly in the I/O area.
- Variable-length unblocked or undefined tape records are read backwards and processed directly in the input area.
- Two input, or output, areas are used and the records (either blocked or unblocked) are processed in the I/O areas.
- Undefined records for journal tape are read.

Whenever this entry is included for a file, the DTFSR entry WORKA must be omitted, and the GET or PUT instructions must not specify work areas.

Since a read by an optical reader is accomplished by a backward scan, the rightmost character in the record is placed in the rightmost position of the I/O area. Subsequent characters are placed in sequence from right to left. The register specified indicates the leftmost position of the record to the user.

```
LABADDR=name
```

The user may require one or more of his own disk or tape labels in addition to the standard file header label or trailer label (on tape). If so, he must include his own routine to check or build the label(s). The symbolic name of his routine is specified in this entry. IOCS branches to this routine after it has processed the standard label. This entry is also required whenever nonstandard labels are checked or written by the problem program. (DTFSR FILABL specifies NSTD.)

LABADDR allows one users' label routine to be specified for all types of labels for the file: header labels, end-of-file labels, and end-of-volume labels. For an input file, the user can determine the type of label that was read by the identification in the label itself. For an output tape file, however, IOCS indicates to the user the type of label. In this case, IOCS supplies a code in the low-order byte of register 0, as follows:

```
O indicates header labels.
F indicates end-of-file labels.
V indicates end-of-volume labels.
```

In his routine the user can test this byte and then build the appropriate type of label. At the end of his routine, the programmer must return to IOCS by use of the LBRET macro. The user may not issue a macro instruction that calls in a transient routine. For example, the OPEN, CLOSE, DUMP, PDUMP, CANCEL, or CHKPT macro cannot be issued in the LABADDR routine. For a more complete discussion, see Label Processing.

```
CUAREA=name
```

This operand is used with INAREA for a combined file on an IBM 1442) that requires separate input and output areas. It specifies the symbolic name of the output area from which the updated card record is punched. If only one area is used for input and output, IOAREA1 should be used.

```
OUBLKSZ=n
```

This operand is used with OUAREA for a combined file. It is similar to INBLKSZ, and specifies the maximum number, n, of characters that are transferred from the output area (OUAREA) at any one time. If combined files use IOAREA1, BLKSIZE must be used.

```
PRINTOV=YES
```

This operand must be included whenever the PRTOV macro instruction is included in the problem program.

```
READ={FORWARD|BACK}
```

This operand may be included for a magnetic tape input file to specify the direction in which the tape is to be read. If this entry is omitted, IOCS assumes forward reading. One specification or the other is entered immediately after the = sign:

FORWARD For a tape read in the normal forward direction.

BACK For a tape read backwards.

```
RECFORM={FIXUNB|FIXBLK|VARUNB|VARBLK|
      UNDEF}
```

This operand specifies the type of records (fixed or variable length, blocked or unblocked, or undefined) in the input or output file. One of the following parameters may be entered immediately after the = sign:

FIXUNB For fixed-length unblocked records.

FIXBLK For fixed-length blocked records. This applies only to disk and magnetic tape input or output and optical reader journal tape input.

VARUNB For variable-length unblocked records. This applies only to disk input or output (2311), magnetic tape input or output (2400 or 3420), card punch output (1442, 2520, or 2540), and printer output (1403, 1404, 1443, 1445, or 3211).

VARBLK For variable-length blocked records. This applies only to disk and magnetic tape input or output.

UNDEF For undefined records. This applies to any file except card input (1442, 2501, 2520, or 2540).

The records in a file can be specified as:

- Disk and magnetic tape input or output. FIXUNB, FIXBLK, VARUNB, VARBLK, or UNDEF.
- Card input. FIXUNB.
- Card output. FIXUNB, VARUNB, or UNDEF.
- Optical reader input.
  All modes: FIXUNB or UNDEF.
  Journal tape mode: FIXBLK.
- Paper tape input. FIXUNB or UNDEF.
- Printer-keyboard input or output. FIXUNB or UNDEF.
- Printer output. FIXUNB, VARUNB, or UNDEF.

```
RECSIZE={n|(r)}
```

For input or output files, this operand must be included for disk, magnetic tape, and optical reader journal tape records that are fixed-length blocked (RECFORM=FIXBLK) or undefined (RECFORM=UNDEF). For paper tape records, this entry may be included for fixed-length unblocked or for undefined records (RECFORM=FIXUNB or =UNDEF). For other devices, this entry must be included whenever records are undefined (RECFORM=UNDEF).

For fixed-length blocked disk, magnetic tape or optical reader journal tape records, this entry specifies the number, n, of characters in an individual record. The input/output routines use this factor to block or deblock records, and to check record length of input records.

For undefined records, this entry specifies the number, (r), of the general-purpose register (2-12) that contains the length of each individual input or output record. When undefined records are read, IOCS supplies the physical record size in the register. In the case of paper tape records, this applies to both fixed unblocked and undefined records. When undefined records are built, the programmer must load the length of each record (in bytes) into the specified register before he issues the PUT instruction for the record. This becomes the count portion of the CCW that IOCS sets up for the file. Thus, it determines the length of the record to be transferred to an output device. If an undefined punch or printer output record contains a control character in the main-storage output area (DTFSR CTLCHR specified), the length loaded into the RECSIZE register must also include one byte for this character.

For undefined document records, RECSIZE contains only the length of the last field of a document read by the user-supplied channel command word chain.

> Note: When processing undefined records on an optical reader in document mode, the user gains complete usage of the two registers normally used in the RECSIZE operand. To do this, make sure that the suppress length indicator is always on when processing undefined records.

```
REWIND={UNLOAD|NORWD}
```

If no specifications are given by the programmer, tape files are automatically rewound, but not unloaded, on an OPEN or CLOSE instruction or on an end-of-volume condition. If other operations are desired for a tape input or output file, this entry may be included with one of the following entered immediately after the = sign:

UNLOAD — To rewind the tape on OPEN, and to rewind and unload on CLOSE or an end-of-volume condition.

NORWD — To prevent rewinding the tape at any time.

```
TPMARK=NO
```

This operand is included if the user does not want a tapemark written as the first record on a tape output file if labels are not specified. This entry is also included if no tapemark is written following nonstandard header labels. If this entry is omitted for a tape output file, a tapemark is the first record if no labels are specified. Also, if this entry is omitted, a tapemark is written following nonstandard header labels.

```
TRANS=name
```

This entry applies to an input file read from the IBM 2671 Paper Tape Reader, and it specifies the symbolic name of a code translation table. The table must conform to the specifications of the machine TRANSLATE instruction.

The input file records may be punched in 5-, 6-, 7-, or 8-channel paper tape, using any one of several different recording codes. If a code other than EBCDIC is used, it must be translated to EBCDIC code for use in System/360 programming. For IOCS to perform this translation, the user provides a translation table and specifies the symbolic name of the table in this TRANS entry. Then, the logical IOCS routines translate the paper tape code and make the record available to the programmer in usable form directly in the input area.

```
TRUNCS=YES
```

This operand applies to disk files with fixed-length blocked records (RECFORM=FIXBLK) when short blocks are processed. It must be included:

- For an output file if the TRUNC macro instruction is issued in the problem program.
- For an input file if the TRUNC macro was issued to write short blocks when the file was originally created.

```
TYPEFLE={INPUT|OUTPUT|CMBND}
```

This operand must be included to specify the type of file (input, output, or combined).

INPUT

Must be specified for:

- 2311 disk input files (with or without updating)
- 2400, 3420 magnetic tape input files
- 1442, 2501, 2520, 2540 card files
- 1052, 3210, 3215 keyboard input (both GET and PUT instructions may be issued)
- 1285 Optical Reader files
- 1285 Optical Reader files
- 1287, 1288 Optical Reader files

OUTPUT

Must be specified for:

- 2311 disk output files
- 2400, 3420 magnetic tape output files
- 1403, 1404, 1443, 1445, 3211 printer output
- 1442, 2520, 2540 card punch
- 1052, 3210, 3215 printer output (only PUT instructions may be issued).

CMBND

Must be specified for a 1442, 2520, or 2540 card file that is updated. That is, card records are read, processed, and then punched (PUT) in the same cards from which they were read. Thus, input and output operations are combined for the same file. This operation can be performed in the IBM 1442, 2520, or 2540 if the Punch-Feed-Read special feature is installed and cards are fed and read in the punch feed. (See PUT Macro: Updating.)

If this entry is omitted, INPUT is assumed.

```
UPDATE=YES
```

This operand must be included if a disk input file (TYPEFLE=INPUT) is updated. That is, disk records are to be read, processed, and then transferred back (PUT) to the same disk record locations from which they were read.

```
VARBLD=(r)
```

Whenever variable-length blocked records are built directly in the output area (no work area specified), this entry must be included. It specifies the number, (r), of a general purpose register (2-12), which always contains the length of the available space remaining in the output area.

After a PUT instruction is issued for a variable-length record, IOCS calculates the space still available in the output area and supplies it to the programmer in the designated register. The programmer then compares the length of his next variable length record with the available space to determine if the record fits in the area. This check must be made before the record is built. If the record does not fit, the programmer issues a TRUNC instruction to transfer the completed block of records to the tape file. Then, the present record is built at the beginning of the output area in the next block.

```
VERIFY=YES
```

This operand is included if the user wants disk records to be parity checked after they are written. If this entry is omitted, any records written on disk are not verified.

```
WLRERR=name
```

This operand applies only to disk, magnetic tape, or paper tape input files. It specifies the symbolic name of a user's routine to which IOCS branches if a wrong-length record is read. In his routine, the user can perform any operation he desires for wrong-length records. However, he must not issue any GET macro instructions for this file. Also, if he uses any other IOCS macros in his routine, he must save the contents of register 14. The address of the wrong-length record is supplied by IOCS in register 1. At the end of his routine, the user must return to IOCS by branching to the address in register 14.

Whenever fixed-length blocked records or variable-length records are specified (RECFORM=FIXBLK, =VARUNB, or =VARBLK), a machine check for wrong-length records is suppressed. In this case, IOCS generates a program check for the wrong record length. For fixed-length blocked records, record

length is considered incorrect if the physical disk or tape record (gap-to-gap) is not a multiple of the maximum logical record length specified in DTFSR RECSIZE. This permits the reading of short blocks of logical records, without a wrong length record indication.

For variable-length records on disk or tape, the record length is considered incorrect if it is not the same as the block length specified in the 4-byte block length field.

When fixed-length unblocked records are specified (RECFORM=FIXUNB), IOCS checks for a wrong-length-record indication that may result from an I/O operation.

If this WLRERR entry is omitted and a wrong-length record is detected by IOCS, one of the following conditions results:

- If the DTFSR ERROPT entry is included for this file, the wrong-length record is treated as an error block and handled according to the user's specifications for an error (IGNORE, SKIP, or name of error routine).
- If the DTFSR ERROPT entry is not included, the wrong-length record is ignored.

The WLRERR operand does not apply to undefined records. Undefined records are not checked for incorrect record length.

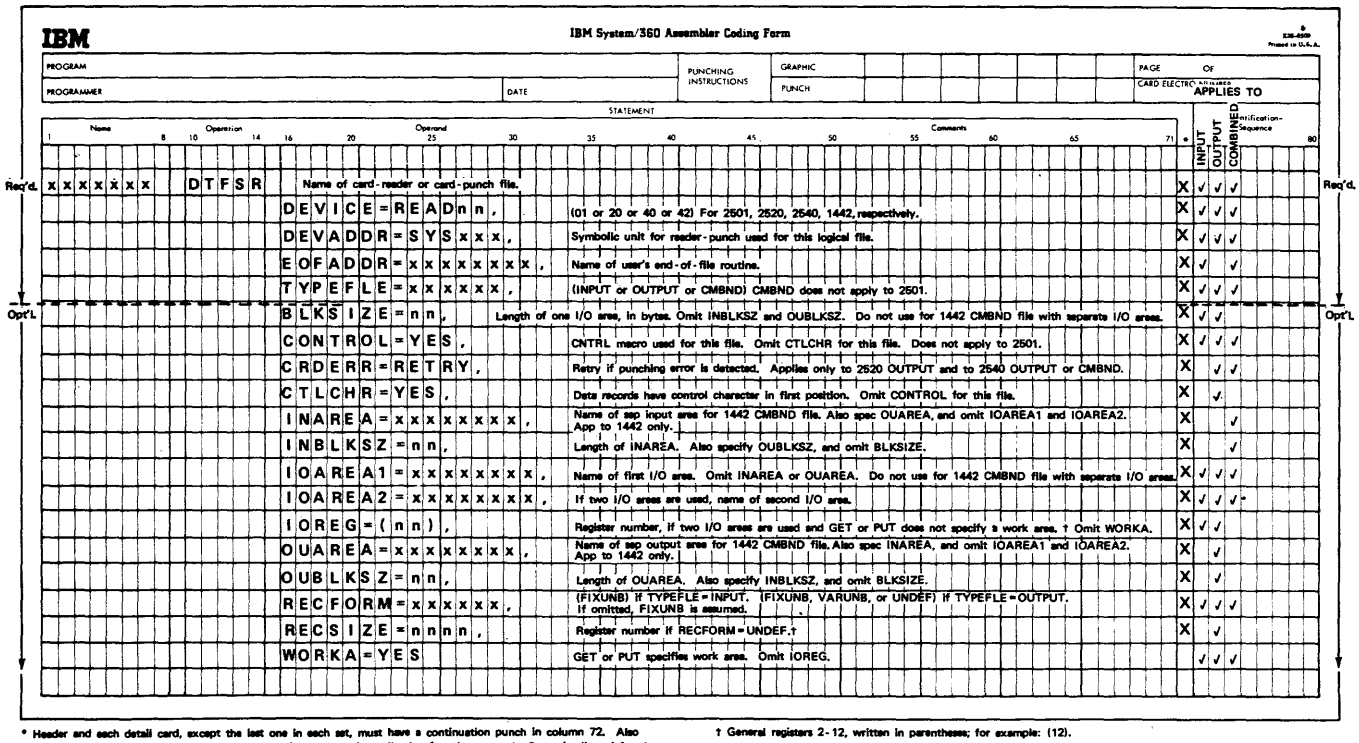
```
WORKA=YES
```

Input/output records can be processed, or built, in workareas instead of the input/output areas. If this is planned, WORKA=YES must be included, and the programmer must set up the workarea(s) in main storage. The symbolic name used in the DS instruction that reserves the workarea (or general register containing the address of the workarea) must be specified in each GET or PUT instruction. IOCS then moves the record to, or from, the specified workarea.

Whenever this entry is included for a file, the DTF entry IOREG must be omitted. Also, for optical character records, a workarea can only be used when processing journal tape.

The DTFEN Card

An end-of-definition card must follow the last set of DTFSR cards that applies to a magnetic tape file or to a DASD file. If two or more DTFSR macros are used in the same program, they may not be assembled separately from each other because duplicate labels may be generated. However, the set of DTFSR macros may be assembled separately from the user program. The DTFEN card must be punched with DTFEN in the operation field and blanks in the name field. The operand field may be blank or it may contain OVLAY as a parameter (to provide compatibility with BOS). DOS interprets the DTFEN card as a signal to begin generation of the required disk or tape I/O modules.



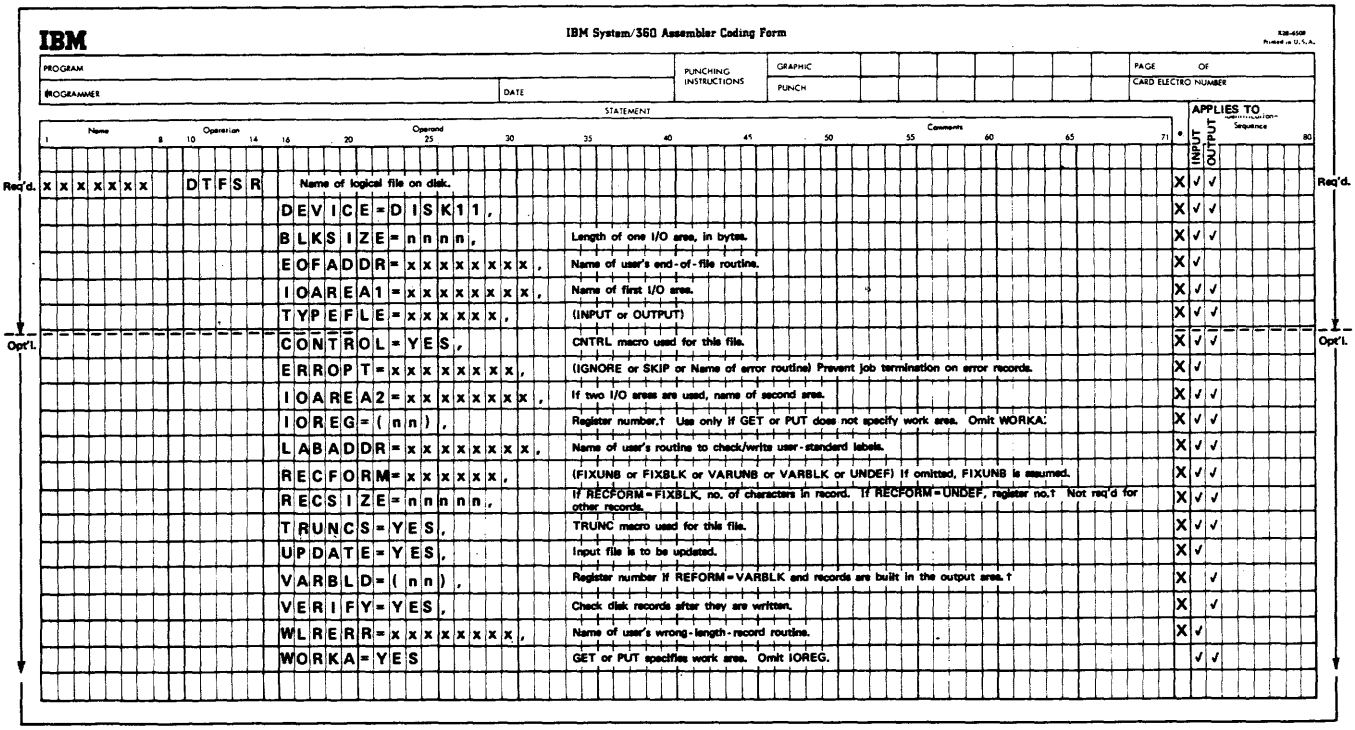
IBM System/360 Assembler Coding Form

Name	Operation	Operand	Comments	*	INPUT	OUTPUT	COMBINED
Req'd. xxxxxxx	DTFSR	Name of card-reader or card-punch file.			X	✓	✓	✓	Req'd.
	DEVICE=READnn,	(01 or 20 or 40 or 42) For 2501, 2520, 2540, 1442, respectively.			X	✓	✓	✓	
	DEVADDR=SYSxxx,	Symbolic unit for reader-punch used for this logical file.			X	✓	✓	✓	
	EOFADDR=xxxxxxxx,	Name of user's end-of-file routine.			X	✓		✓	
	TYPEFLE=xxxxxx,	(INPUT or OUTPUT or CMBND) CMBND does not apply to 2501.			X	✓	✓	✓	
Opt'L.	BLKSIZE=nn,	Length of one I/O area, in bytes. Omit INBLKSZ and OUBLKSZ. Do not use for 1442 CMBND file with separate I/O areas.			X	✓	✓		Opt'L.
	CONTROL=YES,	CNTRL macro used for this file. Omit CTLCHR for this file. Does not apply to 2501.			X	✓	✓	✓	
	CRDERR=RETRY,	Retry if punching error is detected. Applies only to 2520 OUTPUT and to 2540 OUTPUT or CMBND.			X		✓	✓	
	CTLCHR=YES,	Data records have control character in first position. Omit CONTROL for this file.			X		✓		
	INAREA=xxxxxxxx,	Name of sep input area for 1442 CMBND file. Also spec OUAREA, and omit IOAREA1 and IOAREA2. App to 1442 only.			X			✓	
	INBLKSZ=nn,	Length of INAREA. Also specify OUBLKSZ, and omit BLKSIZE.			X			✓	
	IOAREA1=xxxxxxxx,	Name of first I/O area. Omit INAREA or OUAREA. Do not use for 1442 CMBND file with separate I/O areas.			X	✓	✓	✓	
	IOAREA2=xxxxxxxx,	If two I/O areas are used, name of second I/O area.			X	✓	✓	✓	*
	IOREG=(nn),	Register number, if two I/O areas are used and GET or PUT does not specify a work area. † Omit WORKA.			X	✓	✓		
	OUAREA=xxxxxxxx,	Name of sep output area for 1442 CMBND file. Also spec INAREA, and omit IOAREA1 and IOAREA2. App to 1442 only.			X		✓		
	OUBLKSZ=nn,	Length of OUAREA. Also specify INBLKSZ, and omit BLKSIZE.			X		✓		
	RECFORM=xxxxxx,	(FIXUNB) If TYPEFLE=INPUT. (FIXUNB, VARUNB, or UNDEF) If TYPEFLE=OUTPUT. If omitted, FIXUNB is assumed.			X	✓	✓	✓	
	RECSIZE=nnnnn,	Register number if RECFORM=UNDEF.†			X		✓		
	WORKA=YES	GET or PUT specifies work area. Omit IOREG.				✓	✓	✓	

* Header and each detail card, except the last one in each set, must have a continuation punch in column 72. Also each detail card, except the last one, must contain a comma immediately after the operand. Space is allowed for the longest operand plus the comma. If a smaller operand is used, the comma should be moved over accordingly. In the last detail card of a set, the comma position must be blank.

† General registers 2-12, written in parentheses; for example: (12).

Figure 24. DTFSR Macro - Card (Part 1 of 5)



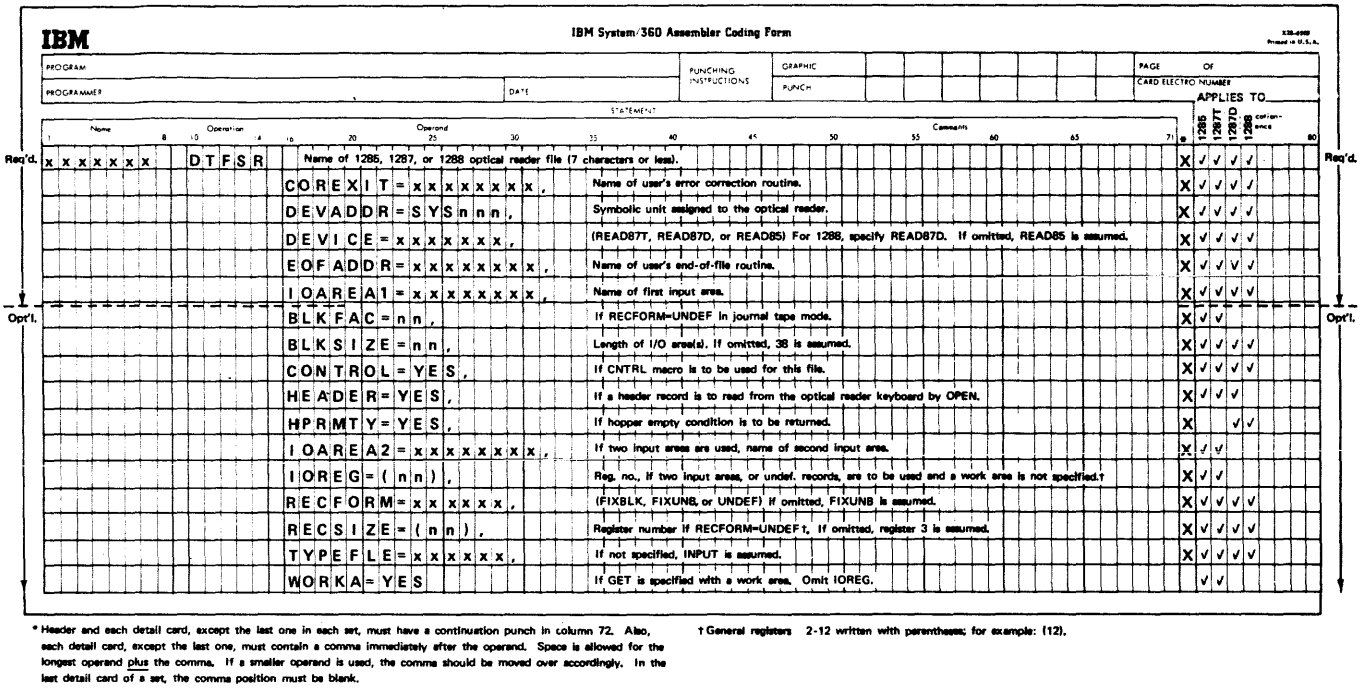
IBM System/360 Assembler Coding Form

Name	Operation	Operand	Comments	*	INPUT	OUTPUT
Req'd. xxxxxxx	DTFSR	Name of logical file on disk.			X	✓	✓	Req'd.
	DEVICE=DISK11,				X	✓	✓	
	BLKSIZE=nnnn,	Length of one I/O area, in bytes.			X	✓	✓	
	EOFADDR=xxxxxxxx,	Name of user's end-of-file routine.			X	✓		
	IOAREA1=xxxxxxxx,	Name of first I/O area.			X	✓	✓	
	TYPEFLE=xxxxxx,	(INPUT or OUTPUT)			X	✓	✓	
Opt'L.	CONTROL=YES,	CNTRL macro used for this file.			X	✓	✓	Opt'L.
	ERROPT=xxxxxxxx,	(IGNORE or SKIP or Name of error routine) Prevent job termination on error records.			X	✓		
	IOAREA2=xxxxxxxx,	If two I/O areas are used, name of second area.			X	✓	✓	
	IOREG=(nn),	Register number.† Use only if GET or PUT does not specify work area. Omit WORKA.			X	✓	✓	
	LABADDR=xxxxxxxx,	Name of user's routine to check/write user-standard labels.			X	✓	✓	
	RECFORM=xxxxxx,	(FIXUNB or FIXBLK or VARUNB or VARBLK or UNDEF) If omitted, FIXUNB is assumed.			X	✓	✓	
	RECSIZE=nnnnnn,	If RECFORM=FIXBLK, no. of characters in record. If RECFORM=UNDEF, register no.† Not req'd for other records.			X	✓	✓	
	TRUNCS=YES,	TRUNC macro used for this file.			X	✓	✓	
	UPDATE=YES,	Input file is to be updated.			X	✓		
	VARBLD=(nn),	Register number if REFORM=VARBLK and records are built in the output area.†			X		✓	
	VERIFY=YES,	Check disk records after they are written.			X		✓	
	WLRERR=xxxxxxxx,	Name of user's wrong-length-record routine.			X	✓		
	WORKA=YES	GET or PUT specifies work area. Omit IOREG.				✓	✓	

* Header and each detail card, except the last one in each set, must have a continuation punch in column 72. Also, each detail card, except the last one, must contain a comma immediately after the operand. Space is allowed for the longest operand plus the comma. If a smaller operand is used, the comma should be moved over accordingly. In the last detail card of a set, the comma position must be blank.

† General registers 2-12 written with parentheses, for example: (12).

Figure 24. DTFSR Macro - 2311 Disk (Part 2 of 5)



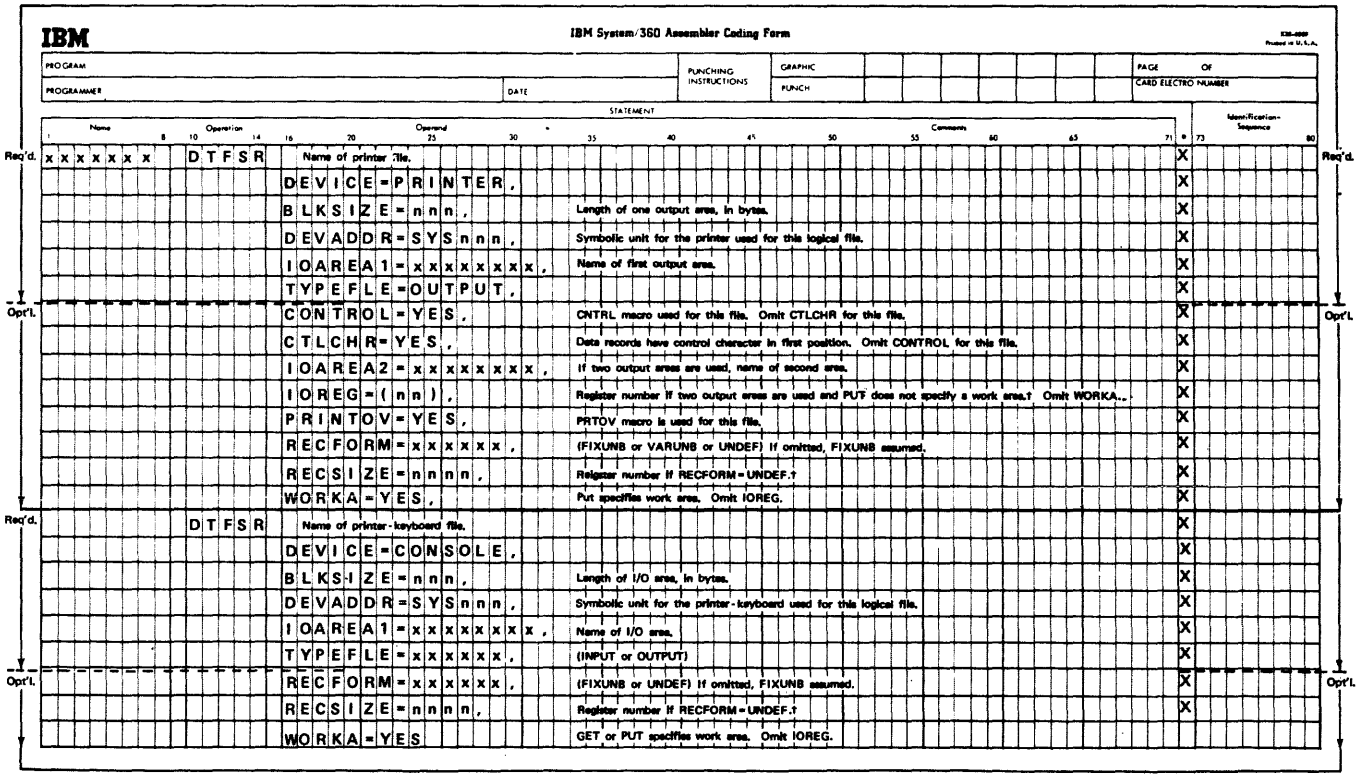
IBM System/360 Assembler Coding Form

Name	Operation	Operand	Comments	1285	1287	1287D	1288
Req'd. xxxxxx	DTFSR	Name of 1285, 1287, or 1288 optical reader file (7 characters or less).		X	✓	✓	✓	✓	Req'd.
		COREXIT=xxxxxxxx,	Name of user's error correction routine.	X	✓	✓	✓	✓	
		DEVADDR=SYSnnn,	Symbolic unit assigned to the optical reader.	X	✓	✓	✓	✓	
		DEVICE=xxxxxxx,	(READ87T, READ87D, or READ85) For 1288, specify READ87D. If omitted, READ85 is assumed.	X	✓	✓	✓	✓	
		EOFADDR=xxxxxxxx,	Name of user's end-of-file routine.	X	✓	✓	✓	✓	
		IOAREA1=xxxxxxxx,	Name of first input area.	X	✓	✓	✓	✓	
Opt'l.		BLKFAC=nn,	If RECFORM=UNDEF in journal tape mode.	X	✓	✓			Opt'l.
		BLKSIZE=nn,	Length of I/O area(s). If omitted, 38 is assumed.	X	✓	✓	✓	✓	
		CONTROL=YES,	If CNTRL macro is to be used for this file.	X	✓	✓	✓	✓	
		HEADER=YES,	If a header record is to read from the optical reader keyboard by OPEN.	X	✓	✓	✓		
		HPRMTY=YES,	If hopper empty condition is to be returned.	X			✓	✓	
		IOAREA2=xxxxxxxx,	If two input areas are used, name of second input area.	X	✓	✓			
		IOREG=(nn),	Reg. no., if two input areas, or undef. records, are to be used and a work area is not specified.†	X	✓	✓			
		RECFORM=xxxxxx,	(FIXBLK, FIXUNB, or UNDEF) If omitted, FIXUNB is assumed.	X	✓	✓	✓	✓	
		RECSIZE=(nn),	Register number if RECFORM=UNDEF†, If omitted, register 3 is assumed.	X	✓	✓	✓	✓	
		TYPEFLE=xxxxx,	If not specified, INPUT is assumed.	X	✓	✓	✓	✓	
		WORKA=YES	If GET is specified with a work area. Omit IOREG.		✓	✓			

* Header and each detail card, except the last one in each set, must have a continuation punch in column 72. Also, each detail card, except the last one, must contain a comma immediately after the operand. Space is allowed for the longest operand plus the comma. If a smaller operand is used, the comma should be moved over accordingly. In the last detail card of a set, the comma position must be blank.

† General registers 2-12 written with parentheses; for example: (12).

Figure 24. DTFSR Macro - Optical Reader (Part 3 of 5)



IBM System/360 Assembler Coding Form

Name	Operation	Operand	Comments	
Req'd. xxxxxx	DTFSR	Name of printer file.		X	Req'd.
		DEVICE=PRINTER,		X	
		BLKSIZE=nnn,	Length of one output area, in bytes.	X	
		DEVADDR=SYSnnn,	Symbolic unit for the printer used for this logical file.	X	
		IOAREA1=xxxxxxxx,	Name of first output area.	X	
		TYPEFLE=OUTPUT,		X	
Opt'l.		CONTROL=YES,	CNTRL macro used for this file. Omit CTLCHR for this file.	X	Opt'l.
		CTLCHR=YES,	Data records have control character in first position. Omit CONTROL for this file.	X	
		IOAREA2=xxxxxxxx,	If two output areas are used, name of second area.	X	
		IOREG=(nn),	Register number if two output areas are used and PUT does not specify a work area.† Omit WORKA.	X	
		PRINTOV=YES,	PRTOV macro is used for this file.	X	
		RECFORM=xxxxxx,	(FIXUNB or VARUNB or UNDEF) If omitted, FIXUNB assumed.	X	
		RECSIZE=nnnnn,	Register number if RECFORM=UNDEF.†	X	
		WORKA=YES,	Put specifies work area. Omit IOREG.	X	
Req'd.	DTFSR	Name of printer-keyboard file.		X	
		DEVICE=CONSOLE,		X	
		BLKSIZE=nnn,	Length of I/O area, in bytes.	X	
		DEVADDR=SYSnnn,	Symbolic unit for the printer-keyboard used for this logical file.	X	
		IOAREA1=xxxxxxxx,	Name of I/O area.	X	
		TYPEFLE=xxxxxx,	(INPUT or OUTPUT)	X	
Opt'l.		RECFORM=xxxxxx,	(FIXUNB or UNDEF) If omitted, FIXUNB assumed.	X	Opt'l.
		RECSIZE=nnnnn,	Register number if RECFORM=UNDEF.†	X	
		WORKA=YES	GET or PUT specifies work area. Omit IOREG.		

* Header and each detail card, except the last one in each set, must have a continuation punch in column 72. Also each detail card, except the last one, must contain a comma immediately after the operand. Space is allowed for the longest operand plus the comma. If a smaller operand is used, the comma should be moved over accordingly. In the last detail card of a set, the comma position must be blank.

† General registers 2-12, written in parentheses; for example: (12).

Figure 24. DTFSR Macro - Printer/Printer Keyboard (Part 4 of 5)

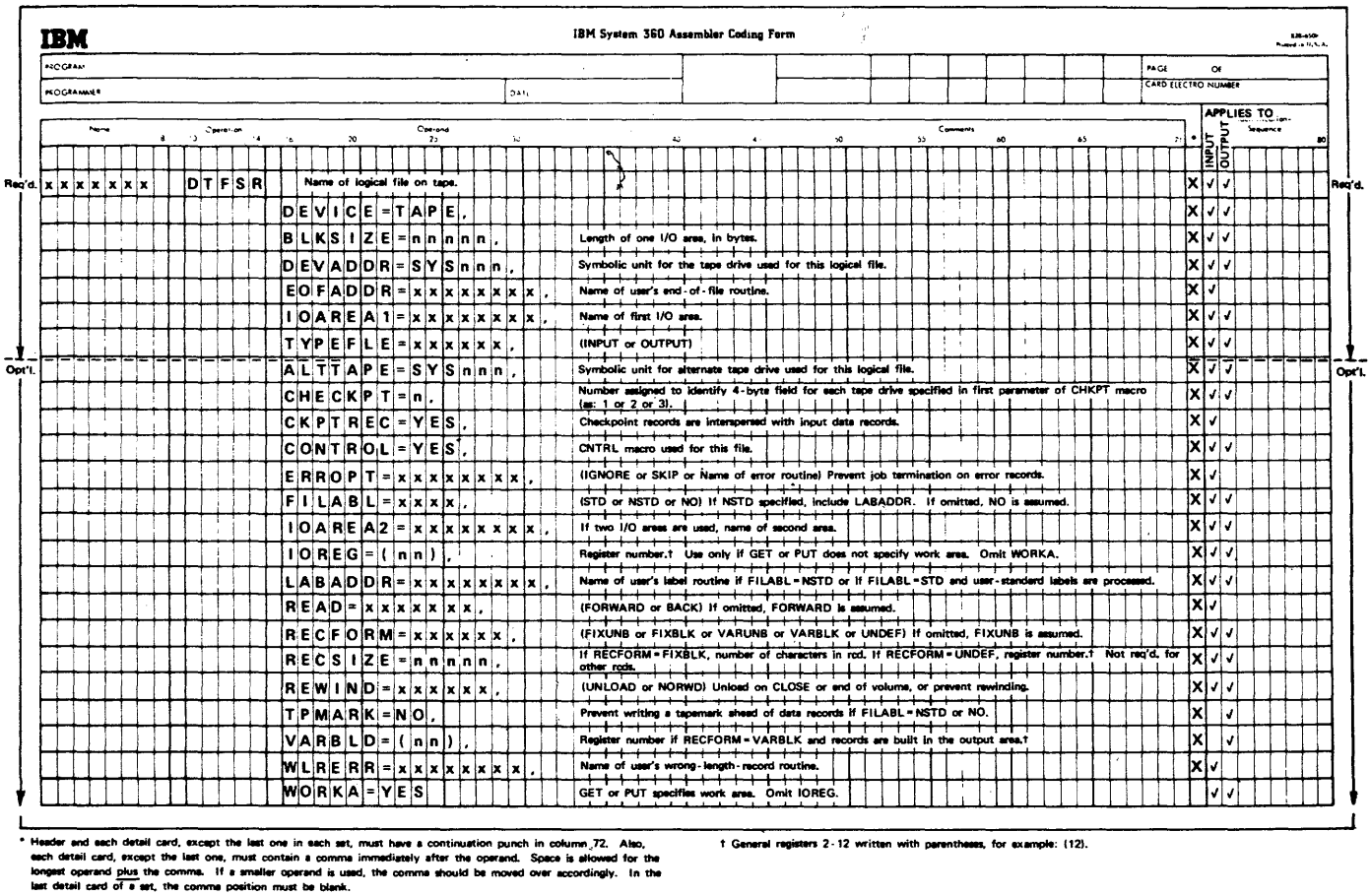


Figure 24. DTFSR Macro - Tape (Part 5 of 5)

Imperative Macro Instructions

After the files are defined by the declarative macro instructions, the following groups of imperative macros can be applied to open, process, and close the sequential files involved.

- Initialization Macros
- Sequential Processing Macros
- Completion Macros

For a guideline of macros to use for a particular I/O file, see Figure 7.

INITIALIZATION MACROS

Before processing a logical file, ready it for use by issuing an OPEN(R) macro. OPEN need not be used with DTFCN and DTFPT files, however. OPEN(R) optionally checks or writes standard labels or nonstandard labels.

Information on labels is contained in the Label Processing section, in Appendix A, and in the Data Management Concepts publication.

OPEN(R) Macro

Op	Operand
for self-relocating programs	
OPENR	{filename1 / (r1)} [,{filename2 / (r2)}...,{filenamen / (rn)}]
for programs that are not self-relocating	
OPEN	filename1 / (r1) [,{filename2 / (r2)}...,{filenamen / (rn)}]

Note: To write the most efficient code (in a multiprogramming environment), we recommend that the self relocating form of OPEN be used. (See also Appendix G)

The OPEN macro instruction activates all files processed with LIOCS.

When the operation OPEN is used, the symbolic address constants that OPEN generates from the parameter list are not self-relocating. When OPENR is specified, the symbolic address constants are self-relocating.

****

Restriction: Self-relocating programs using LIOCS must use the OPENR macro instruction to activate all files, including printer keyboard files. The OPENR macro, in addition to activating files for processing, relocates all address constants within the DTF tables.

Note: Zero constants are not relocated except when they constitute the module address.

****

If OPEN attempts to activate a logical IOCS file (DTF) whose device is unassigned, the job is terminated. If the device is assigned IGN, OPEN(R) does not activate the file but turns on the DTF byte 16, bit 2, to indicate the file is not activated. If DTF byte 16 bit 2 is ON after the file is opened, input/output operations should not be performed on this file.

The symbolic name of the file (DTF filename) is entered in the operand field. A maximum of 16 files may be opened with one OPEN (or OPENR) by entering the filenames as additional operands. Alternately, the user can load the address of the DTF filename in a register and specify the register using ordinary register notation. The high-order 8 bits of this register must be zeros. Filename may be preloaded into any register, 0-15.

Notes: If you use register notation, we recommend using only registers 2-12. This will make your programs more compatible with the Operating System (OS). If it is necessary to open a sequential DASD file more than once, then the DTF must be restored to its original state.

Whenever an input/output DASD or magnetic tape file is opened and the user plans to process user-standard labels (UHL or UTL), or nonstandard tape labels, he must provide the information for checking or building the labels. If this information is obtained from another input file, that file must also be opened, if necessary, ahead of the DASD or tape file. To do this, specify the input file ahead of the tape or DASD file in the same OPEN, or issue a separate OPEN for the file.

If an output tape (specified to contain standard labels) is opened that does not contain a volume label, an operator message is issued. This message gives the operator the opportunity to type a volume serial number so that a volume label can be written on the output tape.

When opening other files such as the card reader, card punch, MICR document reader, paper tape reader, paper tape punch, printer, and printer-keyboard, OPEN simply makes the file available for input or output. For a paper tape punch file with two I/O areas, OPEN loads the user's IOREG with the address of an I/O area.

If the user wishes, a DTFCN (console) file can be opened if device type checking is desired. The unit record OPEN permits the logical unit assignment to be either to a 1052 printer-keyboard or to a printer. An assignment to any device other than these two is invalid and the job is canceled.

For 1403 printers with the Universal Character Set feature, data checks are ignored and blanks are printed unless the user specifies UCS=ON in the DTFPR.

For MICR devices, the OPEN macro sets the entire I/O area to binary zeros.

When LIOCS is used for processing journal tapes on the IBM 1285 Optical Reader, the OPEN macro must be issued at the beginning of each input roll. LIOCS journal tape processing on the 1287 may be done in the same way.

To process two or more rolls on the 1287 as one file (when an end-of-tape condition occurs), run out the tape by pressing the start key on the optical reader. This creates an intervention-required condition instead of the end-of-file key. The next tape can then be loaded and processed as a continuation of the previous tape. However, because OPEN is not reissued, no header information can be entered between tapes.

When processing documents on the 1287, OPEN must be issued to make the file available. If the program is to be self-relocatable, OPENR must be used and for any CCW chain the user writes, addressability must be provided for his data addresses.

OPEN allows header (identifying) information to be entered at the 1285 or 1287 keyboard, for journal tape or cut documents (1287). When header information is entered, it is always read into IOAREA1, which must be large enough to accommodate the desired header information.

DASD OUTPUT: When a multivolume DASD file is created using sequential processing, only one extent is processed at a time. Therefore, only one pack need be mounted at a time. When processing on a volume is completed, message

4n55A WRONG PACK, MOUNT nnnnnn

will be issued so that the next volume may be mounted.

When a file is opened, OPEN checks the standard VOL1 label and the specified extents:

1. The extents must not overlap each other.
2. The first extent must be at least two tracks long if user standard labels are created.
3. Only extent types 1 and 8 are valid.

The data extents of a sequential DASD file can be type 1, type 8, or both. Type 8 extents are called split cylinder extents and use only a portion of each cylinder in the extent. The portion of the cylinder used must be within the head limits of the cylinder and within the range of the defined extent limits. For example, two files can share three cylinders--one file occupying the first two tracks of each cylinder and the other file occupying the remaining tracks. In some applications, the use of split cylinder files reduces the access time.

OPEN checks all the labels in the VTOC to ensure that the file to be created does not destroy an existing file whose expiration date is still pending. It also checks to determine that the extents do not overlap existing extents. After the VTOC checks, OPEN creates the standard label(s) for the file and writes the label(s) in the VTOC.

If the user wishes to create his own user standard (UHL or UTL) labels for the file, he must include the DTF entry LABADDR. OPEN reserves the first track of the first extent for the user header and trailer labels. Then, the user's label routine is given control at the address specified in LABADDR.

After the header labels are built, the first extent of the file is ready to be used. The extents are made available in the order of the sequence numbers on the actual extent statements. When the last extent on the mounted volume is filled, user standard trailer labels can be built. Then, the next specified volume in the extent statements is mounted and opened.

For a file-protected DASD, when OPEN makes the first extent of the new volume available, it makes the extent(s) from the previous volume unavailable. When the last extent on the final volume of the file is processed, OPEN issues an operator message. The operator has the option of canceling the job or typing in an extent on the printer-keyboard and continuing the job.

DASD INPUT: In a multivolume file (a file having extents on more than one disk pack), only one extent is processed at a time, and thus, only one pack need be mounted at a time. When processing on a volume is completed, message

4n55A WRONG PACK, MOUNT nnnnnn

will be issued so that the next volume may be mounted.

When a volume is opened, OPEN checks the standard VOL1 label and goes to the VTOC to check the file label(s). OPEN checks the specified extents in the extent statements against the extents in the labels to make sure the extents exist. If LABADDR is specified, OPEN makes the user standard header (UHL) labels available to the user for checking (one at a time).

After the labels are checked, the first extent of the file is ready to be processed. The extents are made available in the order of the sequence number on the extent statements. The same extent statements that were used to build the file can be used when the file is used as input.

Note: If EXTENT cards with specified limits are included in the job stream, or if an EXTENT was created by replying with an extent to message

4450A NO MORE AVAILABLE EXTENTS

when the file was built, then an additional EXTENT card must be submitted on input to process that extent. If no EXTENT cards are submitted, however, this additional extent is processed normally.

When the last extent on the mounted volume is processed, the user standard trailer labels are made available for checking one at a time. The next volume is then opened.

For DASD devices that are file protected when OPEN makes the first extent of the new volume available, OPEN makes the extent(s) from the previous volume unavailable.

LBRET Macro

Name	Operation	Operand
[name]	LBRET	{1,2,3}

The LBRET macro is issued in user subroutines when the user has completed processing labels and wishes to return control to IOCS. LBRET applies to subroutines that write or check DASD or magnetic tape user standard labels, write or check tape nonstandard labels, or check DASD extents. The operand used depends on the function to be performed. See Label Processing.

CHECKING USER STANDARD DASD LABELS: IOCS passes the labels to the user one at a time until the maximum allowable number is read (and updated), or until the user signifies he wants no more. In his label routine, the user issues LBRET 3 if he wants IOCS to update (rewrite) the label just read and pass him the next label. LBRET 2 is issued if he simply wants IOCS to read and pass him the next label. If an end-of-file record is read when LBRET 2 or LBRET 3 is used, label checking is automatically ended. If the user wants to eliminate the checking of one or more remaining labels, LBRET 1 should be issued.

WRITING USER STANDARD DASD LABELS: The user builds the labels one at a time and uses LBRET to return to IOCS, which writes the labels. LBRET 2 is used if the user wants control returned to him after IOCS writes the label. If, however, IOCS determines that the maximum number of labels has already been written, label processing is terminated. LBRET 1 is used if the user wishes to stop writing labels before the maximum number of labels is written.

CHECKING USER STANDARD TAPE LABELS: IOCS reads and passes the labels to the user one at a time until a tapemark is read, or until the user signifies he does not want any more labels. LBRET 2 is used if the user wants to process the next label. If IOCS reads a tapemark, label processing is automatically terminated. LBRET 1 is used if the user wants to bypass any remaining labels.

WRITING USER STANDARD TAPE LABELS: The user builds the labels one at a time and returns to IOCS, which writes the labels. When LBRET 2 is used, IOCS returns control to the user (at the address specified in LABADDR) after writing the label. LBRET 1 must be used to terminate the label set.

WRITING OR CHECKING NONSTANDARD TAPE LABELS: The user must process all his nonstandard labels at once. LBRET 2 is used after all label processing is completed and the user wants to return control to IOCS. Appendix D shows an example of this.

Sequential Processing Macros

The sequential processing macro instructions permit the programmer to store and retrieve records without coding blocking/deblocking routines. The programmer can, therefore, concentrate on processing his data. Another major feature of these macro instructions is the ability to use one or two I/O areas and to process records in either a workarea or an I/O area.

The sequential processing routines provide for overlapping the physical transfer of data with processing. The amount of overlapping actually achieved is governed by the problem program through the assignment of I/O areas and workareas. An I/O area is that area of main storage to or from which a block of data is physically transferred by logical IOCS. A workarea is an area used for processing an individual logical record from the block of data. A workarea cannot be used with paper tape records. The I/O area(s) is specified in the associated DTF macro, while the

workarea is specified in the sequential processing macro.

The following combinations of I/O areas and workareas are possible:

1. One I/O area without a workarea
2. One I/O area with a workarea
3. Two I/O areas without a workarea
4. Two I/O areas with a workarea

When processing spanned records, the user may use:

1. One I/O area with a workarea, or
2. Two I/O areas with a workarea.

Although two I/O areas are permitted, normal overlap is curtailed because each I/O command from the user may require multiple I/O operations by MTMCD.

GET Macro

Name	Operation	Operand
[name]	GET	{filename (1)} [,{workname (0)}]

****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

GET makes the next sequential logical record from an input file available for processing in either an input area or a specified workarea. It is used for any input file in the system, and for any type of record: blocked or unblocked, fixed or variable length, and undefined.

If GET is used with a file containing checkpoint records, the checkpoint records are bypassed automatically.

Filename: GET requires the first operand. The parameter value must be the same as specified in the header entry of the DTF for the file from which the record is to be retrieved. The filename can be specified as a symbol or in either special or ordinary register notation. The latter is preferable (see Recommendation).

Note: The register used must have a high order byte of zero.

Workname: This is an optional parameter specifying the workarea name or a register (in either special or ordinary register notation) containing the address of the workarea. The workarea address should never be preloaded into register 1. This parameter is used if records are to be processed in a workarea that the user himself defines (for example, using a DS instruction). If the operand is specified, all GETs to the named file must always use a register or a workname. Using the second operand causes GET to move each individual record from the input area to a workarea.

All records from a logical file may be processed in the same workarea, or different records from the same logical file may be processed in different workareas. In the first case, each GET for a file specifies the same workarea. In the second case, different GET instructions specify different workareas. It might be advantageous to plan two workareas, for example, and to specify each area in alternate GET instructions. This permits the programmer to compare each record with the preceding one. In this manner, he checks for a change in a control field within the record. However, only one workarea can be specified in any one GET instruction.

Required DTF Entries

The input area must be specified in the entry IOAREA1 of the DTF macro. For any file other than a combined file, two input areas may be used to permit an overlapping of data transfer and processing operations. The second area is specified in IOAREA2. Whenever two input areas are specified, the IOCS routines transfer records alternately to each area. They completely handle this flip-flop so that the next sequential record is always available to the problem program for processing.

For a combined file, the input area is specified in IOAREA1 and the output area in IOAREA2. If the same area is used for both input and output, IOAREA2 is omitted.

When records are processed in the input area(s), a register must be specified in the entry IOREG of the DTF macro if:

1. Records are blocked, or

2. Undefined or variable-length magnetic tape records are read backwards, or

3. Two input areas are used, for either blocked or unblocked records, or

4. Neither BUFOFF=0 nor WORKA=YES is specified for ASCII files

This register identifies the next single record to be processed. It always contains the absolute address of the record currently available. The GET routine places the proper address in the register.

If a workarea is used, WORKA=YES must be specified. IOREG should not be specified.

When the GET macro detects an end-of-file condition, IOCS branches to the user's end-of-file routine (specified by EOFADDR). For MICR document processing, the user does not regain control until either a buffer becomes filled with a stacker selected document, or error conditions are posted in the buffer status indicators.

An example of GET/PUT processing is shown in the following coding. The parameter IOAREA1 points to the first I/O area for this file. IOAREA2 points to the second I/O area. GET points to the file-definition block and to the workarea (A3) to which logical records are moved from areas A1 and A2 by LIOCS.

```
Name      Operation   Operand          Col. 72

          ┌──────────────────────────┐
          │                          │
FNAME<-┘  DTFMT                     │
          .                          │
          ┌----------> IOAREA1=A1,   |  X
          |┌---------> IOAREA2=A2,   |  X
          ||.          WORKA=YES     |
          ||.                        |
A1   <-┘|DS           500C          |
A2   <--┘DS           500C          |
          .                          |
          .                          |
          .            ┌----------┘
          .            V
          GET          FNAME,A3
          .                  ^
          .                  |
A3   <-┐  DS           100C  |
       |  .                  |
       |                     |
       └-------------------┘
```

Unblocked Records

Records retrieved from any input file are considered fixed unblocked unless otherwise specified.

Whenever records are unblocked (either fixed or variable length) and only one input area is used, each GET transfers a single record from an I/O device to the input area. The record is then transferred to a workarea if one is specified in the GET instruction. If two input areas are specified, each GET makes the last record that was transferred to main storage available for processing in the input area or workarea. The same GET also starts the transfer of the following record to the other input area.

When an IBM 2540 Card Read Punch is used for a card input file, each GET instruction normally reads the record from a card in the read feed. However, if the 2540 has the Punch-Feed-Read special feature installed and if CMBND is specified in the entry TYPEFLE, each GET reads the record from a card in the punch feed, at the punch-feed-read station. This record can be updated with additional information that is then punched back into the same card when the card passes the punch station and a PUT instruction is issued. (See Put: Macro Updating.)

Blocked Records

When records on DASD or magnetic tape are specified as blocked in the entry RECFORM, each individual record must be located for processing (deblocked). Therefore, blocked records (either fixed or variable length) are handled as follows:

1. The first GET instruction transfers a block of records from DASD or tape to the input area. It also initializes the specified register to the absolute address of the first data record, or it transfers the first record to the specified workarea.

2. Subsequent GET instructions either add an indexing factor to the register or move the proper record to the specified workarea, until all records in the block are processed.

3. Then, the next GET makes a new block of records available in main storage, and either initializes the register or moves the first record.

Spanned Records

When unblocked or blocked spanned records are processed, the entry RECFORM=SPNUNB or RECFORM=SPNBLK, respectively, must be included in both the file definition (DTFMT, DTFSD) and the appropriate module (SDMODVI, SDMODVU, or MTMOD). GET assembles spanned record segments into logical records in the user's work area. Null segments are recognized and skipped. They are not assembled into logical records. The length of the logical record passes to the user through a register specified under RECSIZE in the DTF.

If the user chooses to update logical records using SDMODVU, the pointer to the physical record in which a logical record starts is saved on each GET so that the device may be repositioned. The extent sequence number (byte 40 of the DTF) is also saved in case the logical record spans disk extents.

Undefined Records

When undefined records are handled, the entry RECFORM=UNDEF must be included in the file definition. GET treats undefined records as unblocked, and the programmer must locate individual records and fields. If a RECSIZE register is specified, IOCS stores the length of the record read in that register. Undefined records are considered to be variable in length by IOCS. No other characteristics of the record are known by IOCS. They are the responsibility of the user.

Read Backwards, Tape

If records on magnetic tape are read backwards (BACK specified in entry READ), blocks of fixed-length records, blocks of blocked-variable records, or unblocked records, are transferred from tape to main storage in reverse order. The last block is read first, the next-to-last block, second, etc. For blocked records, each GET instruction also makes the individual records available in reverse order. The last record in the input area is the first record available for processing (either by indexing or in a workarea).

Any 9-track tape can be read backwards. 7-track tape can be read backwards only if the data conversion special feature was not used when the tape was written.

PUT Macro

Name	Operation	Operand
[name]	PUT	{filename / (1)} [,{workname / (0)}] [, {STLSP=controlfield / (r)}] [, {STLSK=controlfield / (r)}]

****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

PUT writes or punches logical records that are built directly in the output area or in a specified workarea. It is used for any output file in the system, and for any type of record: blocked or unblocked, fixed or variable length, and undefined. It operates much the same as GET but in reverse. It is issued after a record has been built.

When a PUT is issued, the printer automatically spaces one line. Neither the CNTRL macro nor a control character need be specified.

Filename: PUT requires the first operand. The parameter value must be the same as specified in the header entry of the DTF for the file being built. The filename can be specified as a symbol or in either special or ordinary register notation. The latter is preferable (see Recommendation).

Workname: An optional parameter specifying the workarea name or a register (in either special or ordinary register notation) containing the address of the workarea. The workarea address should never be

preloaded into register 1. This parameter is used if records are built in a workarea that the user himself defines (for example, using a DS instruction). If the operand is specified, all PUTs to the named file must always use a register or a workname. Using the second operand causes PUT to move each record from the workarea to the output area.

Individual records for a logical file may be built in the same workarea or in different workareas. Each PUT instruction specifies the workarea where the completed record was built. However, only one workarea can be specified in any one PUT instruction.

Whenever an output data record is transferred from an output area (or workarea) to an I/O device (by a PUT instruction), the data remains in the area until it is either cleared or replaced by other data. IOCS does not clear the area. Therefore, if the user plans to build another record whose data does not use every position of the output area or workarea, he must clear that area before he builds the record. If this is not done, the new record will contain interspersed characters from the preceding record. For example, in the case of output to a printer, the forms design may require printing in selected positions on one print line and in different positions on another line. In this case, the output area or workarea for the printer file should be cleared between lines.

STLSP=control field: This optional operand specifies a one-byte control field that allows for spacing while using the Selective Tape Listing feature on the 1403 printer. You can also use ordinary register notation to provide the address of the control field. Registers 2 through 12 are available without restriction. The spacing (after printing occurs) is set by the bit configuration of ones in the one-byte control field as follows:

Data Byte Bits	0	1	2	3	4	5	6	7
Tape Position	8	7	6	5	4	3	2	1

The tape position 1 is the leftmost tape on the selective tape listing device.

Note: Double width tapes must be controlled by both bits of the control field.

STLSK=control field: This optional operand specifies a one-byte control field that allows for skipping while using the selective tape listing feature on the 1403 printer. You can also use ordinary register notation to provide the address of the control field. Registers 2 through 12 are available without restriction. The skipping (after printing occurs) is set by the bit configuration in a one-byte control field.

Selective tape listing support in the DTFPR declarative macro instruction allows the user to independently space or skip up to eight paper tapes. This feature is widely used in conjunction with MICR document processing.

Required DTF Operands

The output area must be specified in the entry IOAREA1 of the DTF macro. For any file other than a combined file, two output areas may be used to permit an overlapping of data transfer and processing operations. The second area is specified in IOAREA2. Whenever two output areas are specified, the IOCS routines transfer records alternately from one to the other area. The routines completely handle this flip-flop, so that the proper output record area is always available to the program.

For a combined file, the input area is specified in IOAREA1 and the output area in IOAREA2. If the same area is used for both input and output, IOAREA2 is omitted.

When records are built in the output area(s), a register must be specified in the entry IOREG if:

1. Records are blocked, or
2. Two output areas are used for either blocked or unblocked records.

This register always contains the absolute base address of the currently available output-record area. IOCS places the proper address in the register. The user should always address the I/O areas by using the IOREG as the base register and should not assume which I/O area is presently being used. If a workarea is used, WORKA=YES must be specified; IOREG should not be specified.

If blocked records are variable length and are built in the output area(s), an additional register must be specified in the entry VARELD. IOCS stores the number

of bytes remaining in the output area in the VARBLD register each time a PUT instruction is executed.

Unblocked Records

Records transferred to any output file except DASD or magnetic tape are always considered fixed unblocked unless otherwise specified. Records for DASD or tape output are treated as unblocked if this is specified in the entry RECFORM.

Whenever records are unblocked (either fixed or variable length), each PUT transfers a single record from the output area (or input area if updating is specified) to the file. If a workarea is specified in the PUT instruction, the record is first moved from the workarea to the output area (or input area) and then to the file. For fixed DASD unblocked records, IOCS follows the rule that if there is not enough space for another record in the extent specified, then there is not enough space for an EOF record.

Blocked Records

When blocked records are written on DASD or magnetic tape, the individually built records must be formed into a block in the output area. Then, the block of records is transferred to the output file. The blocked records may be either fixed or variable length.

Fixed-length blocked records can be built directly in an output area or in a workarea. Each PUT instruction for these records either adds an indexing factor to the register (IOREG), or moves the completed record from the specified workarea to the proper location in the output area. When an output block of records is complete, a PUT instruction causes the block to transfer to the output file and initializes the register, if one is used.

Variable-length blocked records can also be built in either an output area or a workarea. The length of each variable length record must be determined by the problem program and included in the output record as it is built. The problem program can calculate the length of the output record from the length of the corresponding input records. That is, variable length output records are generally developed from previously written variable length input records. Each variable length input record must include the field that contains the length of the record.

When variable-length blocked records are built in a workarea, the PUT instruction performs the same functions as it does for fixed-length blocked records. The PUT routines check the length of each output record to determine if the record fits in the remaining portion of the output area. If the record fits, PUT immediately moves the record. If it does not fit, PUT causes the completed block to be written and then moves the record.

However, if variable-length blocked records are built directly in the output area, the VARBLD entry, the TRUNC macro, and additional user programming are required. The user's program must determine if each record built will fit in the remaining portion of the output area. This must be known before record processing for a subsequent record begins, so that the completed block can be written. Thus, the length of the record must be precalculated and compared with the amount of remaining space.

The amount of space available in the output area at any time can be supplied to the program (in a register) by the IOCS routines. For this, the user must specify a general purpose register in the DTF entry VARBLD. This register is in addition to the register specified in IOREG. Each time a PUT instruction is executed, IOCS loads into the specified register the number of bytes remaining in the output area. The problem program uses this to determine if the next variable-length record will fit. If it will not fit, a TRUNC macro instruction must be issued to transfer the block of records to the output file. The entire output area is then available for building the next block.

Spanned Records

When PUT handles unblocked or blocked spanned records, the entry RECFORM=SPNUNB or RECFORM=SPNBLK, whichever applies, must be included in both the file definition (DTFMT, DTFSD) and the appropriate module (SDMODVO, SDMODVU, or MTMOD). Logical records in the user's workarea are divided into spanned record segments according to the length specified in the RECSIZE parameters. In constructing the segments, full use is made of the space available in each physical record and device track. On output, spanned records do not span volumes. If there is not enough space on the current volume to contain a spanned record, SDMODVO:

1. Rereads the last block of the previous spanned record.

2. Rewrites the last block (truncated to the last segment of the previous spanned record, if necessary) to erase the remainder (if any) of the track;

3. Writes an eight-byte record-block descriptor word and one null segment on each remaining track on the current volume.

4. Attempts to put the entire spanned record on the next volume.

For update files, SDMODVU repositions the device to the first block of the logical record by using the pointer saved in GET processing. If the logical record spans extents, the extent sequence number that was also saved in GET processing is used to ensure that updating starts in the proper extent; that is, from the beginning of the logical record.

Undefined Records

When undefined records are handled, PUT treats them as unblocked. The programmer must provide any blocking he wants. He must also determine the length of each record (in bytes) and load it in a register before he issues the PUT instruction for that record. The register used for this purpose must be specified in the DTF entry RECSIZE.

Updating

A sequential file on 2311, 2314, 2319, or 2321 DASD, a card input file in a 1442 or 2520, or a card file in the punch feed of a 2540 equipped with the Punch-Feed-Read special feature can be updated. That is, each DASD or card record can be read, processed, and transferred back to the same disk location, or card, from which it was read. In the case of a card file, the file must be specified as a combined file (CMBND) in the entry TYPEFLE.

When updating a file, one I/O area can be specified (entry IOAREA1) for both the input and output of a card record. If a second I/O area is required, it can be specified with IOAREA2.

A PUT for a card or DASD record must always be followed by a GET before another PUT is issued. GETs can be issued as many times in succession as desired. When updating a disk file, the record is not actually transferred with the PUT but with the next GET for the file.

For a file in a 2540 with the Punch-Feed-Read special feature, a PUT instruction must be issued for each card. A PUT instruction may be omitted, however, if a particular card does not require punching by a 1442 or 2520. The operator must run out the 2540 punch following a punch-feed-read job.

In the following example, data is punched in the same card that was read. Information from each card is read, processed, and then punched into the same card to produce an updated record.

```
|Name |Operation|Operand          | Col. 72 |
|-----|---------|-----------------|---------|
|FILEC|DTFCD    |                 |    X    |
|     |         |TYPEFLE=CMBND,   |    X    |
|     |         |IOAREA1=AREA,    |    X    |
|     |         |DEVADDR=SYS005,  |    X    |
|     |         |RECFORM=FIXUNB,  |    X    |
|     |         |IOAREA2=AREA2    |         |
|     |.        |                 |         |
|     |.        |                 |         |
|     |.        |                 |         |
|     |GET      |FILEC            |         |
|     |.        |                 |         |
|     |.        |                 |         |
|     |.        |                 |         |
|     |PUT      |FILEC            |         |
|     |.        |                 |         |
|     |.        |                 |         |
```

Punch and Printer Control

Stacker selection in a card read punch, and line spacing or skipping in a printer, can be controlled either by specified control characters in the data records or by the CNTRL macro instruction. Either method, but not both, may be used for a particular logical file.

When control characters in data records are used, the DTF entry CTLCHR must be specified, and every record must contain a control character in the main-storage output area. This control character must be the first character of each fixed-length or undefined record, or the first character following the record-length field in a variable-length record. The BLKSIZE specification for the output area must include the byte for the control character. If undefined records are specified, the RECSIZE specification must also include this byte.

When a PUT instruction is executed, the control character in the data record determines the command code (byte) of the channel command word (CCW) that IOCS establishes.

If CTLCHR=ASA: the control character is translated into the command code.

If CTLCHR=YES: the control character is used directly as the command code.

If the problem program using ASA control characters sends a space and/or skip command (without printing) to the printer, the output area must contain the first-character forms control and the remainder of the area must be blanks (X'40').

The particular character included in the record is determined by the function to be performed. For example, if double spacing is to occur after a particular line is printed, the code for double spacing must be the control character in the output line to be printed. The first character after the control character in the output data becomes the first character punched or printed. Appendix B gives a complete listing of control characters.

RELSE Macro

Name	Operation	Operand
[name]	RELSE	{filename / (1)}

The RELSE (release) macro instruction is used with blocked input records read from a DASD device, or with blocked spanned records read from, or updated on, a DASD device. This macro is also used with blocked input records read from magnetic tape. It allows the programmer to skip the remaining records in a block and continue processing with the first record of the next block when the next GET instruction is issued. When used with blocked spanned records, RELSE makes the next GET skip to the first segment of the next record.

If RELSE immediately precedes FSL or BSL (tape spacing for spanned records), then logical record spacing is ignored.

The symbolic name of the file, specified in the DTF header entry, is the only parameter required for this instruction. It can be specified as a symbol or in register notation.

The release instruction discontinues the deblocking of the present block of records, which may be either fixed or variable length. RELSE causes the next GET instruction to transfer a new block to the input area, or switch I/O areas, and make the first record of the next block available for processing. GET initializes the register or moves the first record to a workarea.

For example, this function can apply to a job in which records on DASD or tape are categorized. Each category (perhaps a major grouping) is planned to start as the first record in a block. For selective reports, specified categories can be located readily by checking only the first record in each block.

TRUNC Macro

Name	Operation	Operand
[name]	TRUNC	{filename / (1)}

The TRUNC (truncate) macro instruction is used with blocked output records written on DASD or magnetic tape. It allows the programmer to write a short block of records. Blocks do not include padding. Thus, the TRUNC macro can be used for a function similar to the RELSE instruction for input records, but in reverse. That is, when the end of a category of records is reached, that block can be written and the new category can be started at the beginning of a new block.

The symbolic name of the file, specified in the DTF header entry, is the only parameter required in this macro. If the TRUNC macro is issued for fixed-length blocked DASD records, the DTF entry TRUNCS must be included in the file definition.

When TRUNC is issued, the short block is written (on DASD or tape) and the output area is made available to build the next block. The last record written in the short block is the record that was built and included in the output block by the last PUT preceding the execution of the TRUNC macro. Therefore, if records are built in a workarea and the problem program determines that a record belongs in a new block, the TRUNC instruction should be issued first to write out the block. This should be followed by the PUT instruction for this particular record to move the record into the new block. If records are built in the output area, however, the

programmer must determine if a record belongs in the block before he builds the record.

Whenever variable-length blocked records are built directly in the output area, TRUNC macro must be used to write a complete block of records. When the PUT macro is issued after each variable length record is built, the output routines supply the programmer with the space (number of bytes) remaining in the output area. From this, the programmer determines if his next variable length record fits in the block. If it does not fit, he issues the TRUNC macro to write out the block and make the entire output area available to build the record. The amount of remaining space is supplied in the register specified in the entry VARBLD (see PUT Macro and DTFMT VARBLD).

CNTRL Macro

Name	Operation	Operand
[name]	CNTRL	{filename / (1)},code[,n1][,n2]

The CNTRL (control) macro instruction provides commands for these input/output units: magnetic tape units, card read punches, punches, printers, DASD, and optical readers. Commands apply to physical nondata operations of a unit and are peculiar to the unit involved. They specify such functions as rewinding tape, card stacker selection, line spacing on a printer, etc. For optical readers, commands specify marking error lines, correcting a line for journal tapes, document stacker selecting, or ejecting and incrementing documents. The CNTRL macro does not wait for completion of the command before returning control to the user, except for certain mnemonics for optical readers.

CNTRL is used with a logical file in a unit, and it usually requires two or three parameters. The first parameter must be the name of the file specified in the DTF header entry. It can be specified as a symbol or in register notation.

The second parameter is the mnemonic code for the command to be performed. This must be one of a set of predetermined codes (Figure 25).

The third parameter, n1, is required whenever a number is needed for stacker selection or immediate printer carriage control. The parameter, n2, applies to delayed spacing or skipping. In the case of a printer file, the parameters n1 and n2 may be required.

The CNTRL macro instruction must not be used for printer or punch files if the data records contain control characters and the entry CTLCHR is included in the file definition.

Whenever CNTRL is issued in the problem program, the DTF entry CONTROL must be included (except for DTFMT) and CTLCHR must be omitted. If control characters are used when CONTROL is specified, the control characters are ignored and treated as data.

IBM Unit	Mnemonic Code	n_1	n_2	Command
3420, 2400 Series Magnetic Tape Units	REW			Rewind Tape
	RUN			Rewind and Unload Tape
	ERG			Erase Gap (Writes Blank Tape)
	WTM			Write Tapemark
	BSR			Backspace to Interrecord Gap
	BSF			Backspace to Tapemark
	BSL			Backspace Logical Record
	FSR			Forward Space to Interrecord Gap
	FSF			Forward Space to Tapemark
	FSL			Forward Space Logical Record
2540 Card Read Punch	PS	1 2 3		Select Pocket 1, 2, or 3
2520, 1442 Card Read Punch	SS		1 2	Select Stacker 1 or 2
	E			Eject to Stacker 1 (1442 only)
		See Note		
1403, 1404, 1443, 1445, 3211 Printers	SP	c	d	Carriage Space 1, 2, or 3 lines
	SK	c	d	Skip to Channel c and/or d
1403 Printer with Universal Character Set Feature or 3211 Printer	UCS	ON		Data Checks are Processed with an Operator Indication
		OFF		Data Checks are Ignored and Blanks are Printed
3211 Printer	FOLD			Print Upper Case Characters for any Byte with Equivalent Bits 2-7
	UNFOLD			Print Character Equivalents of any EBCDIC Byte
2321 Data Cell Drive	SEEK			Seek to Address
	RESTR			Return Strip to Subcell
2311 Disk Storage Drive	SEEK			Seek to Address
2314, 2319 Direct Access Storage Facility				
1285 Optical Reader	MARK			Mark Error Line
	READKB			Read 1285 Keyboard
1287 Optical Reader	MARK			Mark Error Line in Tape Mode
	READKB			Read 1287 Keyboard in Tape Mode
	EJD			Eject Document
	SSD	1 2 3 4		Select Stacker A, B, Reject, or Alternate Stacking Mode
	ESD	1-4		Eject Document and Select Stacker
	INC			Increment Document at Read Station
1288 Optical Page Reader	ESD	1 3		Select Stacker A Reject Stacker (R)
	INC			Increment Document at Read Station

Note: c = An Integer that Indicates Immediate Printer Control (before printing).
d = An Integer that Indicates a Delayed Printer Control.

Figure 25. CNTRL Macro Instruction Command Codes

Magnetic Tape Unit Codes

The CNTRL macro instruction controls magnetic tape functions that are not concerned with reading or writing data on the tape. These functions are grouped in the following categories:

Rewinding tape to the load point
- REW - Rewind
- RUN - Rewind and unload

Moving tape to a specified position
- BSR - Backspace to interrecord gap
- BSF - Backspace to tapemark
- FSR - Forward space to interrecord gap
- FSF - Forward space to tapemark

Forward or backward logical record spacing
- FSL - Forward space logical record
- BSL - Backward space logical record

Writing a tapemark
- WTM - Write tapemark

Erasing a portion of the tape
- ERG - Erase gap (writes blank tape)

The tape rewind (REW and RUN) and tape movement (BSR, BSF, FSR, and FSF) functions can be used before a tape file is opened.

Note: If you are using a self-relocating program, you must open the file before issuing any commands to the file.

This allows the tape to be positioned at the desired location for opening a file, so that:

- The file is located in the middle of a multifile reel.
- The entry REWIND specifies NORWD, but for some conditions rewinding is required for the file.

The tape movement functions (BSR, BSF, FSR, and FSF) apply to input files only, and the following factors should be considered:

1. The FSR (or BSR) function permits the user to skip over a physical tape record (from one interrecord gap to the next). The record passes without being read into main storage. The FSF (or BSF) function permits the user to skip to the end of the logical file (identified by a tapemark).
2. The functions of FSR, FSF, BSR, and BSF always start at an interrecord gap.
3. If blocked input records are processed and if the user does not want to process the remaining logical records in the block, as well as one or more succeeding blocks (physical records), he must issue a RELSE macro before the CNTRL macro. Then, the next GET makes the first record of the new block available for processing. If the CNTRL macro (with FSR for example) is issued without a preceding RELSE, the tape is advanced. The next GET makes the next record in the old block available for processing.
4. For any I/O area combination except one I/O area and no workarea, IOCS is always reading one physical tape record ahead of the one that is being processed. Thus, the next physical record (block) after the current one is in main storage ready for processing. Therefore, if a CNTRL FSR function is performed, the second physical tape record beyond the present one is passed without being read into main storage.
5. If FSR or BSR is used, LIOCS does not update the block count. Furthermore, IOCS cannot sense tapemarks on an FSR or BSR command. Therefore, IOCS does not perform the usual EOV or EOF functions in these cases.

The tape spacing functions (FSL or BSL) apply to spanned record input files only. These codes are used when logical record spacing is desired. Consider these factors when FSL or BSL is specified:

1. Logical record spacing is ignored if it immediately follows a RELSE macro instruction.
2. Forward and backward spacing refer to the absolute direction of the spacing. For example, if BSL is specified on an input file with READ=BACK, only one logical record is skipped.
3. If an end-of-file, end-of-volume, or an error condition occurs while a FSL or BSL spacing function is being executed, the condition is handled as if it occurred during a normal GET operation.

Printer Codes

The CNTRL macro instruction can be used for any printer forms control. The codes for printer operation cause spacing (SP) over a

specified number of lines, or skipping (SK) to a specified location on the form. The third parameter is required for immediate spacing and skipping (before printing). The fourth parameter is required for delayed spacing or skipping (after printing).

The SP and SK operations can be used in any sequence. However, two or more consecutive immediate skips (SK) to the same carriage channel on the same printer result in a single skip immediate. Likewise, two or more consecutive delayed spaces (SP) and/or skips (SK) to the same printer result in the last space or skip only. Any other combination of consecutive controls (SP and SK), such as immediate space followed by a delayed skip or immediate space followed by another immediate space, causes both specified operations to occur.

1403 Printer With Universal Character Set or the 3211 Printer Codes

The CNTRL macro can be used before a PUT for the file to change the method of processing data checks. They can be either:

1. Processed with an indication given to operator, or

2. Ignored with blanks printed in place of the unprintable characters.

A data check occurs on a 1403 with the UCS feature or a 3211 when a character (except null, 00000000, or blank, 01000000) sent to the printer does not match any of the characters in the UCS buffer.

Before opening a file, the UCS job control command (BLOCK parameter) determines data check processing. That is, if BLOCK (1403) or NOCHK (3211) is specified, the data check is ignored and a blank is printed. Otherwise, an indication is given to the system operator. After opening a file, data checks are controlled according to the DTFPR UCS parameter until a CNTRL macro is used to change the method of processing data checks.

If the UCS form of the CNTRL macro is used for a printer without the UCS feature, the CNTRL macro is ignored.

The CNTRL macro can also be used before a PUT for the file to control the printing of lower case letters on the 3211 Printer. Lower case letters can either be printed or replaced by upper case equivalents.

Prior to using a CNTRL macro, the printing of lower case letters is controlled by the UCSB FOLD parameter of SYSBUFLD. If the FOLD parameter is specified, bits 0 and 1 are considered ones and the upper case equivalent of bits 2-7 are printed. If UNFOLD is specified, the character equivalent of the EBCDIC byte is printed.

2540 Card Read Punch Codes

Cards read or punched on the 2540 normally fall into the pocket specified in the DTF entry SSELECT (or the R1 or P1 pocket if SSELECT is omitted). The CNTRL macro with code PS is used to select a card into a different stacker, which is specified by the third operand (n1) of the CNTRL macro The possible selections are:

Feed	Pocket	Value of n1
Read	R1	1
Read	R2	2
Read	RP3	3
Punch	P1	1
Punch	P2	2
Punch	RP3	3

Input File: CNTRL can be used only when one I/O area, with or without a workarea, is specified for the file. To stack a particular card, the CNTRL instruction should be issued after the GET for that card. Before the next GET instruction is executed, the card is stacked into the specified pocket

> Note: If a user program indicates that operator intervention is required on a 2540 (for example, to correct a card out of sequence in a card deck), the user program has specified CONTROL=YES in CDMOD, and the user program is not using the CNTRL macro, then the user program should issue a CNTRL macro before the operator intervention is requested. The issuing of the CNTRL macro in this situation assures that subsequent commands issued to the 2540 after the operator intervention are not rejected as invalid.

Output File: CNTRL can be used with any permissible combination of I/O and workareas. When the user wants to select a particular card, CNTRL must be issued before the PUT for that card. However, CNTRL does not have to precede every PUT.

A document may be directed to stacker A, B, or R (reject stacker) by specifying a selection number of 1, 2, or 3 respectively. Also, documents may be selected into stackers A and B in an alternate stacking mode, with automatic stacker switching when one stacker becomes full. The selection number for alternate mode is 4. If selection number 4 is used in the first stacker selection macro, stacker A is filled first. If selection number 4 is used after other selection numbers, the last preceding selection number determines the first stacker to be filled. Selection numbers 1 and 3 only are available for the 1288.

1442 and 2520 Card Read Punch Codes

Cards fed in the IBM 1442 and 2520 are normally directed to the stacker specified in the DTF entry SSELECT. If SSELECT is omitted, they go to stacker 1. The CNTRL macro can be used to override the normally selected pocket temporarily.

Input File: CNTRL can be used only when one I/O area, with or without a workarea, is specified for the file. To stack a particular card, the CNTRL instruction should be issued after the GET for that card, and before the GET instruction for the following card. When the next card is read, the previous card is stacked in the specified stacker.

> Note: If CNTRL is not issued after each GET, the same card remains at the read station.

Output File: CNTRL can be used with any permissible combination of I/O areas and workareas. To stack a particular card, the CNTRL instruction should be issued before the PUT for that card. After the card is punched, it is stacked into the specified pocket immediately.

Combined File: CNTRL can be used with any permissible combination of I/O areas and workareas. If a particular card is to be selected, the CNTRL instruction for the file should be issued after the GET and before the PUT for the card. When the next card is read, the previous card is stacked into the specified stacker.

DASD (2311, 2314, 2319, and 2321) Codes

The CNTRL macro instruction to seek (SEEK) on a DASD or restore (RESTR) on a 2321 permits access movement to begin for the next READ, WRITE, GET, or PUT instruction for a file. While the arm is moving for a SEEK or the strip is being restored on a data cell, the programmer can process data and/or request I/O operations on other devices.

For sequential files, IOCS seeks the track that contains the next block (or physical record) for that file and the user does not supply a track address. If the CNTRL macro is not used, IOCS performs the seek or restore operations when a READ, WRITE, GET, or PUT instruction is issued.

1285, 1287, and 1288 Optical Reader Codes

The CNTRL macro instruction for the 1285, 1287, and 1288 is used for the nondata functions of marking a journal tape line, incrementing a document, and ejecting and/or stacker selecting a document. It is also used to read data from the 1285 or 1287 keyboard when processing journal tapes.

When the CNTRL macro is used with the READKB mnemonic, it allows a complete line to be read from the 1285 or 1287 keyboard when processing journal tapes. This permits the operator to key in a complete line on the keyboard if a 1285 or 1287 read error makes this type of correction necessary. When IOCS exits to the user's COREXIT routine, the problem program may issue the CNTRL macro instruction to read from the keyboard. The 1285 or 1287 display tube then displays the full line and the operator keys in the correct line from the keyboard, if possible. The line read from the keyboard is always read left-justified into the correct input area. The macro resets this area to binary zeros before the line is read. After CNTRL READKB is used, the contents of filename+80 are meaningful only for a wrong-length error indication (X'04'). Therefore, the user must determine whether the operator was able to recognize the unreadable line of data. The CNTRL macro with the READKB mnemonic waits for completion of the order before returning control to the user.

When processing journal tapes the CNTRL macro instruction with the MARK mnemonic marks (under program-control) a line on the input tape that results in a data transfer error or is otherwise suspect of error. To ensure that the proper line is marked, the CNTRL macro instruction must be issued in the user's error correction routine (specified in DTFOR COREXIT). If CNTRL is issued at any other time, the line following the one in error is marked.

When processing is done in document mode on the 1287, each document may be ejected with a CNTRL macro instruction. The EJD mnemonic causes the document to eject and the next document is fed. Documents may also be stacker selected by using the CNTRL macro instruction with the SSD mnemonic.

The CNTRL macro instruction with the ESD mnemonic combines the ejection and stacker selection functions. To satisfy the alternate ejection and stacker selection functions, the combined mnemonic must not be immediately preceded by an eject or immediately followed by a stacker select.

If a CNTRL macro is issued in COREXIT routine and a late stacker select or nonrecovery error condition occurs, IOCS branches to the next sequential instruction. Filename+80 should therefore be tested for these conditions after issuing a CNTRL macro.

The CNTRL macro with the INC mnemonic may be used for document incrementation. This macro is not used with documents having a scannable area shorter than 6 inches. When this mnemonic is issued, the document is incremented forward 3 inches. This macro may be used only once per document.

For the 1288, the CNTRL macro with the INC mnemonic can increment only documents with a scannable area longer than 6.5 inches. The document is incremented to the next stopping point as selected by console switches on the 1288. More than one CNTRL macro can be used per document.

Document ejection and/or stacker selection and document increment functions can also be accomplished by including the appropriate CCW(s) within the channel command word list addressed by the READ macro, rather than by using the CNTRL macro. This technique results in increased document throughput.

> Note: For processing documents in a multiprogramming environment where the partition containing 1287 support does not have highest priority, the eject and stacker select functions must be accomplished by a single command. However, when processing documents in a dedicated environment, the stacker select command can be executed. It must follow the eject command within 270 milliseconds if the document was incremented, or within 295 milliseconds if the document was not incremented. The eject and stacker select functions must occur alternately. If the timing requirements are not met, a late stacker selection condition occurs.

CHNG Macro for BPS and BOS Systems Only

Name	Operation	Operand
[name]	CHNG	SYSnnn

This macro instruction is provided only for Basic Programming Support and Basic Operating System upward compatibility. No code is generated from this macro instruction. In the Disk Operating System, tape channel switching is handled automatically by physical IOCS.

ERET Macro

Name	Operation	Operand
[name]	ERET	{ SKIP / IGNORE / RETRY }

This macro instruction enables a problem program ERROPT or WLRERR routine to return to IOCS and specify an action to be taken. The macro applies only to DTFMT-MTMOD, DTFIS-ISMOD, and DTFSD-SDMOD files with the ERREXT operand specified.

The SKIP operand passes control back to the logic module to skip the block of records in error and process the next block. The IGNORE operand passes control back to the module to ignore the error and continue processing with the block in error. The RETRY operand causes the module to retry the operation that resulted in the error.

> Note 1: For sequential disk output, an ERET SKIP is treated as an ERET IGNORE.
>
> Note 2: With MTMOD for any error or with SDMOD wrong-length record errors, ERET RETRY cancels the job with an invalid SVC message.

PRTOV Macro

Name	Operation	Operand
[name]	PRTOV	{ filename / (1) }, { 9 / 12 } [, { routine-name / (0) }]

The PRTOV (printer overflow) macro instruction is used with a logical printer file to specify the operation to be performed when a carriage overflow condition occurs. Whenever this macro instruction is issued in a problem program, the DTFPR or DTFSR entry PRINTOV=YES must be included in the file definition.

PRTOV requires two or three parameters. The first parameter must be the filename, either as a symbol or in register notation.

The second parameter must specify the number of the carriage control channel (9 or 12) used to indicate the overflow. When an overflow condition occurs, IOCS restores the printer carriage to the first printing line on the form (channel 1), and normal printing continues.

A third parameter is required if the programmer prefers to branch to his own routine on an overflow condition, rather than skipping directly to channel 1. It specifies the symbolic name of a user's routine. The name can be specified either as a symbol or in register notation. However, the name should never be preloaded into register 1.

In this case, IOCS does not restore the carriage to channel 1. In his routine, the user may issue any IOCS macro instruction to perform whatever functions he desires. The CNTRL macro cannot be issued to the file unless CONTROL=YES is specified in the DTF. For example, he can print total lines, skip to channel 1, and print overflow page headings. At the end of his routine, the user can return to IOCS by branching to the address in register 14. IOCS supplies this address upon entry to the user's routine. Therefore, if other IOCS macros are used in the routine (for example, the CNTRL macro), the user must save and restore register 14 himself.

The PRTOV macro causes a skip to channel 1, or branches to the user's routine if an overflow condition (channel 9 or 12) is detected on the preceding space or print command. An overflow condition is not recognized during a carriage skip operation. After the execution of any command that causes carriage movement (PUT or immediate CNTRL), the user should issue a PRTOV macro before issuing the next CNTRL or PUT. This ensures that the user's overflow option is executed at the correct time.

MAGNETIC READER MACROS

Within a particular program, the user should utilize either the GET or the READ, CHECK, WAITF combination. For a program operating with two or more MICR devices, the READ, CHECK, WAITF combination allows processing to continue within the program when any document buffer is ready for processing. On the other hand, the GET macro instruction (suggested for a program operating with one MICR device) includes an inherent wait for a document buffer to become available within the file before processing begins. In a multiprogramming environment, control always passes to another partition whenever a WAIT condition occurs.

Before any MICR document processing can be performed, the file(s) must be opened. If an unrecoverable I/O error occurs when using the GET macro logic, no more GETs can be issued for the file. If an unrecoverable I/O error occurs when using the READ, CHECK, WAITF logic or when document processing for that file is complete, the user can effectively continue by closing the file. Further READ, CHECK, WAITFs treat this file as having no documents ready for processing (see byte 0, bits 5 and 6 of Appendix E).

READ Macro

The READ macro instruction makes the next sequential buffer available to the user, but it does not verify that it is ready for processing. The CHECK macro is provided to make that test. If the buffer is not ready for processing, the next READ to that file points to the same buffer.

Name	Operation	Operand
[name]	READ	{filename / (1)} ,MR

****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

The first operand specifies the name of the file associated with the record to be read. This name would be the same as that specified in the DTFMR header entry for the file. It can be given as a symbol or in register notation. The second operand signifies that the file is for a magnetic character reader (MICR).

CHECK Macro

Name	Operation	Operand
[name]	CHECK	{filename (1)} [,{control address (0)}]

****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

A READ macro instruction must already have been issued to the file before a CHECK macro instruction is issued.

The CHECK macro instruction examines the buffer status indicators to determine whether the buffer contains data ready for processing, is waiting for data, contains a special nondata status, or the file (filename) is closed. If the buffer has data ready for processing, control passes to the next sequential instruction. If the buffer is waiting for data, or the file is closed, control passes to the control address, if present. If the buffer contains a special nondata status, control passes to the ERROPT routine for the user to examine the posted error conditions before determining his action. (See Appendix E, byte 0, bits 2, 3, and 4.) The user may return from the ERROPT routine to the next sequential instruction via a branch on register 14, or to the control address in register 0.

If the buffer is waiting for data, or the file is closed, and the control address is not present, control is given to the user at his ERROPT address specified in the DTFMR macro. If an error, a closed file, or a waiting condition occurs (with no control address) and no ERROPT address is present, control is given to the user at the next sequential instruction. If the waiting condition occurred, byte 0, bit 5 of the buffer is set to 1. If the file was closed, byte 0, bits 5 and 6 of the buffer are set to 1 (see Appendix E).

The first operand specifies the name of the file associated with the record to be checked. This name is the same as that specified for the DTFMR header entry for the file. It can be given either as a symbol or in register notation.

The second operand indicates the address to which control passes when a buffer is waiting for data or the file is closed. This parameter can be given either as a symbol or in register notation.

WAITF Macro

Name	Oper-ation	Operand
[name]	WAITF	{filename1 (r1)} [,{filename2 (r2)}...,{filenamen (rn)}]

****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

The WAITF macro instruction is essential to the multiprogramming feature of the system. It allows processing of programs in other partitions while waiting for document data. If any device within the WAITF macro list has records or error conditions ready to be processed, control remains in the partition and processing continues with the instruction following the WAITF macro instruction.

One WAITF macro instruction must be issued after a set of READ-CHECK combinations before the problem program attempts to return to a previously issued combination. Thus, the (multiple) WAITF macro instruction must be issued between successive executions of a particular READ macro instruction.

The operands required are for the files waiting to be processed. The names given

are those found in the DTFMR header entries.

DISEN Macro

This macro instruction stops the feeding of documents through the magnetic character reader or optical reader/sorter. The program proceeds to the next sequential instruction without waiting for the disengagement to complete. The user should continue to issue GETs or READs until the unit exception bit (byte 0, bit 3, Appendix E) of the buffer status indicators is set ON.

Name	Operation	Operand
[name]	DISEN	{filename / (1)}

The only required operand specifies the name of the file to be disengaged. This name is the same as that specified for the DTFMR header entry for the file. The operand can be given either as a symbol or in register notation.

LITE Macro

This macro instruction lights any combination of pocket lights on an IBM 1419 Magnetic Character Reader or 1275 Optical Reader/Sorter. Before using the LITE macro, the DISEN macro instruction must be issued to disengage the device. Processing of the documents should be continued until the unit exception bit (Appendix E byte 0, bit 3) of the buffer status indicators is set ON. When this bit is ON, the follow-up documents have been processed, the MICR reader has been disengaged, and the pocket LITE macro can be issued.

Name	Operation	Operand
[name]	LITE	filename (1) [,{light switches / (0)}]

The first operand indicates the header entry for the DTFMR table that contains the address for the particular device. The second operand indicates a 2-byte area containing the pocket light switches. Both operands can be given either as a symbol or in register notation. If the second operand is given as a symbol it must be on a halfword boundary.

The bit configuration for the pocket light switch area is shown in Figure 26. The pocket lights that are turned ON should have their indicator bits set to 1. If an error occurs during the execution of the pocket lighting I/O commands, bit F is set to 1. This error condition normally indicates that the pocket light operation was unsuccessful.

Bits	0	1	2	3	4	5	6	7	8	9	A	B	C D E	F
Pocket Lights	A	B	0	1	2	3	4	5	6	7	8	9	Reserved with binary zeros	Error indicator bit

Figure 26. Bit Configuration for Pocket Light Switch Area of IBM 1419

OPTICAL READER MACROS

GET Macro

(See Sequential Processing Macro: GET Macro.

CNTRL Macro

(See Sequential Processing Macro: CNTRL Macro.)

DSPLY Macro

Name	Operation	Operand
[name]	DSPLY	{filename / (1)}, (r), (r)

The DSPLY macro displays the document field on the 1287 display scope. A complete field may be keyboard-entered if a 1287 read error makes this type of correction necessary. An unreadable character may be replaced by the reject character either by the operator (if processing in the on-line correction mode) or by the device (if processing in the off-line correction mode). The user may then use the DSPLY macro to display the field in error. The 1287 display tube displays the full field and the operator keys in the correct field from the keyboard, if possible. The field read from the keyboard is always read into the area (normally within IOAREA1) that was originally intended for this field as specified in the CCW. The macro first resets this area to binary zeros. At completion of the operation, the data is left-justified in the area.

This instruction always requires three parameters. The first parameter is the symbolic name specified in the DTFOR header entry for the 1287 file. The second parameter specifies a general purpose register (2-12) into which the problem program places the address of the load format CCW giving the document coordinates for the field to be displayed. When the DSPLY macro is used in the COREXIT routine, the address of the load format CCW can be obtained by subtracting 8 from the 3-byte address that is right-justified in the fullword location beginning at filename+32. (The high-order fourth byte of this full word should be ignored.) If the DSPLY macro is not used in the COREXIT routine, the user must determine the load format CCW address. The third parameter specifies a general purpose register (2-12) into which the problem program places the address of the load format CCW giving the coordinates of the reference mark associated with the displayed field.

The contents of filename+80 are meaningful only for X'40' (1287 scanner cannot locate the reference mark) and X'04' (wrong-length record) after the DSPLY macro is issued. Therefore, the user must determine whether the operator was able to recognize the unreadable line of data.

> Note: When using the DSPLY macro, the user must ensure that the load format CCW is command chained to the CCW used to read that field. This provides the document coordinates for the field to be displayed.

READ Macro

The READ macro instruction is used in sequential processing to cause the next sequential IBM 1287 or 1288 Optical Reader (document mode only) record to be read.

Name	Operation	Operand
[name]	READ	{filename / (1)}, OR, {name / (r)}

> **********
>
> Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.
>
> **********

The first parameter specifies the name in the DTFOR header for this file, and it is always required. The parameter OR is required to indicate an optical character reader. The parameter [name] is always required. This parameter specifies the address of the user provided channel command word list to be used to read a document from the 1287 or 1288 file. The register entry may be used in this parameter to provide the address of the

channel command word list. The first channel command word in the list cannot be a transfer-in-channel CCW.

To accomplish document ejection and/or stacker selection and document increment functions, include the appropriate CCW(s) within the channel command list addressed by the read macro. This technique results in increased processing throughput. This method is preferable to using the CNTRL macro.

> Note: The WAITF macro must be issued after the READ macro and before the program attempts to process an input record for the file.

RESCN Macro

Name	Oper-ation	Operand
[name]	RESCN	{filename / (1)} ,(r1),(r2)[,n1][,n2]

> Note: For the 1287 Models 3 and 4 and the 1288, this macro can only be used with READ BACKWARD commands. If used with READ FORWARD commands, the input area is not cleared.

The RESCN macro selectively rereads a field on a document when a defective character(s) makes this type of operation necessary. The field is always right-justified into the area (normally within IOAREA1) that was originally intended for this field as specified in the CCW. The macro first resets this area to binary zeros.

The first parameter specifies the symbolic name of the 1287D file specified in the DTFOR header entry for the file. The second parameter specifies a general purpose register (2-12) into which the problem program places the address of the load format CCW.

When this macro is used in the COREXIT routine, the address of the load format CCW is obtained by subtracting 8 from the 3-byte address that is right-justified in the fullword location beginning in filename+32. (The high-order fourth byte of this fullword should be ignored.) If the RESCN macro is not used in the COREXIT routine, the user must determine the load format CCW address. The third parameter specifies a general purpose register (2-12) into which the problem program places the address of the load format CCW for reading the reference mark. The previous three parameters are always required, and result in one attempted reread for the field.

The fourth parameter, n1, allows the user to specify the number of attempts (one to nine allowed) to reread the unreadable field. If this parameter is omitted, one is assumed. The fifth parameter, n2, indicates one more reread. It forces on-line correction of any unreadable character(s) by individually projecting the unreadable character(s) on the 1287 display scope. The operator must key in a correction (or reject) character(s). This operand cannot be used for 1288 processing.

If the reread of the field results in a wrong-length record, incomplete read, or an unreadable character, it is indicated in filename+80.

> Note: When using the RESCN macro, the user must ensure that the load format CCW (giving the document coordinates for the field to be read, second parameter) is command chained to the CCW used to read that field. When 1288 unformatted fields are read, the RESCN macro should not be used.

RDLNE Macro

Name	Operation	Operand
[name]	RDLNE	{filename / (1)}

The RDLNE macro provides selective on-line correction when processing journal tapes on the IBM 1285 or the IBM 1287 Optical Reader. This macro reads a line in the on-line correction mode while processing in the off-line correction mode. If the reader cannot read a character, IOCS first resets the input area to binary zeros and then retries the line containing the unread character. If the read is unsuccessful, the user is informed of this condition via his error correction routine (specified in DTFOR COREXIT). The RDLNE macro may then be issued to cause another attempt to read the line. If the character in the line still cannct be read, the character is displayed on the 1285 or 1287 display scope. The operator keys in the correct character, if possible. If the operator cannct readily identify the defective character, he may enter the reject character in the error line. This condition is posted in filename+80 for user examination. Wrong-length records and

incomplete read conditions are alsc posted to filename+80. RDLNE should be used in the COREXIT routine only, or the line following the one in error is read in on-line correction mode.

This macro requires only one parameter, the symbolic name of the 1285 or 1287 file from which the record is to be retrieved. This name is the same as that specified in the DTFOR header entry for the file.

WAITF Macro

Name	Operation	Operand
[name]	WAITF	{filename / (1)}

The WAITF macro instruction is used in sequential processing to ensure that the transfer of an IBM 1287 or 1288 Optical Reader record (document mode only) is completed. It must have only one parameter: the symbolic name of the file containing the record.

This instruction must be issued following a READ and before the problem program attempts to process an input record for the file concerned. The program waits until the transfer of data is complete.

The WAITF macro instruction accomplishes all checking for read errors on the 1287 or 1288 file and exits to the user-provided COREXIT routine for user handling of these conditions, if necessary.

WORK FILE MACROS FOR TAPE AND DISK

A work file can be used for disk and tape input, output, or both. If TYPEFLE=WORK is specified in the DTF macro instruction, work file macro instructions READ, WRITE, and CHECK are provided. Alsc, if NOTEPNT=YES is specified, work file macro instructions NOTE, POINTR, POINTW, and POINTS are provided. Work files process only fixed-length unblocked records and undefined record formats. Work files are not permitted for tape files written in ASCII mode.

A tape work file is a single-volume file used for both input and output, even within a single program phase. It passes intermediate results between successive phases or job steps. However, work files also can be written, read, and rewritten within a single phase, without requiring additional OPEN or CLOSE processing. Work files are defined as an option of the DTFMT and MTMOD macro instructions and are accessed by the READ/WRITE and CHECK macro instructions.

The first time a work file volume is opened, it is opened as an output file. OPEN examines the tape to determine whether the tape contains standard labels. The DTFMT entry FILABL is ignored. If the tape is labeled and the date in the header label has expired, a new label is created, consisting of HDR1 followed by 76 blanks. The job control label information cards are not required. If the tape does not already contain standard labels, labels are not created for the work file. Trailer labels are not processed.

If a work file with standard labels is reopened, OPEN determines from the HDR label that the file is a work file and does not rewrite the labels.

When a tapemark is sensed during a read operation, or when an end-of-reel reflective spot is sensed during a write operation, IOCS exits to the address specified by the user by the entry EOFADDR.

Disk work files are supported as single volume single pack files. They are always opened as output files. Standard label information must be supplied by the user. Both normal extents (type 1) and split extents (type 8) are supported. File protection for work files is ensured only if the labels are unexpired.

DELETING A WORK FILE AFTER USE: The entry DELETFL=NO must not be used. OPEN creates a format 1 label for the file, and CLOSE destroys this label. The next job requiring a work file can use the same extents and filename.

SAVING A WORK FILE AFTER USE: The expiration date in the DLBL job control card must not be the current date. The entry DELETFL=NO must be specified in the DTF for the file. OPEN creates a format 1 label, but CLOSE does not delete it. Thus, the file can be saved until the expiration date is reached.

DELETING AN UNEXPIRED FILE: When the user tries to use the limits of an unexpired file, an operator message is printed to indicate the overlap condition. The operator can then delete the label, after which OPEN creates a label for the new file and the job continues.

READ Macro (for TAPE or DISK Workfiles)

The READ macro instruction reads the next sequential physical record, or part of it, from the file associated with the filename. The record is read into the area of main storage indicated by the third operand.

The DTF entry READ=FORWARD or BACK should specify the type of read for a tape file.

Name	Operation	Operand
[name]	READ	{filename / (1)}, SQ, {area / (0)} [, {length / (r) / S}]

****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

The first parameter specifies the name of the file associated with the record to be read and is always required. This name is the same as the name specified in the DTFMT or DTFSD header entry for the file. The name can be given as a symbol or in register notation.

The parameter SQ (for sequential) is always required. Area specifies the name (as a symbol or in register notation) of the input area used by the file. If tape is to be read backwards, area must be the address of the rightmost byte of the input area.

The length parameter is used only for records of undefined format (RECFORM=UNDEF). To read only a portion of a record, an actual length (or a register containing the number) can be specified. Or, an S can be provided to indicate that the entire physical record should be read.

If the work file records are fixed length unblocked records (RECFORM=FIXUNB), the length parameter, including S, must not be specified in the READ macro. The number of characters to be read is specified in the BLKSIZE entry. The user can change this number (which is stored in the DTF table) at any time by referencing the halfword filenameL.

Note: The CHECK macro must be issued after the READ macro and before the program attempts to process an input record for the file.

WRITE Macro

The WRITE macro instruction writes a record from the indicated area into the file associated with the filename. The record is stored sequentially following the last record written in this file.

Name	Operation	Operand
[name]	WRITE	{filename / (1)}, {SQ / UPDATE}, {area / (0)} [, {length / (r)}]

****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

The first parameter specifies the name of the file associated with the record to be written and is always required. This name is the same as the name specified in the DTFMT or DTFSD header entry for this file. The name can be given as a symbol or in register notation.

The second parameter specifies the type of WRITE to be executed. For magnetic tape, this parameter is always SQ. If SQ is specified for disk work files, a formatting WRITE (write count key and data) is executed. If UPDATE is specified, a nonformatting WRITE (write data) is executed. An update WRITE should be preceded by a READ, WRITE UPDATE, POINTR, or POINTW macro instruction. A CLOSE macro (following an

update write) protects the updated file by not writing an end-of-file record. If SQ is specified and a CLOSE immediately follows an OPEN (no formatting WRITE commands were issued), an end-of-file record is not written.

The parameter area specifies the name, as a symbol or in register notation, of the output area used by the file.

The last parameter is used only for records of undefined format (RECFORM=UNDEF). Length specifies the actual number (or register containing the number) of bytes to be written.

If fixed-length unblocked records (RECFORM=FIXUNB) are written, length is not used in the WRITE macro. The number of characters to be written is specified in the BLKSIZE entry. The user can change this number, which is stored in the DTF table, at any time by referencing the halfword filenameL. For disk, the BLKSIZE entry should not include eight bytes for the length of a count field.

> Note: The CHECK macro must be issued after the WRITE macro to allow for completion of the input/output operation.

CHECK Macro

Name	Operation	Operand
[name]	CHECK	{filename / (1)}

This macro instruction must be used after each READ or WRITE. It prevents user processing until completion of the input/output operation, started by either a READ or a WRITE, for the device associated with the filename.

If the I/O operation is completed without any error or other exceptional condition, CHECK returns control to the next instruction. If the operation results in a read error, CHECK processes the user's option specified in ERROPT. If CHECK finds an end-of-file condition, control is passed to the routine specified in EOFADDR.

NOTE Macro

Name	Operation	Operand
[name]	NOTE	{filename / (1)}

The NOTE macro instruction obtains identification for a physical record that is read or written during processing.

> Note: The CHECK macro must be issued before the NOTE macro to ensure that the last operation has completed.

For magnetic tape, the last record read or written is identified by the number of physical records read or written in the specified file from the load point. The information is returned in register 1 in the form 0bbb, where

- 0 = Eight binary zeros.
- bbb = Physical record number in binary.

For disk, if a READ precedes the NOTE, the record identified is the last record read. If a WRITE precedes the NOTE, the record just written is the identified record. The identification is returned in register 1 in the form 0chr, where

- 0 = Eight binary zeros.
- c = Cylinder number,
- h = Track number,
- r = Record number within the track.

c, h, and r are binary numbers. If NOTE follows a WRITE to a disk file, the unused space remaining on the track following the end of the identified record returns in register 0 as the binary number 00nn.

For magnetic tape, the user must store the identification (in the 0bbb form) so that it can be used later in either a POINTR or POINTW instruction. For disk, the user must construct a six-byte field and store in it the identification of the record and the remaining track capacity (in the form 0chrnn) so that it can be used later in a POINTR or POINTW instruction to find the noted record again.

POINTR Macro

Name	Operation	Operand
[name]	POINTR	{filename / (1)}, {address / (0)}

The POINTR macro instruction repositions the file for reading a record identified by the NOTE macro.

For magnetic tape, address specifies a 4-byte main-storage location containing the required record identification. It can be expressed as a symbol or in register notation. The four-byte number must be in the form obtained from the NOTE macro (0bbb). POINTR repositions the file to read the record that was read or written immediately before the NOTE that was used to create the record identification field was issued. For magnetic tape, a WRITE must not follow POINTR.

For disk, address specifies a six-byte main-storage location containing the required record identification and the remaining track capacity. It can be expressed as a symbol or in register notation. The six-byte number must be supplied in the form obtained from the NOTE macro (0chrnn) where nn is the length remaining on the track. POINTR repositions the file to read the record identification (in the 0bbb form) returned when a previous NOTE macro was issued. If a WRITE UPDATE follows the POINTR macro, the noted record is overwritten. If a WRITE SQ follows the POINTR macro, the record after the noted record is written, and the remainder of the track is erased.

Some programs using disk work files may include multiple WRITE instructions following a NOTE macro. If a POINTR instruction is issued and the work file records are in undefined format, there may be occasions when a replacement record longer than the original record cannot be written in the space available on the track. In this case, the original record remains as the last record on the track when the next WRITE is performed. The replacement record is written as the first record on the next track of the file.

POINTW Macro

Name	Operation	Operand
[name]	POINTW	{filename / (1)}, {address / (0)}

The POINTW macro instruction repositions a file to write a record.

For magnetic tape, address specifies a four-byte main-storage location containing the required record identification. It can be expressed as a symbol or in register notation. The four-byte number must be in the form obtained from the NOTE macro (0bbb). POINTW repositions the file to write a record after the one previously identified by the NOTE. When a READ is issued to a tape file following a POINTW, the tape is positioned to read the record following the one identified by the NOTE.

For disk, address specifies a six-byte main-storage location containing the required record identification. The disk address can be expressed as a symbol or in register notation. The six-byte number must be supplied in the form obtained from the NOTE macro (0chrnn) where nn is the length remaining on the track. POINTW repositions the file to write at the record location that was read or written immediately before the last NOTE macro was issued. If a WRITE UPDATE is then issued, the noted record is overwritten. If a WRITE SQ is issued, the record following the noted record is written and the remainder of the track is erased. A READ macro can follow the POINTW macro. The record identified by the NOTE is the record read.

Some programs using disk work files may include multiple WRITE instructions following a NOTE macro. If a POINTW instruction is issued and the work file records are in undefined format, there may be occasions when a replacement record longer that the original record cannot be written in the space available on the track. In this case, when the next WRITE is performed, the original record remains as the last record on the track. The replacement record is written as the first record on the next track of the file.

POINTS Macro

The POINTS macro instruction repositions a file to the beginning of the file.

Name	Operation	Operand
[name]	POINTS	{filename / (1)}

For a tape file, the tape is rewound. If the file contains any header labels, they are bypassed, and the tape is positioned to the first record following the label set.

For disk, the file is repositioned to the lower limit of the first extent. An example of POINTS with workfile processing follows:

```
   L      12,LENGTH   (load length of var-
                      length record to reg)
A  WRITE  F,SQ,OUT,(12)  (write a record)
   .                  (processing of data
   .                  unrelated to OUT)
   .
   CHECK  F           (wait until record is
   .                  written)
   .
   .
   BNZ    A           (finish processing)
   POINTS F           (reposition to begin-
                      ning of file)
B  READ   F,SQ,IN,S   (read physical
                      record 1)
   .
   .                  (processing data un-
   .                  related to IN)
   CHECK  F           (wait until record is
                      read)
   BNZ    B           (finish processing)
   EOJ
```

On disk or magnetic tape, a POINTS followed by a WRITE (SQ) causes the new record to be written and the remainder of the track is erased. On disk, POINTS should not be followed by a WRITE (UPDATE).

Completion Macros

FEOV Macro

Name	Operation	Operand
[name]	FEOV	{filename / (1)}

The FEOV (forced end-of-volume) macro instruction is used for either input or output files on magnetic tape (programmer logical units only) to force an end-of-volume condition before sensing a tapemark or reflective marker. This indicates that processing of records on the current volume is finished, but that more records for the same logical file are to be read from, or written on, a following volume. If a spanned record is begun on an output file and there is not enough space to contain it, MTMOD issues an FEOV at the end of the last completed spanned record. The last spanned record (for which there was no room) is rewritten on a new volume. For system units, see the SEOV Macro.

The name of the file, specified in the header entry, is the only parameter required in the operand. The name can be specified either as a symbol or in register notation.

When logical IOCS macro instructions are used for a file, FEOV initiates the same functions that occur at a normal end-of-volume condition, except for checking of trailer labels.

For an input tape, FEOV immediately rewinds the tape (as specified by REWIND) and provides for a volume change (as specified by the ASSGN cards). Trailer labels are not checked. FEOV then checks the standard header label on the new volume and allows the user to check any user-standard header labels if LABADDR is specified. If nonstandard labels are specified (FILABL=NSTD), FEOV allows the user to check these labels as well.

For an output tape, FEOV writes

- A tapemark (two tapemarks for ASCII files.)
- A standard trailer label and user-standard labels (if any).
- A tapemark.

If the volume is changed, FEOV then writes the file header label(s) on the new volume (as specified in the entries REWIND, FILABL, LABADDR, and the ASSGN cards). If nonstandard labels are specified, FEOV allows the user to write trailer labels on the completed volume and header labels on the new volume, if desired.

FEOVD Macro

Name	Operation	Operand
[name]	FEOVD	{filename / (r)}

The FEOVD (forced end-of-volume for disk) macro instruction is used for either input or output files to force an end-of-volume condition before it actually occurs. This indicates that processing of records on one volume is finished, but that more records for the same logical file are to be read from, or written on, the following volume. If extents are not available on the new volume, or if the format 1 label is posted as the last volume of the file, control is passed to the EOF address specified in the DTF.

The name of the file is the only required operand. The name can be specified either symbolically or in register notation.

When FEOVD is issued to an input file, an end of extent is posted in the DTF. When the next GET is issued to this file, any remaining extents on the current volume are bypassed, and the first extent on the next volume is opened. Normal processing is then continued on the new volume.

When FEOVD is issued to an output file, a short last block is written, if necessary, with a standard end-of-file record containing a key length of one (indicating end of volume). An end-of-extent condition is posted in the DTF. When the next PUT is issued to the file, all remaining extents on the current volume are bypassed. The first extent on the next volume is then opened, and normal processing continues on the new volume. The DOS FEOVD EOV marker is compatible with the OS EOV marker.

If the FEOVD macro is followed immediately by the CLOSE macro, the end of volume marker is rewritten as an end of file marker, and the file is closed as usual.

CLOSE(R) Macro

Op	Operand
for self-relocating programs	
CLOSER	{filename1 / (r1)} [,{filename2 / (r2)} ... ,{filenamen / (rn)}]
for programs that are not self-relocating	
CLOSE	{filename1 / (r1)} [,{filename2 / (r2)} ... ,{filenamen / (rn)}]

Note: To write the most efficient code (in a multiprogramming environment), we recommend that the self relocating form of OPEN be used. (See also Appendix G.)

The CLOSE macro instruction must be issued to deactivate any file that was previously opened in any input/output unit in the system. Console files cannot be closed. A CLOSE normally deactivates an output file by writing an EOF record and output trailer labels, if any. CLOSE sets a bit in the format 1 label to indicate the last volume of the file. A file may be closed at any time by issuing this macro.

No further commands can be issued for the file unless it is reopened. Sequential DASD files cannot be successfully reopened for output unless the DTFSD table is saved before the file is first opened, and restored between closing the file and reopening it again as an output file.

When the operation CLOSE is used, the symbolic address constants that CLOSE generates from the parameter list are not self-relocating. When CLOSER is specified, the symbolic address constants are self-relocating. The self-relocating format of close is recommended.

The symbolic name of the logical file (assigned in the DTF header entry) to be closed is entered in the operand field. A maximum of 16 files may be closed by one instruction. Alternately, the user can load the address of the filename in a register and specify the register using ordinary register notation. The high-order 8 bits of this register must be zeros. The address of the filename may be preloaded into any register, 0-15.

Notes:

- If you use register notation, we recommend using only registers 2-12. This will make your programs more compatible with the Operating System (OS).
- If CLOSE is issued to a magnetic tape input file that has not been opened, the option specified in the DTF rewind option is performed. If CLOSE is issued to a magnetic tape output file that has not been opened, no tapemark or labels are written, and no rewind option is performed.

For a paper tape punch file with two I/O areas, CLOSE checks for the successful completion of the last operation.

Direct Access Method (DAM)

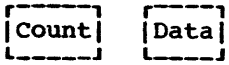
DASD records can be processed in random order by the Direct Access Method (DAM). In this method, the user specifies the address of the record to IOCS and issues a READ or WRITE macro instruction to transfer the specified record. Variations in the parameters of the READ or WRITE instructions permit records to be read, written, updated, or replaced in a file. Whenever this method of processing records is used, the logical file and main-storage area(s) allotted to the file must be defined by the declarative macro DTFDA (Define The File for Direct Access).

Record Types

DASD records that will be processed by DAM can exist on the DASD in either of two formats: with a key area, or without.

With key area:

Without key area:



When processing spanned records, this format applies only to the first segment. For additional information on spanned records, see Appendix F.

Whenever records in a file have keys that are to be processed:

- Every record must have a key, and
- All keys must be the same length.

Whenever the DTFDA entry KEYLEN is not specified for a file, IOCS ignores keys, and the DASD records may or may not contain key areas. A WRITE ID or READ ID reads or writes the data portion of the record. However, when KEYLEN is not specified in the DTF for a file, WRITE AFTER cannot be used to extend a file that has keys.

IOCS considers all records as unblocked (one logical record per one physical record). If the user wants blocked records, he must provide his own blocking and deblocking. Records are also considered to be either fixed, variable, or undefined length. A spanned record indicates variable blocks where the size of each segment is a function of the track size and record size. The record size is set by a formatting WRITE macro instruction. All the variable record segments of a given spanned record are logically contiguous. The type of records in the file must be specified in the DTFDA entry RECFORM. Whenever records specified as undefined are written to a file, the user must determine the length of each record and load it in a register (specified by the DTFDA entry RECSIZE) before he issues the WRITE instruction for that record.

DIRECT ACCESS IOAREA1

The DTFDA entry IOAREA1 defines an area of main storage in which records are read on input or built on output.

Format

The format of the I/O area is determined at assembly time by the following DTFDA entries: AFTER, KEYLEN, READID, WRITEID, READKEY, and WRITEKY. Figure 27 describes the types of DTF macros and the I/O areas that they define. The information in this figure should be used to determine the length of the I/O area specified in the BLKSIZE entry. The I/O area must be large enough to contain the largest record in the file. If the DTF used requires it, the I/O area must include room for an 8-byte count field. The count is provided by IOCS.

Contents

The phrase contents of the IOAREA1 refers to the information provided by or to IOCS for a specific imperative macro instruction. Figure 27 gives a summary of what the contents are for each type of READ/WRITE. When the user builds a record,

he must place the contents (Figure 27) in the appropriate field of the I/O area. The contents that IOCS provides on input are always placed in the appropriate field of the I/O area. For example, if the DTF used for the file resulted in the uppermost format shown in Figure 27, the data would be located to the right of the count and key area.

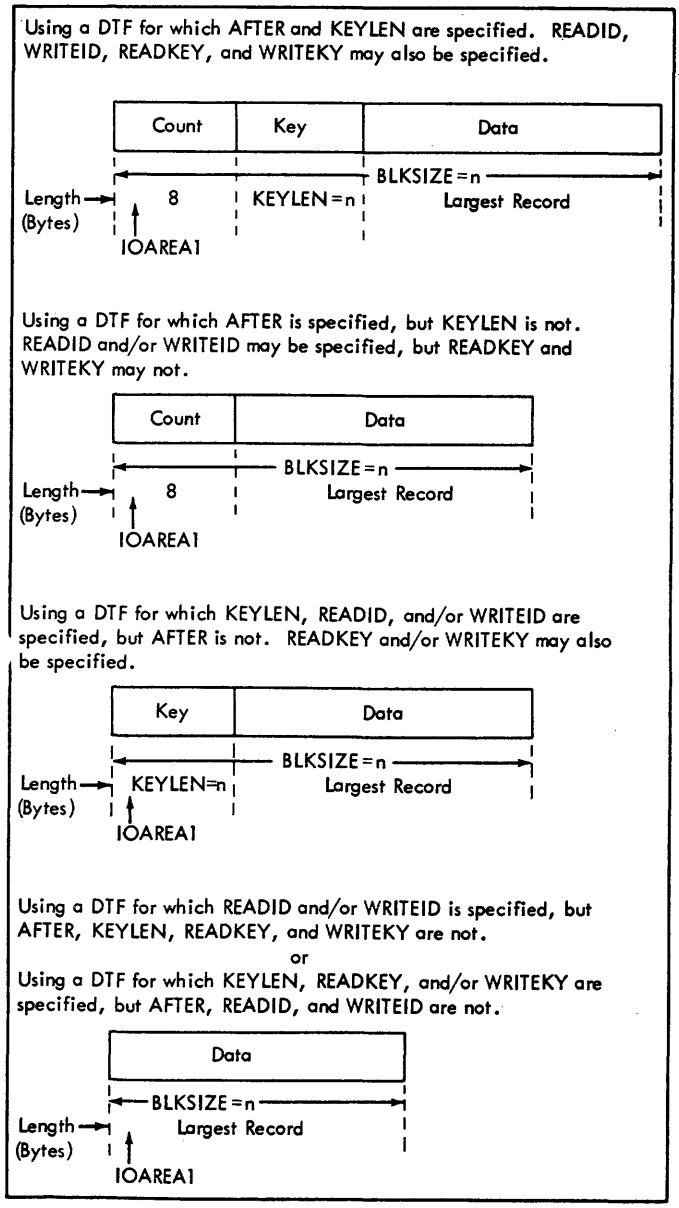


Figure 27. Schematic of I/O Area in Main Storage, for DAM

REFERENCE METHODS

With the direct access method of processing, each record that is read or written is specified by providing IOCS with two references:

- Track reference. This gives the track on which the desired record is located.
- Record reference. This may be either the record key (if the records contain key areas) or the record identifier (ID).

IOCS seeks the specified track, searches it for the individual record, and reads or writes the record as indicated by the macro instruction. If a specified record is not found, IOCS sets a no-record-found indication in the user's error/status byte, as specified by the DTFDA entry ERRBYTE. This indication can be tested by the problem program, and additional processing can be programmed to suit the user's requirements.

Multiple tracks can be searched for a record specified by key (SRCHM). If a record is not found after an entire cylinder (or the remainder of a cylinder) is searched, an end-of-cylinder bit is turned on instead of NRF in ERRBYTE.

When an I/O operation is started, control returns immediately to the problem program. Therefore, when the program is ready to process the input record, or build the succeeding output record for the same file, a test must be made to ensure that the previous transfer of data is complete. Do this by issuing a WAITF macro instruction in the problem program.

After a READ or WRITE instruction for a specified record has been executed, IOCS can make the ID of the next record available to the problem program. The WAITF macro should be issued to assure that the data transfer is complete. The user must set up a field (in which IOCS can store the ID) to request that IOCS supply the ID. He must also specify the symbolic address of this field in the DTFDA entry IDLOC.

When record reference is by key and multiple tracks are searched, the ID of the specified record (rather than the next record) is supplied. The function of supplying the ID is useful for a random updating operation, or for the processing of successive DASD records. If the user is processing consecutively on the basis of the next ID and does not have an end-of-file record, he can check the ID supplied by IOCS against his file limits to

determine when he has reached the end of the logical file.

Track Reference

To provide IOCS with the track reference, the user sets up a track reference field in main storage, assigns a symbolic name in the DTFDA entry SEEKADR, and determines by DTFDA operand specifications which type of addressing system to use. Before issuing any read or write instruction for a record, the user must store the proper track identifier in either the first seven hexadecimal bytes (mbbcchh), first three hexadecimal bytes (ttt), or first eight zoned decimal (tttttttt) bytes of this field. The latter two track references, along with the DSKXTNT and RELTYPE operands, indicate that relative addressing is to be performed. Thus, instead of providing the exact physical track location (mbbcchh), only the track number relative to the starting track of the file need be provided. If these operands are omitted, the physical addressing system is assumed.

The fields for each of these track reference systems are shown in Figure 28. For reference to records by record number, r or rr is used (see Record Reference: Identifier). When the READ or WRITE is executed, IOCS refers to this field to select the specific track on the appropriate DASD.

Record Reference

The direct access method allows records to be specified by either record key or record identifier.

Keys

If records contain key areas, the records on a particular track can be randomly searched by their keys. This allows the user to refer to records by the logical control information associated with the records, such as an employee number, a part number, a customer number, etc.

For this type of reference the programmer must specify the symbolic name of a main-storage key field in the DTFDA entry KEYARG. He then stores each desired key in this field.

Bytes	Decimal Identifier	Contents in Zoned Decimal	Information
0-7	tttttttt	0-16777215	Track number relative to the first track of the file.
8-9	rr	0-99	Record number relative to the first record on the track. If reference is by key, rr should be zero.
Bytes	**Hexadecimal Identifier**	**Contents in Hexadecimal**	**Information**
0-2	ttt	0-FFFFFF	Track number relative to the first track of the file.
3	r	0-FF	Record number relative to the first record on the track. If reference is by key, r should be zero.

Figure 28. Types of Track Reference Fields (Part 1 of 2)

Bytes	Physical Identifier	Contents in Hexadecimal	Information
0	m	0-DD	Number of the volume on which the record is located. Volumes and their symbolic units for a file must be numbered consecutively. The first volume number for each file must be zero, but the first symbolic unit may be any SYSnnn number. The system references the volume by adding its number to the first symbolic unit number. Example 1: The first extent statement // EXTENT SYS005,...and M=0 results in the system referencing SYS005. Example 2: // VOL SYS005,... and M=2 results in the system referencing SYS007 (previous job control standard label card).
1-2	b,b	0,0 (disk) 0,0-9 (cell)	For 2321 the first byte is zero. The cell number (0-9) is specified in the second byte. These two bytes are always zero for disk storage references.
3-4	c,c	0,0-C7 (disk) 0-13,0-9 (cell)	For disk the number of the cylinder (0-199) in which the record is located. The first byte is always zero, and the second byte specifies one of the available cylinders in a disk pack. These two bytes with the next two (hh) provide the track identification. For 2321 the number of the subcell (0-19) is located in the first byte. One of the ten strips (0-9) is located in the second byte. Note: The last four strips on each cell are reserved for alternate tracks.
5-6	h,h	0,0-9 (2311 disk) 0,0-13 (2314 or 2319 disk) 0-4,0-13 (2321 cell)	For disk the number of the read/write head that applies to the record. The first byte is always zero, and the second byte specifies one of the disk surfaces in a disk pack. For 2321 the first byte (0-4) specifies one of the five head bar positions (equivalent to cylinder on disk). The second byte (0-19) specifies one of the twenty head elements.
7	r	0-FF	Sequential number of the record on the track. Note: r = 0 if reference is by key.

Figure 28. Types of Track Reference Fields (Part 2 of 2)

Identifier (ID)

Records on a particular track can be randomly searched by their position on the track, rather than by control information (key). To do this, use the record identifier. The record identifier, which is part of the count area of the DASD record, consists of five bytes (CCHHR). The first four bytes (cylinder and head) refer to the location of the track and the fifth byte (record) uniquely identifies the particular record on the track. When records are specified by ID, they should be numbered in succession without missing

numbers on each track. The first data record on a track should be record number 1, the second number 2, etc.

Whenever records are identified by a record ID, the r-byte of the track-reference field (Figure 28) must contain the number of the desired record. When a READ or WRITE instruction that searches by ID is executed, IOCS refers to the track-reference field to determine which record is requested by the program. The number in this field is compared with the corresponding fields in the count areas of the disk records. The r-byte or bytes specifies the particular record on the track.

CREATING A FILE OR ADDING RECORDS BY DAM

The problem program can preformat a file or an extension to an existing file in one of two ways depending on the type of processing to be done. If the WRITE AFTER macro is used exclusively, the WRITE RZERO macro is enough for preformatting the tracks. If nonformatting functions of the WRITE macro are used, the tracks should be preformatted with an IBM-supplied utility program (Clear Disk). The Clear Disk utility also resets the capacity record to reflect an empty track.

In addition to reading, writing, and updating records randomly, the direct access method permits the user to create a file or write new records on a file. When this is done, all three areas of a DASD record are written: the count area, the key area (if present), and the data area. The new record is written after the last record previously written on a specified track. The remainder of the track is erased. This method is specified in a WRITE instruction by the parameter AFTER.

IOCS ensures that each record fits on the track specified for it. If the record fits, IOCS writes the record. If it does not fit, IOCS sets a no-room-found indication in the user's error/status byte (specified by the DTFDA entry ERRBYTE). If WRITE AFTER is specified, IOCS also determines (from the capacity record) the location where the record is to be written.

Whenever the AFTER option is specified, IOCS uses the first record on each track (R0) to maintain updated information about the data records on the track. Record 0 (Figure 29) has a count area and a data area, and contains the following:

Count Area

Flag (not normally transferred to main storage)

Physical Identifier

Key Length (KL)

Data Length (DL)

Data Area (8 bytes)

5 Bytes--Physical ID of last record written on track (cchhr).

2 Bytes--Number of unused bytes remaining on track.

1 Byte--For the DAM on the Operating System/360.

Each time a WRITE AFTER instruction is executed, IOCS updates the data area of this record.

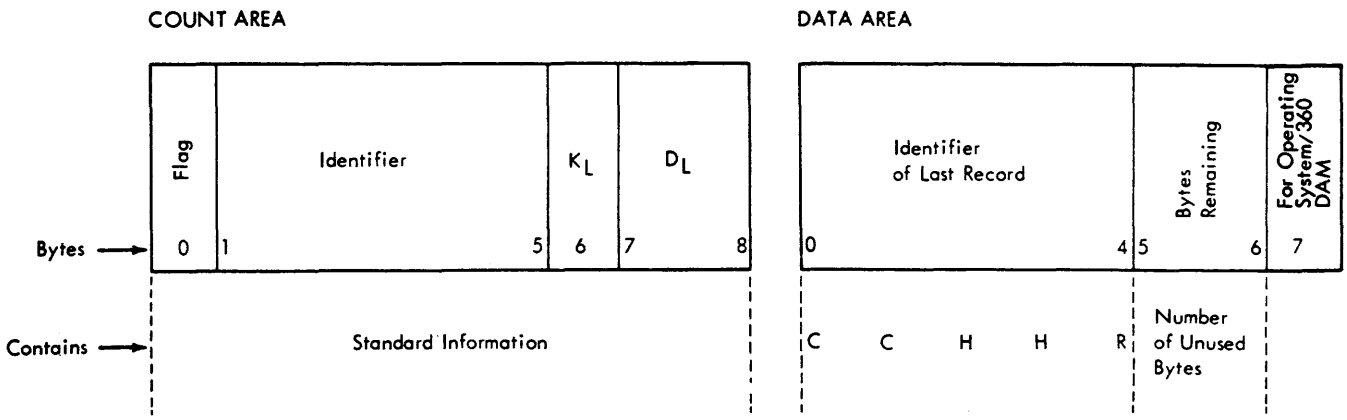


Figure 29. Contents of Record 0 for Capacity-Record Option

Direct Access Macros

Before a direct access file can be processed, it must be defined by the DTFDA declarative macro. After defining the file, the user can operate with that file using imperative macros. The file must be initialized, processed, and deactivated.

DIRECT ACCESS FILE (DTFDA)

The DTFDA detail entries that apply to a file when records are processed by the Direct Access Method are explained here and shown in Figure 33.

Enter the symbolic name of the file in the name field and DTFDA in the operation field.

```
AFTER=YES
```

This operand must be included if any records (or an additional record) are written in a file by a format WRITE (count, key and data) following the last record previously written on a track. The remainder of the track is erased. That is, whenever the macro instruction WRITE filename, AFTER or WRITE filename, RZERO is used in a program, this entry is required.

```
BLKSIZE=n
```

This operand indicates the size of the I/O area by specifying the maximum number, n, of characters that are transferred to or from the area at any one time. When undefined records are read or written, the area must be large enough to accommodate the largest record.

If key length is specified by DTFDA KEYLEN, and if macro instructions that transfer the key areas of records are issued, this area must provide for both the key area and data area of a record (see IOAREA1 and Figure 32). The length must specify an additional eight bytes if AFTER=YES is specified. If a file is created or additional records are written in a file, the count area of the records must be specified, and the AFTER=YES operand must be included.

When processing variable-length records, the value of n must be within the track capacity of the device containing the file. When processing spanned records, the value of n includes the eight bytes required for the block descriptor word, but it does not include the key length, if specified, or the count. For spanned records, the value of n must be larger than eight bytes and less than 32,768 bytes.

IOCS uses this specification to construct the count field of the CCW for reading or writing fixed-length and variable length records.

```
|CONTROL=YES                              |
```

This operand must be included if a CNTRL macro instruction is issued for this file. The CNTRL macro for seeking on the 2311, 2314, or 2319 allows the user to specify a track address to which access movement should begin for the next READ or WRITE instruction for a file. While the arm is moving, the programmer may process data and/or request I/O operations on other devices.

For the 2321, the CNTRL macro enables the user to seek to a specific address or to restore the strip to its subcell.

```
|DEVADDR=SYSnnn                           |
```

This operand must specify the symbolic unit (SYSnnn) associated with a file if the extent statement symbolic unit is not provided. If such a unit is provided, its specification overrides the DEVADDR parameter. This specification, or symbolic unit, represents an actual I/O address and is used in the job control ASSGN statement to assign the actual I/O device address to the file.

```
|DEVICE={2311|2314|2321}                  |
```

This operand specifies whether the logical file is on a 2311, 2314, 2319, or 2321 DASD. If this entry is omitted, 2311 is assumed.

Note: Specify 2314 for 2319.

```
|DSKXTNT=n                                |
```

The n indicates the maximum number of extents (up to 256) that are specified for a file. When RECFORM=FIXUNB, VARUNB, or UNDEF, and DSKXTNT=n, is specified, it indicates that a relative ID is used in the SEEKADR and IDLOC locations. If DSKXTNT=n is omitted, a physical ID is assumed in the SEEKADR and IDLOC locations.

If RECFORM=SPNUNB is specified, then DSKXTNT is required. If relative addressing is used, RELTYPE=DEC or HEX must also be specified.

```
|ERRBYTE=name                             |
```

This operand is required for IOCS to supply indications of exceptional conditions to the problem program. The symbolic name of a 2-byte field (in which IOCS can store the error-condition or status codes) is entered after the = sign.

The ERRBYTE codes are available for testing by the problem program after the attempted transfer of a record is complete. After testing the ERRBYTE status code, the problem program can return to IOCS by issuing another macro instruction. One or more of the error status indication bits may be set to 1 by IOCS as in the bits indicated in Figure 30.

```
|ERREXT=YES                               |
```

This operand enables unrecoverable I/O errors (occurring before a data transfer takes place) to be indicated to the problem program. This error information is indicated in the ERRBYTE-name bits and is available after the WAITF macro instruction is issued.

```
|FEOVD=YES                                |
```

This operand is specified if coding is generated to handle end-of-volume records. It should be specified only when reading a file that was built using sequential disk and the FEOVD macro was used.

Byte	Bit	Error/Status Code Indication	Explanation
0	0	---	---
0	1	Wrong-length record	The wrong-length record indication is applicable for undefined records or fixed-length records. Fixed-Length Records: This bit is set on under the following conditions: • A READ KEY or WRITE KEY is issued, and the keylength differs from the length as specified by KEYLEN=n. No data is transferred. • A READ KEY is issued, and the data length differs from the specified length (BLKSIZE minus KEYLEN, or BLKSIZE minus the value of KEYLEN plus 8 if AFTER=YES was specified). • A READ ID is issued, and the record length differs from the specified length (BLKSIZE, or BLKSIZE minus 8 if AFTER=YES was specified). • A WRITE KEY is issued, and the data length of the record is greater than specified in the count field in the DASD record on disk. The original record positions are filled, and the remainder of the updated record is truncated and lost. • A WRITE ID is issued, and the record length is greater than specified in the count field in the DASD record on disk. The original record positions are filled, and the remainder of the updated record is truncated and lost. Note: If an updated record is shorter than the original record, it is padded with binary zeros to the length of the original record. The wrong-length record bit is not set on.

Figure 30. ERRBYTE Error Status Indication Bits (Part 1 of 5)

Byte	Bit	Error/Status Code Indication	Explanation
0	1	Wrong-length record (continued)	Undefined-Length Records: This bit is set on under the following conditions: • A READ KEY or WRITE KEY is issued, and the keylength differs from the length as specified by KEYLEN=n. No data is transferred. • A READ KEY is issued, and the data length is greater than the maximum data size (BLKSIZE minus KEYLEN, or BLKSIZE minus the value of KEYLEN plus 8 if AFTER=YES was specified). IOCS supplies the actual data length of the record read in the RECSIZE register. • A READ ID is issued, and the record length is greater than the maximum record length (BLKSIZE, or BLKSIZE minus 8 if AFTER=YES was specified). IOCS supplies the actual data length of the record read in the RECSIZE register. • A WRITE (KEY, ID, or AFTER) is issued, and the data length of the record (loaded into the RECSIZE register) is greater than the maximum data size (BLKSIZE minus KEYLEN, or BLKSIZE minus the value of KEYLEN plus eight if AFTER=YES was specified). The length of the record written is equal to the maximum data size. • A WRITE KEY is issued and the data length (loaded into the RECSIZE register) is greater than specified in the count field of the DASD record on disk. The original record positions are filled, and the remainder of the updated record is truncated and lost. • A WRITE ID is issued, and the record length is greater than specified in the count field of the DASD record on disk. The original record positions are filled, and the remainder of the updated record is truncated and lost. Note: If an updated record is shorter than the original record, it is padded with binary zeros to the length of the original record. The wrong-length record bit is not set on.

Figure 30. ERRBYTE Error Status Indication Bits (Part 2 of 5)

Byte	Bit	Error/Status Code Indication	Explanation
0	1	Wrong-length record (continued)	Variable-length records: This bit is set on under the following conditions: • When a READ is issued and the LL count is greater than the maximum value specified by the BLKSIZE operand, the wrong-length record indicator is set on. • When a nonformatting WRITE is issued and the record is larger than the physical record on the device, the record is written with the low-order bytes truncated and the wrong-length record indicator is set ON. The indicator also is set ON if the record is shorter than the physical record, but the low-order bytes of the physical record are padded with binary zeros. • When a formatting WRITE is issued and the LL count is greater than the maximum specified block size, the record is written with the low-order bytes truncated and the wrong-length record indicator is set on. Spanned Records: This bit is set on under the following conditions: • When a READ is issued and the logical record size is larger than the value specified, only the number of bytes specified by BLKSIZE is read and the wrong-length indicator is set on. • When a nonformatting WRITE is issued and the record length is not the same length as the record being processed, the wrong-length record indicator is set ON. If the length specified is longer than the record being processed, the low-order bytes are ignored. If the length specified is less than the record being processed, it is padded with binary zeros. • If a formatting WRITE is issued, the wrong-length indicator is set ON when the logical record size is larger than the size specified with BLKSIZE and the record is truncated to the size specified with BLKSIZE. • If the first physical record encountered is not an only or first segment, the wrong-length indicator is set on. The no-record-found indicator is also set ON. • If another first segment is encountered after the first segment is read out before a middle or last segment, the wrong-length indicator is set ON.

Figure 30. ERRBYTE Error Status Indication Bits (Part 3 of 5)

Byte	Bit	Error/Status Code Indication	Explanation
0	2	Nondata Transfer Error	The block in error was neither read nor written. If ERREXT is specified and this bit is OFF, transfer took place and the problem program should check for other errors in the ERRBYTE field.
0	3	---	---
	4	No room found	The no-room-found indication is applicable only when the WRITE AFTER form of the macro is used for a file. The bit is set on if IOCS determines that there is not enough room left on the track to write the record. The record is not written.
0	5	---	---
0	6	---	---
0	7	Reference outside extents	The relative address given is outside the extent area of the file. No I/O activity has been started and the remaining bits should be OFF. If IDLOC is specified, its value is set to 9s for a zoned decimal ID or Fs for hexadecimal ID.
1	0	Data check in count area	This is an unrecoverable error.
1	1	Track overrun	The number of bytes on the track exceeds the theoretical capacity.
1	2	End of cylinder	The end-of-cylinder indicatiON bit is set on when SRCHM is specified for READ or WRITE KEY and the end-of-cylinder is reached before the record is found. If IDLOC is also specified, certain conditions also turn this bit ON. For further information, see IDLOC under DTFDA.
1	3	Data check when reading key or data	This is an unrecoverable error.
1	4	No record found	The no-record-found indication is given when a search ID or key is issued and a record is not found. This applies to both READ ccmmands and WRITE commands and may be caused by: a. The record searched for does not exist in the file. b. The record cannot be found because of a machine error (that is, incorrect seek). For spanned record processing, if the first physical record encountered is nct the first or only segment, then the no-record-found indicator is set on.

Figure 30. ERRBYTE Error Status Indication Bits (Part 4 of 5)

Byte	Bit	Error/Status Code Indication	Explanation
1	5	End of file	The end-of-file indication is applicable only when the record to be read has a data length of zero. The ID returned in IDLOC, if specified, is hexadecimal FFFFF. The bit is set only after all the data records are processed. For example, in a file having n data record (record n + 1 is the end-of-file record), the end-of-file indicator is set ON when the user reads the n + 1 record. This bit is also posted when an end of volume marker is detected. It is the user's responsibility to determine if this bit means true EOF or end of volume on a SAM file.
1	6	End of volume	The end-of-volume indication is given in conjunction with the end-of-cylinder indication. This bit is set ON if the next record ID (n + 1, 0, 1) that is returned on the end of the cylinder is higher than the volume address limit. The volume address limit is cylinder 199, head 9, for a 2311 disk pack cylinder 199, head 19, for a 2314 or 2319 disk pack, and subcell 19, strip 5, cylinder 4, head 19 for a data cell. These limits allow for the reserved alternate track area. If both the EOC and EOV indicators are set on, the ID returned in IDLOC is FFFFF.
1	7	Not used	

Figure 30. ERRBYTE Error Status Indication Bits (Part 5 of 5)

HOLD=YES

This operand is specified only if the track hold function is specified:

- At system generation time, or
- Included in the DAMOD macro, or
- Use when the file is referenced.

If the SRCHM operand is used, only the first track IOCS seeks is protected.

When a READ is issued while processing spanned records, the track containing the first segment is held until the user releases it. If a formatting WRITE macro is issued, DAMOD reads ahead to determine if enough space exists to write the record. All the tracks required to write the record are held and then released, one by one, as they are written.

IDLOC=Name

This operand is included if the programmer wants IOCS to supply the ID of a record after each READ or WRITE (ID or KEY) is completed. The symbolic name of a record reference field (in which IOCS is to store the ID) is specified after the = sign in this operand. WAITF should be used before referencing this field.

IOCS supplies the ID in the same form used in the SEEKADR location. The ID forms, given in Figure 28, are supplied in IDLOC in the same format except when physical IDs are used. Only the last five bytes of the physical ID (cchhr) are supplied as compared with the complete relative ID including preceding zeros.

IOCS supplies the ID of the record specified in the READ/WRITE instruction, or the ID of the next record location. The following may occur when this option is taken.

- Whenever a READ or WRITE ID (or if a READ or WRITE KEY without SRCHM) is issued, the address returned is that of the next record location.

  Exception: When the record to be read or written is the last record of the cylinder, an end-of-cylinder indication is posted in ERRBYTE1, bit 2, and the address returned is that of the first record of the next cylinder. If, in addition, the end-of-volume indication is posted, the address returned in IDLOC is all 1 bits.

- Whenever a READ or WRITE KEY with SRCHM is specified, the address returned is that of the same record location.

  Exception: When the record is not found, an end-of-cylinder condition is posted and the information returned is unpredictable.

If a READ or WRITE (ID or KEY) is issued for spanned records, the address returned is that of the first segment of the record whose IDLOC is requested. The ID (physical ID only), is located in the first five bytes of the count field in the IOAREA1 area.

For more information on the SRCHM specification see Figure 31.

MACRO INSTRUCTION	ID SUPPLIED (Normal I/O Completion)	
	With SRCHM	Without SRCHM
READ Filename,KEY	Same record	Next record
READ Filename,ID	Next record	Next record
WRITE Filename,KEY	Same record	Next record
WRITE Filename,ID	Next record	Next record
WRITE Filename,RZERO	Dummy Record	Dummy Record
WRITE Filename,AFTER[,EOF]	Dummy Record	Dummy Record

Figure 31. ID Supplied After a READ or WRITE Instruction

If IDLOC is specified and end of cylinder is reached on a 2311, 2314, or 2319.file, the cylinder number is increased by 1, the head number is set to 0, and the record number is set to 1. On a 2321 file, an end-of-cylinder condition with IDLOC specified causes the high-order position of the head number to be increased by 1, the low-order position of the head number is set to 0, and the record number is set to 1. An overflow from the high-order position of the head number causes the low-order position of the cylinder number to be increased by 1, and the high-order position of the head number is set to 0. The low-order position of the head number is 0, and the record number is set to 1. Subsequent overflows of address locations increase the next higher positions of the addresses. It is the user's responsibility to check the validity of the address returned in IDLOC.) When using relative addressing with IDLOC specified, all user extents (except the last extent for each file) should end on cylinder boundaries.

```
|IOAREA1=Name                                                    |
```

This operand must be included to specify the symbolic name of the input/output area used by the file. The input/output routines transfer records to or from this area. The specified name must be the same as the name used in the DS instruction that reserves this area of main storage.

The main-storage input/output area must be large enough to contain the maximum number of bytes required in any READ or WRITE instruction issued for a file in the problem program. This is affected by the length of record data areas, and by the use of the count and key areas as follows:

- If undefined records are specified in the DTFDA entry RECFORM, the area must provide space for the largest data record that will be processed.

- If variable or spanned records are specified in the DTFDA entry RECFORM, the area must be large enough to contain the largest logical record in the file, plus an additional eight bytes for control words. The user must place the first byte of his record in the ninth byte of the I/O area for all write operations. He must also place the record length in the length field, bytes 4 and 5 of the I/O area.

  When a READ macro is issued, the record length is in bytes 4 and 5 of the I/O area and the first byte of the record is in the ninth byte of the I/O area.

- If the DTFDA entry KEYLEN is specified and any instructions that read or write the key area of a record are issued in

the problem program, the input/output area must provide room for the key area as well as the data area. The length needed for the key is the length specified in KEYLEN.

- If any write instructions that transfer the count area to a disk record are issued in the problem program, eight bytes of main storage must be allotted at the beginning of the I/O area. In these eight bytes, IOCS constructs the count field to be transferred to disk.

Whenever a WRITE instruction is issued, IOCS assumes that the input/output area (see Figure 27) contains the information implied by the type of instruction that is being executed (Figure 32).

MACRO INSTRUCTION	I/O AREA CONTENTS	
	With KEYLEN	Without KEYLEN
READ Filename,KEY	Data	
READ Filename,ID	Key and Data	Data
WRITE Filename,KEY	Data	
WRITE Filename,ID	Key and Data	Data
WRITE Filename,RZERO	Anything	Anything
WRITE Filename,AFTER[,EOF]	Count, Key, and Data	Count and Data

Figure 32. I/O Area Requirements for DAM

KEYARG=Name

This operand must be included if records are identified by key. That is, if the macro instruction (READ filename,KEY) or (WRITE filename,KEY) are issued in the problem program, this entry and the corresponding KEYLEN operand is required. KEYARG specifies the symbolic name of the key field in which the user supplies the record key for the READ/WRITE routines.

The KEYARG operand is required for formatting WRITE (WRITE filename AFTER) operations for files containing keys. It is required also when READ filename ID is specified and if KEYLEN is not zero. When record reference is by key, IOCS uses this specification at assembly time to construct the data address field of the CCW for search commands.

KEYLEN=n

This operand must be included if record reference is by key or if keys are read or written. It specifies the number, n, of bytes in each key. All keys must be the same length. If this card is omitted, IOCS assumes a key length of zero.

If there are keys recorded on DASD and this entry is absent, a WRITEID or READID reads or writes the data portion of the record.

When record reference is by key, IOCS uses this specification to construct the count field of the CCW for this file. IOCS also uses this in conjunction with IOAREA1 to determine where the data field in the main-storage I/O area is located (see IOAREA1).

LABADDR=name

The user may require one or more user labels in addition to the standard file label. If so, he must include his own routine to check, or write, the labels. The symbolic name of such a routine is specified in this entry. IOCS branches to this routine after it has processed the standard label. See Writing and Checking User Standard Labels for a complete discussion of the function of the LABADDR routine.

MODNAME=name

This operand specifies the name of the logic module that is used with the DTF table to process the file. If the logic module is assembled with the program, the MODNAME in the DTF macro instruction must specify the same name as the DAMOD macro instruction. If this entry is omitted, standard names are generated for calling the logic module. If two DTF macro instructions call for different functions that can be handled by a single module, only one module is called.

```
|RDONLY=YES                              |
```

This operand is specified if the DTF is used with a read only module. Each time a read only module is entered, register 13 must contain the address of a 72-byte doubleword aligned save area. Each DTF should have its own uniquely defined save area. Each time an imperative macro (except OPEN(R), LBRET, SETL, or SETFL) is issued using a particular DTF, register 13 must contain the address of the save area associated with that DTF. The fact that the save areas are unique for each task makes the module reentrant (that is, capable of being used concurrently by several tasks). For more information see Shared Modules and Files.

If an ERROPT or WLRERR routine issues I/O macro instructions using the same read only module that caused control to pass to either error routine, the problem program must provide another save area. One save area is used for the normal I/O, and the second for I/O in the ERROPT or WLRERR routine. Before returning to the module that entered the ERROPT routine, register 13 must be set to the save area address originally specified for that DTF. If the operand is omitted, the module generated is not reenterable and no save area need be established.

```
|READID=YES                              |
```

This operand must be included if any input records are specified by ID (identifier) in the problem program. That is, whenever the macro instruction READ filename,ID is used in the program, this entry is required.

```
|READKEY=YES                             |
```

This operand must be included if any input records are specified by key in the problem program. That is, whenever the macro instruction READ filename,KEY is used in the program, this entry is required.

```
|RECFORM={FIXUNB|SPNUNB|UNDEF|VARUNB}   |
```

This operand specifies the type of records in the input, or output, file. Either of the following specifications may be entered immediately after the = sign:

FIXUNB For fixed-length records. All records are considered unblocked in the DAM method. If the user wants blocked records, he must provide his own blocking and deblocking.

SPNUNB For spanned records. This specification is for unblocked variable length logical records of less than 32,768 bytes per record.

UNDEF For undefined records. This specification is required only if the records are not fixed length.

VARUNB For format V (variable length) records. This specification is for unblocked variable length records.

```
|RECSIZE=(r)                            |
```

This operand must be included if undefined records are specified (RECFORM=UNDEF). It specifies the number (r) of the general-purpose register that contains the length of each individual input or output record. This may be any register 2-12.

Whenever each undefined record is read, IOCS supplies the length of the data area for that record in the specified register.

When an undefined record is written in a file, the programmer must load the length of the data area of the record (in bytes) into this register, before he issues the WRITE instruction for the record. IOCS adds the length of the key when required.

When records are written in the file (AFTER specified in the WRITE instruction), IOCS uses the length to construct the count area written on DASD. IOCS adds the length of both the count and the key when required.

```
|RELTYPE={DEC|HEX}
```

This operand specifies whether the zoned decimal (DEC) or hexadecimal (HEX) form of the relative ID is being used. When RECFORM=FIXUNB, VARUNB, or UNDEF, then RELTYPE should only be supplied if the DSKXTNT operand (relative ID) is specified. If omitted, a hexadecimal relative ID is assumed. However, if DSKXTNT is also omitted, a physical ID is assumed in the SEEKADR and IDLOC addresses.

When RECFORM=SPNUNB, RELTYPE must be specified when relative addressing is used. If RELTYPE is omitted, a physical ID is assumed in the SEEKADR and IDLOC addresses.

```
|SEEKADR=name
```

This operand must be included to specify the symbolic name of the user's track-reference field. In this field, the user stores the track location of the particular record read or written. The READ, WRITE, and CNTRL routines refer to this field to determine which volume and which track contains the desired record. Whenever records are to be located by searching for a specified ID, the track-reference field must also contain the number of the record on the track. See Figure 28 for the types of track reference fields that can be used.

```
|SEPASMB=YES
```

This operand must be included if the DTF is assembled separately. This causes a CATALR card with the filename to be punched ahead of the object deck and defines the filename as an entry point in the assembly.

```
|SRCHM=YES
```

If input/output records are identified by key, this entry may be included to cause IOCS to search multiple tracks for each specified record. The instruction (READ filename,KEY) or (WRITE filename,KEY) searches the track specified in the track-reference field and all following tracks in the cylinder, until the record is found or the end of the cylinder is reached. If the logical file ends before the end of the cylinder and the record is not found, the search continues into the next file, if any, on the cylinder. EOC, instead of NRF, is indicated. Without SRCHM=YES, each search is confined to the specified track.

```
|TRLBL=YES
```

This operand, if specified with the LABADDR operand, indicates that user standard trailer labels are to be read or written following the user standard header labels on the user label track. Both operands must be specified for trailer label processing. For more information on processing labels, see the Label Processing section.

```
|TYPEFLE={INPUT|OUTPUT}
```

This operand must be included to indicate how standard volume and file labels are to be processed:

INPUT Standard labels are to be read.

OUTPUT Standard labels are to be written.

Because logical files on DASD must always contain labels, this entry is always required.

```
|VERIFY=YES
```

This operand is included if the user wants to check the parity of disk records after they are written. VERIFY is always assumed when 2321 records are written. If this operand is omitted, any records written on 2311, 2314, or 2319 are not verified.

```
|WRITEID=YES
```

This operand must be included if the DASD storage location for writing any output record or updating an input file is specified by a record ID (identifier). That is, whenever the macro instruction (WRITE filename,ID) is used in the program, this operand is required.

WRITEKY=YES

This operand must be included if the DASD location for writing any output record or updating an input file is specified by record key. That is, whenever the macro instruction WRITE filename,KEY is used in the program, this operand is required.

XTNTXIT=name

This operand is included if the programmer wants to process label extent information. It specifies the symbolic name of the user's XTNTXIT routine. During an OPEN, IOCS branches to the user's routine after each specified extent is checked and validated. Upon entering the user's routine, IOCS stores in register 1 the address of a 14-byte field that contains the label extent information (in binary form).

Bytes	Contents
0	Extent type code (as specified in the extent statement)
1	Extent sequence number
2-5	Lower limit of the extent (cchh)
6-9	Upper limit of the extent (cchh)
10-11	Symbolic unit number (in hexadecimal format)
12	Old bin number
13	Present bin number of the extent (B2)

The user returns to IOCS by use of the LBRET macro instruction.

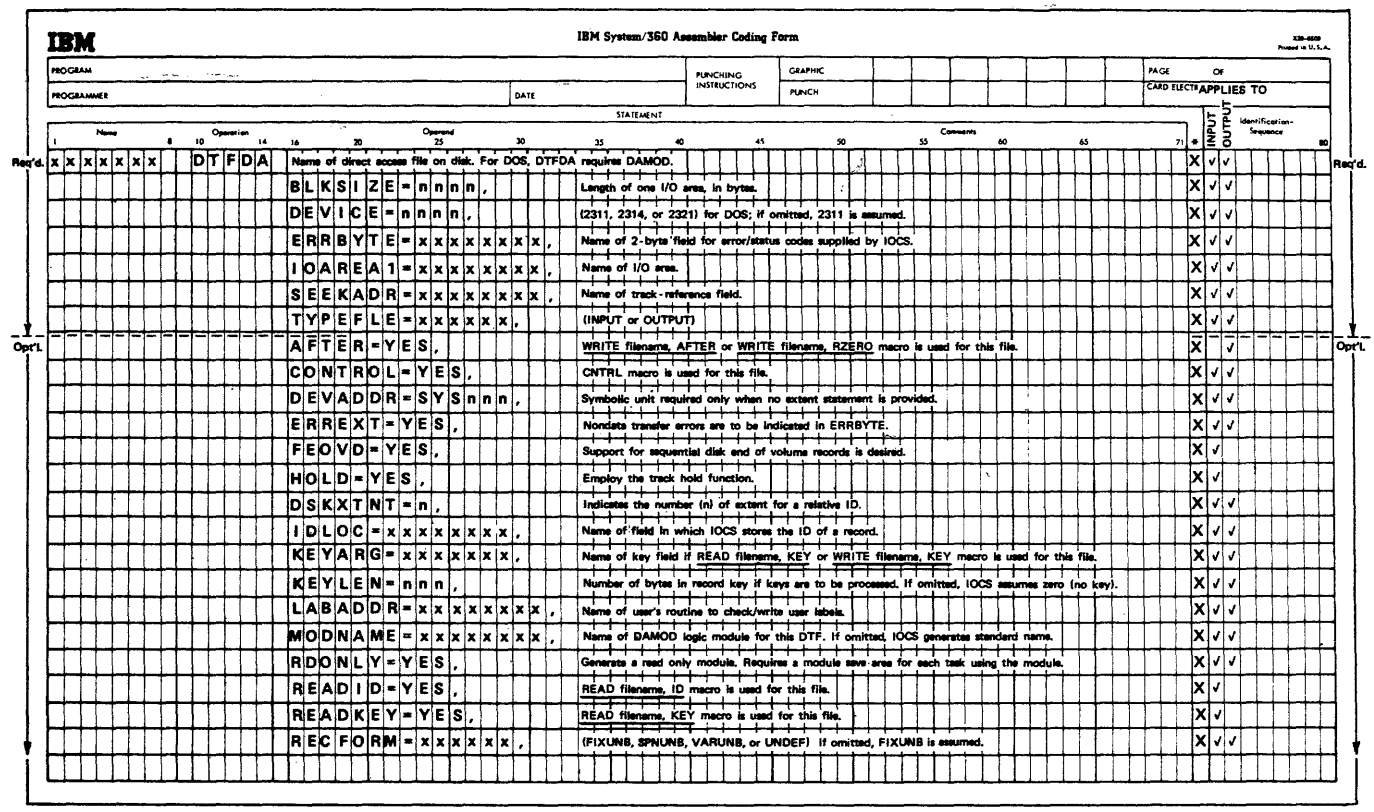
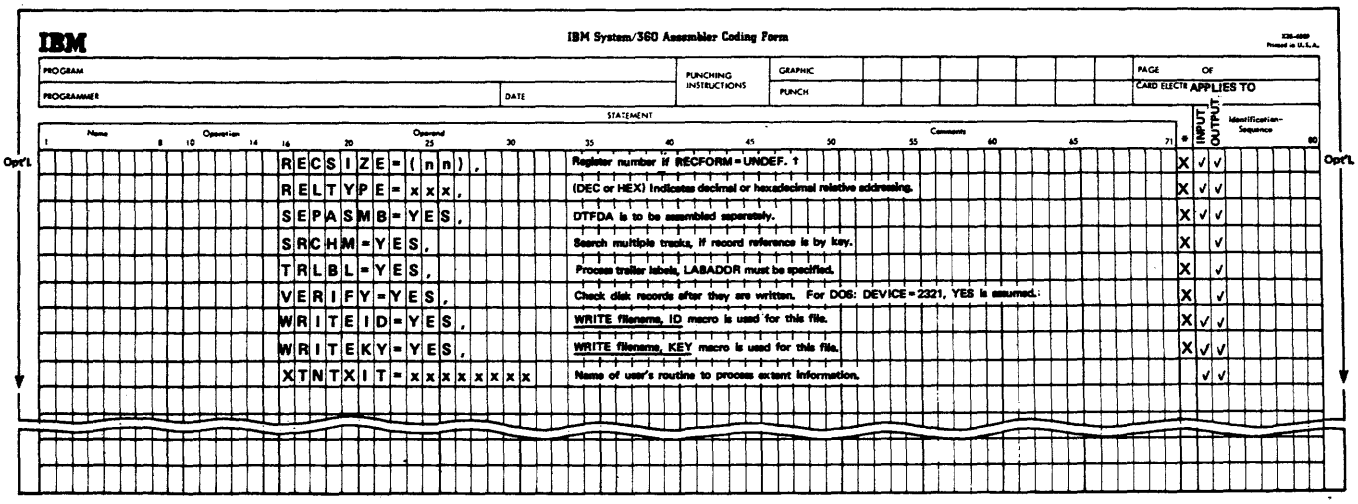


Figure 33. DTFDA Macro (Part 1 of 2)



* Header and each detail card, except the last one in each set, must have a continuation punch in column 72. Also each detail card, except the last one, must contain a comma immediately after the operand. Space is allowed for the longest operand plus the comma. If a smaller operand is used, the comma should be moved over accordingly. In the last detail card of a set, the comma position must be blank.

† General registers 2-12, written in parentheses; for example: (12).

Figure 33. DTFDA Macro (Part 2 of 2)

DIRECT ACCESS MODULE (DAMOD)

A set of DAMOD entries is included for each DAM logic module necessary to support each DTFDA macro in a particular problem program. The logic modules are described by a DAMOD header entry and a series of keyword parameters.

The header entry contains DAMOD in the operation field and may contain a user module name in the name field. The parameters are explained here and shown in Figure 34.

Name	Operation	Operand	Remarks
[Modname]	DAMOD[1] DAMODV[2]		Must be included.
		AFTER=YES	When WRITE with the operand AFTER or RZERO is used.
		ERREXT=YES	Required if nondata transfer error conditions are to be indicated in the ERRBYTE status bits.
		FEOVD=YES	Required if support for sequential disk end of volume records is desired.
		HOLD=YES	Required if the track hold function is to be used.
		IDLOC=YES	Required if IDLOC specified in DTFDA.
		RDONLY=YES	Required if a read only module is to be generated.
		RECFORM={FIXUNB[1] / UNDEF[1] / VARUNB[2] / SPNUNB[2]}	Describes record format.
		RELTRK=YES	Required if the module is to process relative identifiers along with physical identifiers.
		SEPASMB=YES	If the module is assembled separately.

1 - DAMOD is for fixed length unblocked and undefined records.
2 - DAMODV is for variable length unblocked and spanned unblocked records.

Figure 34. DAMOD Macro

AFTER=YES

This operand generates a logic module that can format write (count, key, and data). It performs the functions required by WRITE filename,AFTER and WRITE filename,RZERO. The module also processes any files in which the AFTER operand is not specified in the DTF.

HOLD=YES

This operand is specified if the track hold function is:

- Specified at system generation time,
- Included in the DTFDA macro,
- Referenced.

For more information see the HOLD operand under the DTFDA macro.

ERREXT=YES

Include this operand if nondata transfer error conditions are indicated in the ERRBYTE status indication bit.

FEOVD=YES

This operand is specified if coding is to handle end-of-volume records. It should be specified only if the user is reading a file built using sequential disk and the FEOVD macro.

IDLOC=YES

This operand generates a logic module that returns record identifier (ID) information to the user. The module also processes any files in which the IDLOC operand is not specified in the DTF.

RDONLY=YES

This operand generates a read only module. RDONLY=YES must be specified in the DTF.

```
|RECFORM={UNDEF|FIXUNB|SPNUNB|VARUNB}      |
```

If UNDEF is specified, the logic module generated can handle both unblocked fixed length and undefined records. If the entry is omitted, or if FIXUNB is specified, the logic module generated can handle only fixed-length unblocked records. If SPNUNB is specified, the module can handle both format V (variable length) and spanned format records. If VARUNB is specified, the module can handle only format V records.

```
|RELTRK=YES                                 |
```

This operand generates a logic module that can process with both physical and relative identifiers. If the operand is omitted, the module can process only with physical identifiers.

```
|SEPASMB=YES                                |
```

This operand must be included if the logic module is assembled separately. This causes a CATALR card with the module name (standard or user) to be punched ahead of the object deck.

Recommended Module Name List for DAMOD

Each name begins with a 3-character prefix (IJI) and consists of a 5-character field corresponding to the options permitted in the generation of the module.

DAMOD name = IJIabcde

a = F RECFORM=FIXUNB
  = B RECFORM=UNDEF (handles both UNDEF and FIXUNB)
  = S RECFORM=SPNUNB
  = V RECFORM=VARUNB

b = A AFTER=YES
  = Z AFTER is not specified

c = E IDLOC=YES and FEOVD=YES
  = I IDLOC=YES
  = R FEOVD=YES
  = Z neither is specified

d = H ERREXT=YES and RELTRK=YES
  = P ERREXT=YES
  = R RELTRK=YES
  = Z neither is specified

e = W HOLD=YES and RDONLY=YES
  = X HOLD=YES
  = Y RDONLY=YES
  = Z neither is specified

Subset/Superset DAMOD Names

The following chart shows the subsetting and supersetting allowed for DAMOD names. Five parameters allow supersetting. For example, the module IJIBAIZZ is a superset of the module with the name IJIFAZZZ. See Subset/Superset: (Module Names).

```
|                  + + + + + +              |
|          I J I  B A E H W W               |
|                 F Z I P X Y               |
|                 +   Z Z Z +               |
|                 S   + + + X               |
|                 V   E H W Z               |
|                     R R Y                 |
|                     Z Z Z                 |
|+Subsetting/supersetting permitted.        |
```

Notes: The module IJIBAEHW will cause assembly error IJY095 ENTRY TABLE OVERFLOW. The valid entry points for this module total more than 100, which is the maximum for assembler language. A user program can have only one DAMOD for fixed unblocked or undefined records and/or only one DAMOD for variable unblocked or spanned unblocked records; otherwise, duplicate name flagging occurs during assembly time.

Initialization - OPEN(R) Macro

After the file is defined with the DTFDA macro, the OPEN(R) macro can initialize the file, labels can be processed using the LBRET macro (see Label Processing), and the CNTRL, READ, WAITF, and WRITE macros can be used to process the file. The file can then be deactivated by the CLOSE(R) macro.

Op	Operand
for self-relocating programs	
OPENR	{filename1 / (r1)} [,{filename2 / (r2)}...,{filenamen / (rn)}]
for programs that are not self-relocating	
OPEN	{filename1 / (r1)} [,{filename2 / (r2)}...,{filenamen / (rn)}]

Note: To write the most efficient code (in a multiprogramming environment), we recommend that the self relocating form of OPEN be used. (See also Appendix G.)

When the operation OPEN is used, the symbolic address constants that OPEN generates from the parameter list are not self-relocating. When OPENR is specified, the symbolic address constants are self-relocating.

Self-relocating programs using IIOCS must use the OPENR macro instruction to activate all files, including printer-keyboard files. The OPENR macro, in addition to activating files for processing, relocates all address constants (except zero constants) within the DTF tables specified in the operand field(s) in register notation. If symbolic notation is used, the user needs to establish addressability through a base register.

If OPEN attempts to activate a logical IOCS file (DTF) whose device is unassigned, the job is terminated. If the device is assigned IGN, the OPEN(R) does not activate the file and turns ON the DTF byte 16, bit 2, to indicate the file is not activated.

The symbolic name of the file (DTF filename) is entered in the operand field. A maximum of 16 files may be opened with one OPEN (or OPENR) by entering the filenames as additional operands. Alternately, the user can load the address of the DTF filename in a register and specify the register using ordinary register notation. The high-order 8 bits of this register must contain zeros. A filename may be preloaded into any register, 0-15.

Whenever an input/output DASD file is opened and the user plans to process user-standard labels (UHL only), he must provide the information for checking or building the labels. If this information is obtained from another input file, that file must be opened, if necessary, ahead of the DASD or tape file. To do this, specify the input file ahead of the DASD file in the same OPEN or issue a separate OPEN preceding the OPEN for the file.

If an output file is created using the direct access method of processing, all volumes used must be mounted at the same time, and all the volumes are opened before the processing is begun.

For each volume, OPEN checks the standard VOL1 label and checks the extents specified in the extent cards:

1. The extents must not overlap.
2. Only type-1 extents can be used.
3. If user standard header labels are created, the first extent must be at least two tracks long.

OPEN checks all the labels in the VTOC to ensure that the created file does not destroy an existing unexpired file. OPEN then creates the standard label(s) for the file and writes the label(s) in the VTOC.

If the user wishes to create his own user labels (UHL) for the file, he must include the DTF entry LABADDR. OPEN reserves the first track of the first extent for these header labels and gives control to the user's label routine.

If the XTNTXIT entry is specified, OPEN stores the address of a 14-byte extent information area in register 1. (See DTFDA for the format of this area.) Then, OPEN gives control to the user's extent routine. The user can save this information for use in specifying record addresses.

After the user labels are written, the next volume is opened. When all the volumes are open, the file is ready for processing. If the DASD device is file protected, all extents specified in extent cards are available to the user.

Direct access input processing requires that all volumes containing the file be on-line and ready at the same time. All volumes used are opened before any processing can be done.

For each volume, OPEN checks the standard VOL1 label and then checks the file label(s) in the VTOC. OPEN checks some of the information specified in the extent cards for that volume. If LABADDR

is specified, OPEN makes the user standard header labels available one at a time for checking.

If the XTNTXIT entry is specified, OPEN stores the address of a 14-byte extent information area in register 1. (See DTFDA for the format of this area.) Control is then given to the user's extent routine. The user can save this information for use in specifying record addresses. Then, the next volume is opened. After all the volumes are open, the file is ready for processing. If the DASD device is file protected, all extents specified in extent cards are available for writing.

LBRET Macro

Name	Operation	Operand
[name]	LBRET	{1,2,3}

The LBRET macro is issued in user subroutines when the user has completed processing and wishes to return control to IOCS. LBRET applies to subroutines that write or check DASD user standard labels. The operand used depends on the function to be performed. See Label Processing.

CHECKING USER STANDARD DASD LABELS: IOCS passes the labels to the user one at a time until the maximum allowable number has been read and updated, or until the user signifies he wants no more. In his label routine, the user issues LBRET 3 if he wants IOCS to update (rewrite) the label read and pass him the next label. LBRET 2 is issued if he simply wants IOCS to read and pass him the next label. If an end-of-file record is read when LBRET 2 or LBRET 3 is issued, label checking is automatically ended. If the user wants to eliminate the checking of one or more remaining labels, he uses LBRET 1.

WRITING USER STANDARD DASD LABELS: The user builds the labels one at a time and uses LBRET to return to IOCS to write the labels. LBRET 2 is used if the user wants to have control return to him after IOCS writes the label. If, however, IOCS determines that the maximum number of labels are written, label processing is terminated. LBRET 1 is used if the user wishes to stop writing labels before the maximum number are written.

CHECKING DASD EXTENTS: When using the direct access method, the user can process his extent information. After each extent is processed, the user should issue a LBRET 2 to obtain the next extent. When extent processing is complete, control returns to IOCS by using the LBRET 1 macro.

READ Macro

Name	Operation	Operand
[name]	READ	{filename / (1)}, {KEY / ID}

****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

The READ and WAITF macro instructions transfer a record from DASD storage to an input area in main storage. The input area must be specified in the DTFDA entry IOAREA1.

The READ macro instruction is written in either of two forms depending on the type of reference used to search for the record. Both forms may be used for records in any one DTFDA-specified logical file if the logical file has keys.

This macro always requires two parameters. The first parameter specifies the name of the file from which the record is to be retrieved. This name is the same as that specified in the DTFDA header entry for the file and can be given either as a symbol or in register notation. The second parameter specifies the type of reference used for searching the records in the file.

If records in the file are undefined (RECFORM=UNDEF), DAM supplies the data length of each record in the designated register in the DTF entry RECSIZE.

Record Reference by Key

If the record reference is by key (control information in the key area of the DASD record), the second parameter in the READ

instruction must be the word KEY, and the DTFDA entry READKEY must be included in the file definition.

Whenever this method of reference is used, the problem program must supply the desired record key to IOCS before the READ instruction is issued. For this, the key must be stored in the key field (specified in the DTFDA entry KEYARG). When the READ instruction is executed, IOCS searches the previously specified track (stored in the 8-byte track-reference field) for the desired key. When a DASD record containing the specified key is found, the data area of the record is transferred to the data portion of the main-storage input area.

Only the specified track is searched unless the programmer requests that multiple tracks be searched on each READ instruction. A search of multiple tracks is specified by including the DTFDA entry SRCHM in the file definition. With this entry, the specified track and all following tracks are searched until the desired record is found or the end of the cylinder is reached. The search of multiple tracks continues through the cylinder even though part of the cylinder may be assigned to a different logical file.

Record Reference by ID

If the record reference is by ID (identifier in the count area of records), the second parameter in the READ instruction must be the letters ID, and the DTFDA entry READID must be included in the file definition.

Whenever this method of reference is used, the problem program must supply both the track information and the record number in the track-reference field. When the READ instruction is executed, IOCS searches the specified track for the particular record. When a record containing the specified ID is found, both the key area (if present and specified in DTFDA KEYLEN) and the data area of the record are transferred to key and data portions of the main-storage input area.

WRITE Macro

Name	Operation	Operand
[name]	WRITE	{filename | (1)}, {KEY | ID | AFTER[,EOF] | RZERO}

****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

The KEY, ID, or AFTER form of the instruction transfers an output record from main storage to DASD storage. The output area must be specified in the DTFDA entry IOAREA1, and the WAITF macro must be used.

The first parameter specifies the symbolic name of the file to which the record is transferred. This name is the same as the one specified in the DTFDA header entry for the file and can be given either as a symbol or in register notation. The second parameter specifies the type of reference that is used to find the proper location to write the output record. The third parameter is optional and applies only to the WRITE filename,AFTER form of the macro instruction.

The WRITE filename,AFTER,ECF form of the macro instruction writes an end-of-file record (a record with a length of zero) on a specified track after the last record on a track. The WRITE filename,RZERO resets the capacity record of a specified track to its maximum value and erases this track after record zero.

If records in the file are undefined (RECFORM=UNDEF), the programmer must determine the length of each record and load it in a register for IOCS use before he issues the WRITE instructions for that record. The register for this purpose must be specified in the DTFDA entry RECSIZE.

Record Reference by Key

If the DASD storage location for writing records is determined by the record key (control information in the key area of the DASD record), the word KEY is entered as the second parameter of the WRITE macro instruction. Also the DTFDA entry WRITEKY must be included in the file definition.

Whenever this method of reference is used, the problem program must supply the key of the desired record to IOCS before the WRITE instruction is issued. The key must be stored in the key field (specified by the DTFDA entry KEYARG). When the WRITE instruction is executed, IOCS searches the previously specified track (stored in the track-reference field) for the desired key. When a DASD record containing the specified key is found, the data in the main storage output area is transferred to the data area of the DASD record. This replaces the information previously recorded in the data area. The DASD count field of the original record controls the writing of the new record. If a record is shorter than the original record, it is padded with zeros. A record longer than the original record is written only to the extent of the area indicated in the count field on the track, and any excess bytes are lost. IOCS turns on the wrong-length-record bit in the error-status field if any short or long records occur.

Only the specified track is searched unless the programmer requests that multiple tracks be searched on each WRITE instruction. Searching multiple tracks is specified by including the DTFDA entry SRCHM in the file definition. In this case, the specified track and all following tracks are searched until the desired record is found or the end of the cylinder is reached. The search of multiple tracks continues through the cylinder even though part of the cylinder may be assigned to a different logical file.

Record Reference by ID

If the DASD storage location for writing records is determined by the record ID (identifier in the count area of records), the letters ID are entered as the second parameter of the WRITE instruction. Also, the DTFDA entry WRITEID must be included in the file definition.

Whenever this method of reference is used, the problem program must supply both the track information and the record number in the track-reference field. When the WRITE instruction is executed, IOCS searches the specified track for the particular record. When the DASD record containing the specified ID is found, the information in the main storage output area is transferred to the key area (if present and specified in DTFDA KEYLEN) and the data area of the DASD record. This replaces the key and data previously recorded. IOCS uses the count field of the original record to control the writing of the new record. If a record is shorter than the original record, it is padded with zeros. A record longer than the original record is written only to the extent of the area indicated in the count field on the track, and any excess bytes are lost. IOCS turns on the wrong-length-record bit in the error/status field if any long records occur. If an updated record is shorter than the original record, it is padded with binary zeros to the length of the original record. The wrong length record bit is not set on.

Record Reference: After

If a record is written following the last record previously written on a track (regardless of its key or ID), the second parameter of the WRITE instruction must be the specification AFTER. For this operation the DTFDA entry AFTER must be included in the file definition.

Whenever this method of reference is used for writing records, the problem program must supply the track information in the track-reference field. When WRITE is executed, IOCS examines the capacity record (record 0) on the specified track to determine the location and amount of space available for the record. If the remaining space is large enough, the information in the main-storage output area is transferred to the track in the location immediately following the last record. The count area, the key area (if present and specified by DTFDA KEYLEN), and the data area are written. IOCS then updates the capacity record. If the space remaining on the track is not large enough for the record, IOCS does not write the record and, instead, sets an indication in the user's error/status byte (specified by the DTFDA entry ERRBYTE).

Whenever a new file is built in an area of the disk pack or data cell containing outdated records, the capacity records must first be set up to reflect empty tracks by issuing the WRITE RZERO macro.

If one record is written on a track, and this record is near the maximum size, the capacity record will show a negative number of bytes remaining on the track. For example, on a 2311, a 3625 byte record can be written, but the value in the capacity record will be negative because the track tolerance will be subtracted from the number of bytes remaining on the track.

Record Reference: RZERO

The RZERO instruction resets the capacity record to reflect an empty track. The problem program must supply, in SEEKADR, the cylinder and track number of the track to be reinitialized. Any record number is valid but will be ignored. IOCS writes a new R0 with the maximum capacity of the track (3625 for an IBM 2311, 2000 for an IBM 2321, 7294 for an IBM 2314 or 2319) and erases the full track after R0.

This macro should be issued every time the problem program reuses a certain portion of a pack. It may be used as a utility function to initialize a limited number of tracks or cylinders.

WAITF Macro

Name	Operation	Operand
[name]	WAITF	{filename / (1)}

The WAITF macro instruction makes sure that the transfer of a record is complete. It requires only one parameter: the name of the file containing the record. The parameter can be specified either as a symbol or in register notation.

This instruction must be issued before the problem program attempts to process an input record or build another output record for the file concerned. The program does not regain control until the data transfer is complete. Thus, the WAITF macro instruction must be issued after any READ or WRITE instruction for a file, and before the succeeding READ or WRITE instruction for the same file. The WAITF macro makes error/status information, if any, available to the problem program in the field specified by DTFDA ERRBYTE.

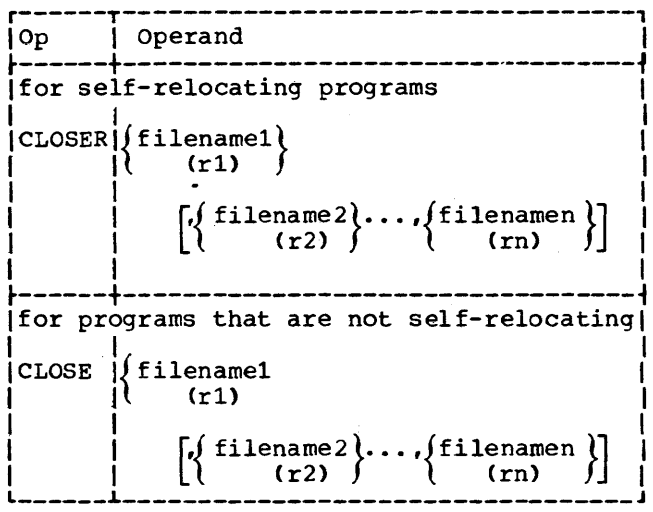
CNTRL Macro

Name	Operation	Operand
[name]	CNTRL	{filename / (1)},code

The CNTRL (control) macro instruction can begin DASD access movement, (SEEK) or restore a data cell strip (RESTR), for the next READ or WRITE for a file. It requires two parameters.

The first parameter specifies the name of the file, which is the same name as that specified in the DTFDA header entry for the file, and can be given either as a symbol or in register notation. The second parameter must be the word SEEK (for 2311, 2314, 2319, and 2321) or RESTR (for 2321 only). The seek address must be provided in the field with the symbolic name given in the DTFDA entry SEEKADR before issuing the CNTRL macro.

Completion - CLOSE(R) Macro



Note: To write the most efficient code (in a multiprogramming environment), we recommend that the self relocating form of OPEN be used. (See also Appendix G.)

The CLOSE macro instruction deactivates any file that was previously opened in any input/output unit in the system. If trailer labels are specified, they are written on output, and checked on input. A file may be closed at any time by issuing this macro instruction. No further commands can be issued for the file unless it is reopened.

When the operation CLOSE is used, the symbolic address constants that CLOSE generates from the parameter list are not self-relocating. When CLOSER is specified, the symbolic address constants are self relocating. This latter form of CLOSER is therefore recommended.

The symbolic name of the logical file (assigned in the DTF header entry) to be closed is entered in the operand field. A maximum of 16 files may be closed by one instruction by entering additional filename parameters as operands. Alternately, the user can load the address of the filename in a register and specify the register using ordinary register notation. The high-order 8 bits of this register must be zeros. The address of the filename may be prelcaded into any register, 0-15.

Note: If you use register notation, we recommend using only registers 2-12. This will make your programs more compatible with the Operating System (OS).

Indexed Sequential Access Method (ISAM)

The Indexed Sequential Access Method (ISAM) permits DASD records to be processed in either random or sequential order. For random processing, the user supplies the key (control information) of the desired record to ISAM and issues a READ or WRITE macro instruction to transfer the specified record. For sequential processing, the user specifies the first record to be processed and then issues GET or PUT macro instructions until all desired sequential records are processed. The successive records are made available in sequential order by key. Variations in macro instructions permit:

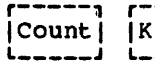
- A logical file of records to be loaded onto DASD (created).
- Individual records to be read from, added to, or updated in the file.

Whenever the indexed sequential system of processing is used, the logical file and main-storage areas allotted to the file must be defined by the declarative macro DTFIS (Define The File for Indexed Sequential system). For the detail parameter entries for this definition, see Declarative Macro Instructions.

> Note: DOS does not support a null ISAM file. If an attempt is made to access a null file, an error indication (X'10') is posted in filename.C.

Record Types

When an ISAM file is originally organized, it is loaded onto the volume(s) from presorted input records. These records must be sorted by key and all records in the file must contain key areas:



All keys must be the same length, and the length must be specified in the DTFIS entry KEYLEN.

The logical records must be fixed length, and the length must be specified in the DTFIS entry RECSIZE. Logical records may be either blocked (two or more logical records in one physical record) or unblocked (one logical record per one physical record). This must be specified in the DTFIS entry RECFORM. When blocked records are specified, the key of the highest record (last) in the block is the key for the block and, therefore, ISAM stores it in the key area of the record. The number of records in a block must be specified in the DTFIS entry NRECDS.

Storage Areas

Records in one logical file are transferred to, or from, one or more I/O areas in main storage. The areas must always be large enough to contain the key area and a block of records, or a single record if unblocked records are specified. Also, space must be allowed for the count area when a file is loaded, or when records are added to a file. For the functions of adding or retrieving records, the I/O area must also provide space for a sequence-link field used with overflow records (see Addition of Records and Overflow Areas). When an overflow record is brought into the I/O area, the sequence-link field should not be altered by the problem programmer. The I/O area requirements are illustrated schematically in Figure 35 and described in detail in the DTFIS entries IOAREAL, IOAREAR, IOAREAS, and IOAREA2.

Records may be processed directly in the I/O area or in a work area for either random or sequential retrieval. If the records are processed in the I/O area, a register must be specified in the DTFIS entry IOREG. This register is used for indexing, and points to the beginning of each record.

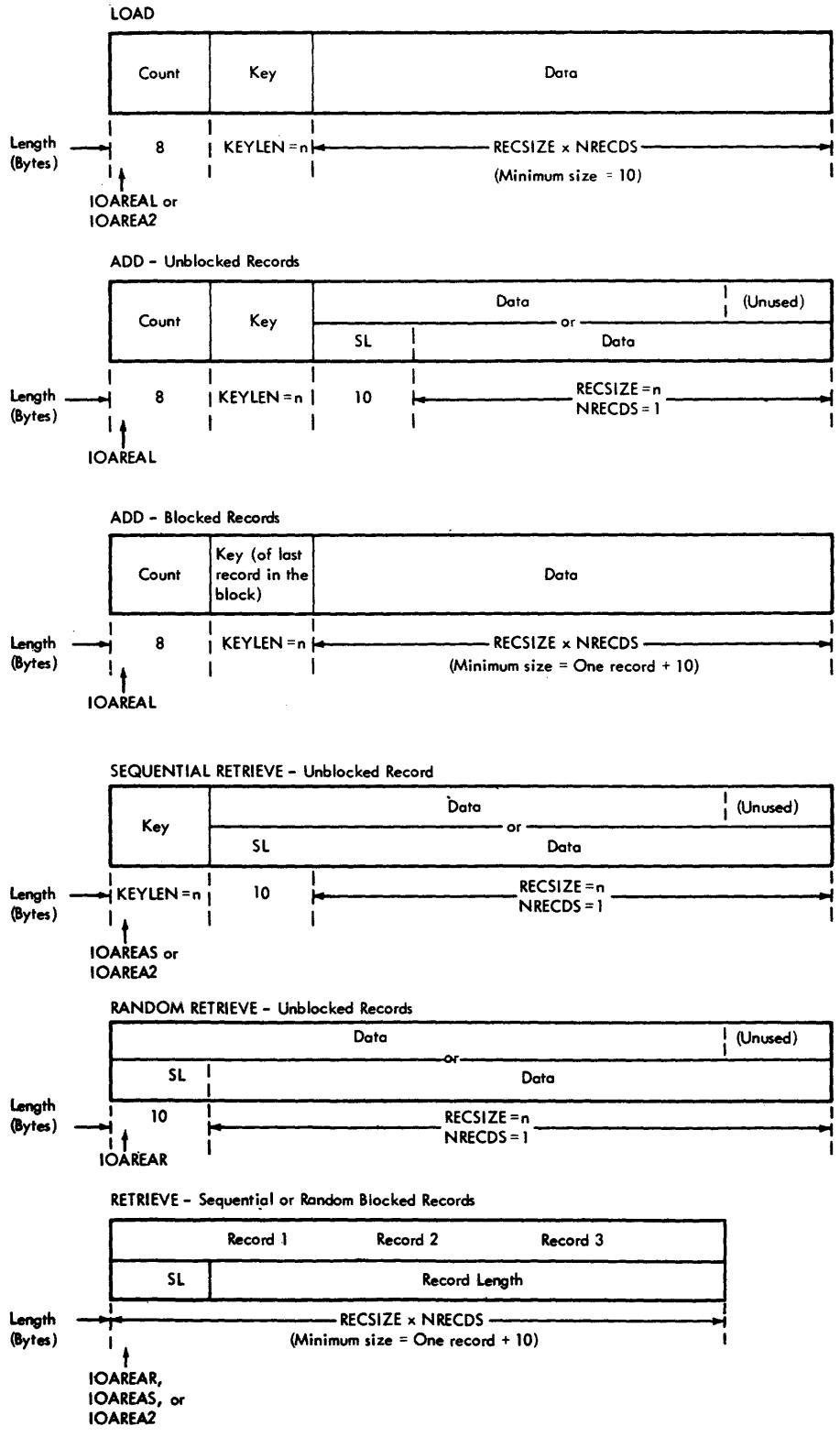


Figure 35. Schematic of I/O Area in Main Storage, for ISAM

If the records are processed in a workarea, the DTFIS entry WORKL, WORKR, or WORKS must be specified. ISAM moves each individual input record from the I/O area to the workarea where it is available to the problem program for processing. Similarly, on output ISAM moves the completed record from the workarea to the I/O area where it is available for transfer to DASD storage. Whenever a workarea is used, a register is not required.

Organization of Records on DASD

When a logical file of presorted records is loaded onto DASD, ISAM organizes the file in a way that allows the user to access any record. For any type of processing, the entire ISAM file must be on line. If an ISAM file is assigned ignore, no I/O processing can be done for that file.

Reference can be made to records at random throughout the logical file, or to a series of records in the file in their (collated) presorted sequence. The organization also provides for additions to the file at a later time, while still maintaining both the random and sequential reference capabilities.

ISAM loads the records (one after the other) into a specified area of the DASD volume. This area is called the prime area of the logical file on DASD. Both the starting and ending limits of this area are specified by the user in job control extent statements. At least one record must be written at load time if an ISAM file is referenced.

Indexes

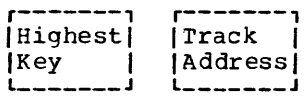
As ISAM loads a file of records sorted by key, it builds a set of indexes for the file. The indexes are utilized for both random and sequential reference to records as follows:

- They permit rapid access to individual records for random processing.
- They supply the records in key order during sequential processing.

Two or possibly three indexes are built, depending on the user's specifications. Both a track index and a cylinder index are always constructed. A master index is also constructed if the DTFIS entry MSTIND is included in the file definition.

Once a file is loaded and the related indexes are built, the ISAM routines search for specified records by referring to the indexes. When a particular record (specified by key) is requested for processing, ISAM searches the master index (if used), then the cylinder index, then the track index, and finally the individual track. Each index narrows the search by pointing to the portion of the next-lower index whose range includes the specified key. Because of the high speed and efficiency of the direct access devices, a master index should be established only for exceptionally large files, for which the cylinder index occupies several tracks (possibly five or more). That is, it is generally faster to search only the cylinder index (followed by the track index) when the cylinder index occupies four or less tracks.

The indexes are made up of a series of entries, each of which includes the address of a track and the highest key on that track or cylinder.



Each entry is a separate record composed of both a key area and a data area. The key area contains the highest key on the track or cylinder, and its length is the same as that specified for logical data records (in the DTFIS entry KEYLEN). The data area of each index is ten bytes long, it contains track information including the track address.

The indexes are terminated by a dummy entry that contains a key of all one bits. To avoid any possibility of errors, the user should not use a key of all one bits for any of his records.

Track Index

The track index is the lowest-level index for the logical file. A separate track index is built for each cylinder used by the file, and contains index entries for that cylinder only. Each track index is located on the cylinder that it is indexing. It always begins on track zero, and it may extend over more than one track.

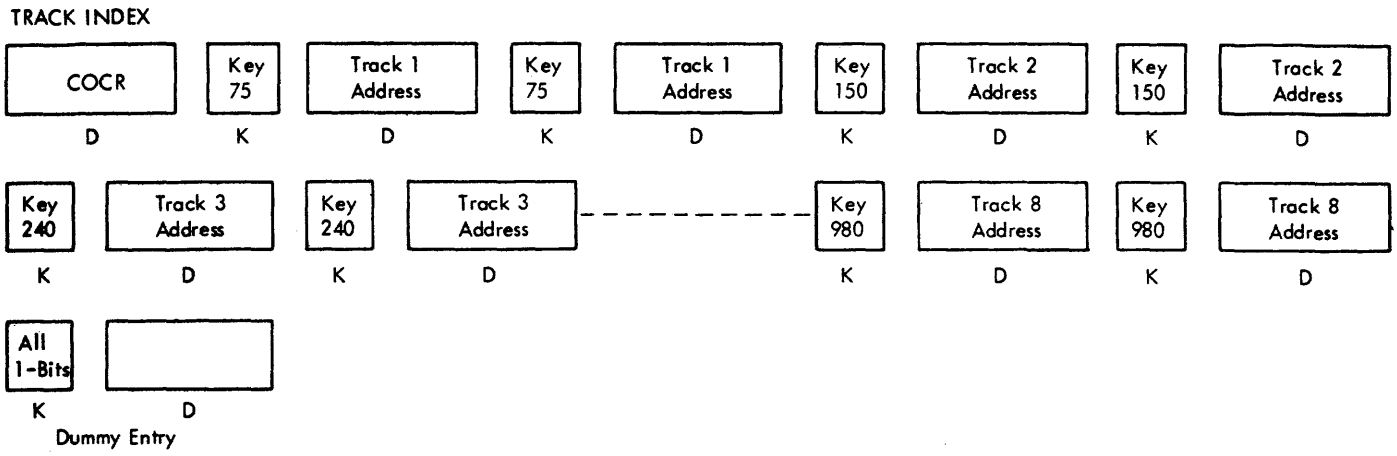


Figure 36. Schematic Example of a Track Index

When the track indexes are originally constructed, they contain two similar entries (normal and overflow) for each track used on the cylinder. For example, if the prime area of the logical file uses eight tracks on a cylinder, the track index might contain the entries shown in Figure 36. The use of two index records for each track is required because of overflow records that occur if more records are inserted in the file at a later time (see Addition of Records and Overflow Areas). When overflow records for a track exist, the second (overflow) index record contains the key of the highest record in the overflow chain and the address of the lowest record in the overflow chain for the track. The dummy entry indicates the end of the track index. Any following records are logical-file data records.

Cylinder Index

The cylinder index is an intermediate level index for the logical file. It contains an index entry for each cylinder occupied by the file. This index is built in the location specified by the user in a job control extent. The user may change the upper extent limit; however, no validity check is performed by the ISMOD. Thus, it is the user's responsibility to make sure the change is correct.

The cylinder index may not be built on one of the cylinders that contains prime data records for a file. Also, it should not be built on a cylinder that contains overflow records as this could prevent future expansion of the overflow area. The cylinder index should be on a separate cylinder, or it may be on a separate volume that is on-line whenever the logical file is processed.

The cylinder index may be located on one or more successive cylinders. Whenever the index is continued from one cylinder to another, the last index entry on the first cylinder contains a linkage field that points to the first track of the next cylinder. A cylinder index may not be continued from one volume to another, however.

This index contains one entry for each cylinder occupied by the data file. The key area contains the highest key associated with the cylinder, and the data area contains the address of the track index for that cylinder. For example, if a file requires nine cylinders, the cylinder index might contain the entries shown in Figure 37. The dummy entry indicates the end of the cylinder index.

References to a cylinder index also apply to the 2321. The bar-position address of a data cell corresponds to the cylinder of a disk drive in ISAM.

Master Index

The master index is the highest-level index for a logical file built by DOS. This index is optional, and it is built only if it is specified by the DTFIS entry MSTIND. A master index is built in the location specified by a job control extent statement. Like the cylinder index, it may be located on the same volume with the logical-file records or on a different volume that is on-line whenever the records are processed.

The master index must immediately precede the cylinder index on a volume, and it may be located on one or more successive cylinders. Whenever it is continued from one cylinder to another, the last index entry on the first cylinder contains a linkage field that points to the first track of the next cylinder. A master index may not be continued from one volume to another.

The master index contains an entry for each track of the cylinder index. The key area contains the highest key on the cylinder index track, and the data area contains the address of that track. For example, if a master index is located on track x and a cylinder index is located on tracks x+1 through x+20, the master index might contain the entries shown in Figure 38. The dummy entry indicates the end of the master index.

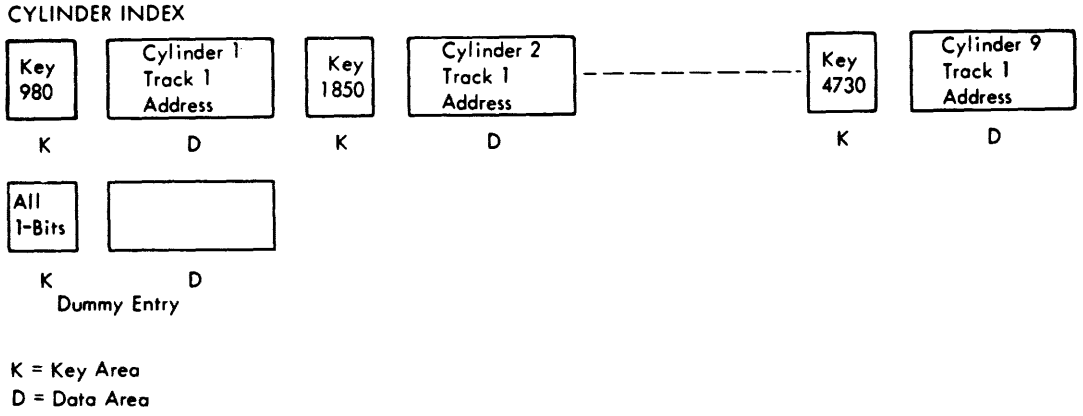


Figure 37. Schematic Example of a Cylinder Index

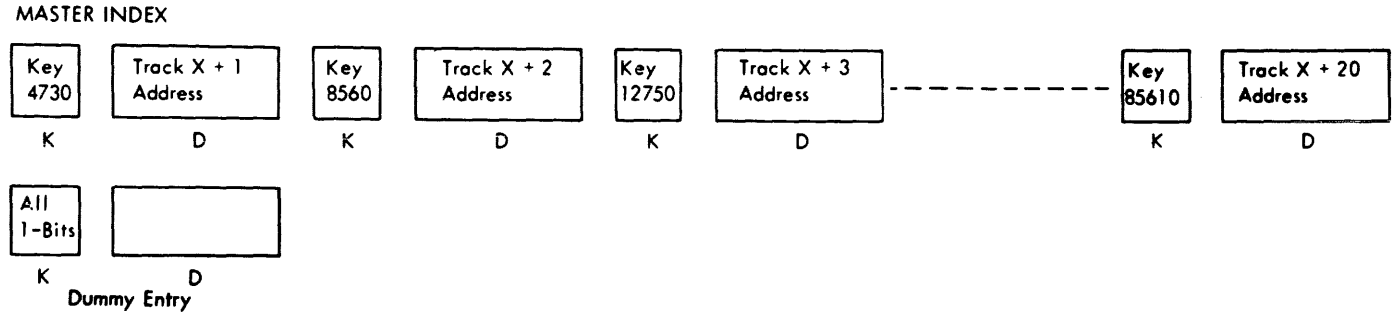


Figure 38. Schematic Example of a Master Index

ADDITION OF RECORDS AND OVERFLOW AREAS

After a logical file is organized on DASD, it may subsequently become necessary to add records to the file. These records may contain keys that are above the highest key presently in the file and, thus, constitute an extension of the file. Or, these records may contain keys that fall between keys already in the file and therefore require insertion in the proper sequence in the organized file.

If all records to be added have keys that are higher than the highest key in the organized file, the upper limit of the prime area of the file can be adjusted (if necessary) by the specification in a job control extent statement. The new records can then be added by presorting them and loading them into the file. No overflow area is required, and the file is merely extended further on the volume. However, new records can be batched with the normal additions and added to the end of the file.

However, if records must be inserted among those already organized, an overflow area is required. ISAM uses the overflow area to permit the insertion of records without necessitating a complete reorganization of the established file. The fast random and sequential retrieval of records is maintained by inserting references to the overflow chains in the track indexes, and by using a chaining technique for the overflow records. For chaining, a sequence-link field is prefixed to the user's data record in the overflow area. The sequence-link field contains the address of the record in the overflow area that has the next-higher key. Thus, a chain of sequential records can be followed when searching for a particular record. The sequence-link field of the highest record in the chain indicates the end of the chain. All records in the overflow area are unblocked, regardless of the specification (in DTFIS RECFORM) for the data records in the logical file.

To add a record by insertion, ISAM searches the indexes first to determine on which track the record must be inserted. After the proper track index is located, the point of insertion can then be determined. The keys of the last records on the tracks in the originally organized file determine the track where an inserted record belongs. A record is always inserted on the track where:

1. The last key is higher than the insertion, and

2. The last key of the preceding track is lower than the insertion.

For example, assume tracks 2 and 3 are organized with the record keys shown in Figure 39. Then, records with keys such as 151, 175, 199, 215, and 239 are inserted on track 3 (or in the related overflow chain that has developed). Any key lower than 150 is added to either track 1 or track 2; any key higher than 240 belongs to track 4 or above. The track indexes always retain the highest key of each track as it was originally organized.

After the proper track is determined, ISAM searches the individual records on the track or overflow area (if necessary) to find where the record belongs in key order. This results in either of two conditions:

1. The record falls between two records presently on the track. In this case, ISAM adds the record by inserting it in the proper sequence and shifting each succeeding record one record location higher on the track, until the last record is forced off the track. ISAM transfers this last record to the overflow area, and prefixes the record (data area) with a sequence-link field. The first time a record is inserted on a track, the sequence-link of the overflow record indicates that this is the highest record associated with the track. Thereafter, the sequence-link field of each overflow record points to the next-higher record for that track.

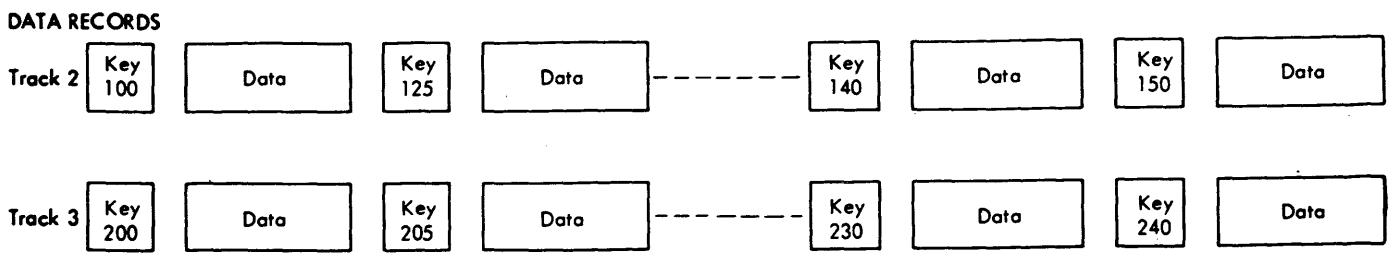


Figure 39. Example of Data Records as Originally Organized on Tracks 2 and 3

ISAM also updates the track index to reflect this change. The first index record for the track has the key field changed to indicate the new last-record located on the track. The second index record for the track has the track address (in the data area) changed to point to the address of the lowest overflow record. If a record with a key 105 is added to a file organized as shown in Figure 39, and if the overflow area is located on track 9, the track index records contain the information shown in Figure 40.

INDEX ENTRIES FOR ONE TRACK

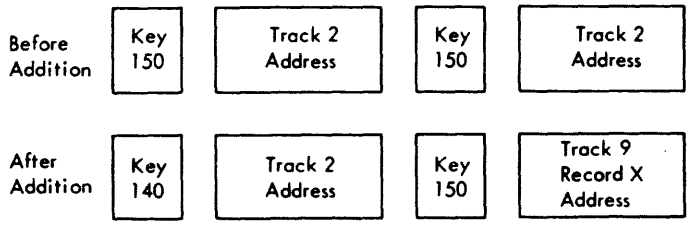


Figure 40. Example of Track Index Entries Before and After Addition of a Record on Track 2

2. The record falls between the last record presently on the track and the last record originally on the track. Thus, it belongs in the overflow area. ISAM writes the record in the overflow area following the last record previously written. ISAM searches through the chain of records associated with the corresponding track for this record and identifies the sequential position the record should take. Then the sequence-link fields of the new record, and of the record preceding it by sequential key, are adjusted to point to the proper records. If records 150, 140, and 130 are already in the overflow area and record 135 is to be added, for example, the sequence-link fields of records 130 and 135 must be adjusted (Figure 41).

RECORD	SEQUENCE-LINK FIELD	
	Before Addition	After Addition
130	140	135
135 (New Record)	—	140

Figure 41. Example of Sequence Link Fields Adjusted for Addition of a Record 135

Overflow-Area Option: The location of the overflow area(s) for a logical file may be specified by the user. The overflow areas may be built by one of three methods:

1. Overflow areas for records may be located on each cylinder within the prime area. In this case, the user must specify the number of tracks that are reserved for overflow on each cylinder occupied by the file. The overflow records that occur within a particular cylinder are written in the cylinder overflow area for that cylinder.

   The number of tracks to be reserved for each cylinder overflow area must be specified in the DTFIS entry CYLOFL when a file of records is loaded and when records are added to an organized file.

2. An independent overflow area may be specified for storing all overflow records for the logical file. In this case, a job control extent statement must be included when the program is executed to specify the area of the volume to be used for the overflow area. This area may be on the same volume with the data records, or on a different volume that is on-line. However, it must be contained within one volume, and the device must be the same kind as that containing the prime data area. If an independent overflow area is not specified, you can add it only during a LOAD, ADD, or ADDRTR job. You cannot add it during a RETRVE job.

3. Both cylinder overflow areas (method 1) and an independent overflow area (method 2) may be used. In this case, overflow records are placed first in the cylinder overflow areas within the data file. When any cylinder overflow area becomes filled, the additional overflow records from that cylinder are written in the independent overflow area. The specifications required for both methods 1 and 2 must be included for this combined method of handling overflows.

All records placed in the overflow area are in the unblocked format and have a sequence-link field prefixed to each record. There must always be one prime data track available for a DASD EOF record when additions are made to the last track in the prime data area containing records. For additional information about overflow areas, see the WRITE Macro later in this section of the manual (ISAM).

EXAMPLE OF AN ORGANIZED FILE

Figure 42 shows schematically a simplified example of a file organized on DASD by the Indexed Sequential Access Method. This figure illustrates an organized file for an IBM 2311 DASD with the last two tracks on each cylinder used for the overflow area. The same file would have similar characteristics if it was created on another IBM DASD type. The assumptions made and the items to be noted are:

1. The track index occupies part of the first track, and prime data records occupy the rest of the track. This is called a shared track.

2. The data records occupy part of track 0 and all of tracks 1-7. Tracks 8 and 9 are used for overflow records in this cylinder.

3. The master index is located on track X on a different cylinder. The cylinder index is located on tracks X+1 through X+20.

4. A dummy entry signals the end of each index.

5. The file was originally organized with records as follows:

Track	Records
0	5-75
1	100-150
2	.
.	.
.	.
7	900-980

6. The track index originally had two similar entries for each track. It now shows that overflow records have occurred for tracks 1 and 7.

7. Records 150, 140, and 130 were forced off the track by insertions on the track. Record 135 was added directly in the overflow area.

8. A sequence-link field (SL) was prefixed to each overflow record. The records for track 1 can be searched in sequential order by following the SL fields:

Record	Sequence-Link Field (SL)
130	SL points to record with key 135.
135	SL points to record with key 140.
140	SL points to record with key 150.
150	End of search. (Key 150 was the highest key on track 2 when the file was loaded.)

9. When the file was loaded, the last record on cylinder 1 was record 980; on cylinder 2, record 1850; and on cylinder 9, record 4730. This is reflected in the cylinder index. The first entry in the master index is the last entry of the first track of the cylinder index.

10. When cylinder overflow areas are used, the first record (record 0) in the track index for a cylinder is the Cylinder Overflow Control Record (COCR). It contains the address of the last overflow record on the cylinder and the number of tracks remaining in the cylinder overflow area. When the number of remaining tracks is zero, overflow records are written in the independent area. The format of record zero data field is as follows: hhrbbtxx

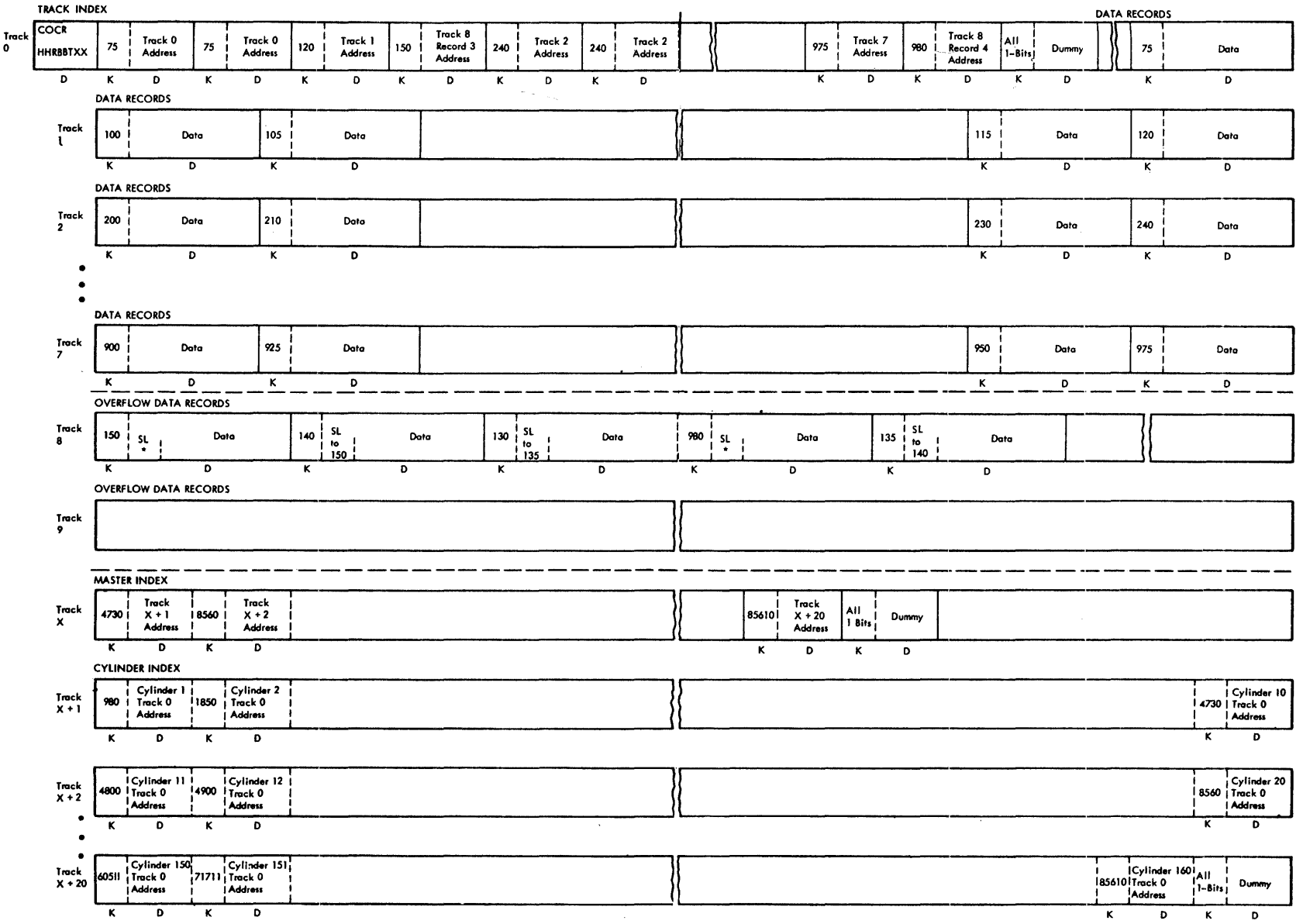
    hh - last cylinder overflow track containing the records.

    r - last overflow record on the track.

    bb - the number of bytes remaining on the track (for fixed-length records this is binary zeros).

    t - the number of remaining tracks available in the cylinder overflow area.

    xx - reserved (with binary zeros).



K = Key Area
D = Data Area
SL = Sequence Link *SL indicates the end of the overflow chain.
COCR = Cylinder Overflow Control Record (Contained in R0)

Figure 42. Schematic of a File on 2311 DASD Organized by ISAM

Indexed Sequential Macros

Before an indexed sequential file can be processed, it must be defined to IOCS by the DTFIS declarative macro. After defining the file, the user can operate with the file using imperative macros. The file must be initialized, processed, and deactivated.

Status or Condition Code Indication

The DTF macro instruction provides a 1-byte field where all status or condition codes are placed after execution of each macro instruction. The problem program must take responsibility for checking these condition codes after each imperative macro, except OPEN and CLOSE, by referencing the field called filenameC. Figure 43 shows the format of the filenameC byte. Filename should be the same as that specified in the DTF header entry for the file. ISAM provides addressability for filenameC by returning the address of the DTF table, or the address of the ERREXT parameter list if ERREXT is specified and an error occurs. The address is returned in register 1 after each ISAM imperative macro is executed. The FilenameC byte is in the format shown in Figure 43.

The problem program will also be responsible for checking byte 16, bit 7 of the DTF for a blocksize compatibility error when adding to, or extending a file. If the blocksize of the problem program is not equal to the blocksize of the previously built file, this bit will be posted.

ADD, RETRVE, and ADDRTR

Bit	Cause	Explanation
0	DASD error	Any uncorrectable DASD error has occurred (except wrong length record).
1	Wrong length record	A wrong length record has been detected during an I/O operation.
2	End of file	The EOF condition has been encountered during execution of the sequential retrieval function.
3	No record found	The record to be retrieved has not been found in the data file. This applies to Random (RANSEQ) and to SETL in SEQNTL (RANSEQ) when KEY is specified, or after GKEY.
4	Illegal ID specified	The ID specified to the SETL in SEQNTL (RANSEQ) is outside the prime data file limits.
5	Duplicate record	The record to be added to the file has a duplicate record key of another record in the file.
6	Overflow area full	An overflow area in a cylinder is full, and no independent overflow area has been specified, or an independent overflow area is full, and the addition cannot be made. The user should assign an independent overflow area or extend the limit.
7	Overflow	The record being processed in one of the retrieval functions (RANDOM/SEQNTL) is an overflow record.

Figure 43. FilenameC--Status or Condition Code Byte -- ADD, RETRVE, and ADDRTR (Part 1 of 2)

Bit	Cause	Explanation
0	DASD error	Any uncorrectable DASD error has occurred (except wrong length record).
1	Wrong length record	A wrong length record has been detected during an I/O operation.
2	Prime data area full	The next to the last track of the prime data area has been filled during the load or extension of the data file. The problem programmer should issue the ENDFL macro, then do a load extend on the file with new extents given.
3	Cylinder Index area full	The Cylinder Index area is not large enough to contain all the entries needed to index each cylinder specified for the prime data area. This condition can occur during the execution of the SETFL. The user must extend the upper limit of the cylinder index by using a new extent card.
4	Master Index full	The Master Index area is not large enough to contain all the entries needed to index each track of the Cylinder Index. This condition can occur during SETFL. The user must extend the upper limit, if he is creating the file, by using an extent card. Or, he must reorganize the data file and assign a larger area.
5	Duplicate record	The record being loaded is a duplicate of the previous record.
6	Sequence check	The record being loaded is not in the sequential order required for loading.
7	Prime data area overflow	There is not enough space in the prime data area to write an EOF record. This condition can occur during the execution of the ENDFL macro.

Figure 43. FilenameC--Status or Condition Code Byte -- LOAD (Part 2 of 2)

ERET (Error Return) Macro:

Name	Operation	Operand
[name]	ERET	{SKIP / IGNORE / RETRY}

At the completion of each imperative macro instruction, filenameC should be checked by the problem program error routine. This macro instruction enables a problem program specifying the ERREXT operand to return to IOCS and specify an action to be taken for each error condition.

The IGNORE (or SKIP) operand passes control back to the module to ignore the error and continue processing with the block in error. The RETRY operand causes the module to retry the operation that caused the error.

When ERREXT is not specified and a data transfer error condition is detected in filenameC, control is passed back to ISAM only by issuing another macro instruction. However, no special action is taken by ISAM to correct or check this error. Also, no information is passed to the problem program except for what is contained in filenameC.

If ERREXT is specified, register 1 contains the address of an 18-byte parameter list. The content of this parameter list is found in Figure 44. When ERREXT is specified, nondata transfer error conditions are indicated in the DTF data transfer bit (byte 2, bit 2) and the problem program error processing routine can return to ISAM via the ERET macro. The ERET IGNORE or ERET SKIP macro returns to ISAM to ignore the error condition and process the record. The ERET RETRY macro returns to ISAM to make another attempt at reading or writing the record.

> Note: The ERREXT coding does not handle nonrecoverable errors that are posted in FilenameC. Examples of nonrecoverable type errors are: no record found (may also be caused by hardware errors), prime data area full, master index full, etc. The supervisor may recover from a No Record Found condition if byte 3 bit 5 of the DTF is set. However, a recovery would then be initiated also for an unrecoverable NRF condition.

If ERREXT is specified, the problem program error processing routine should determine whether or not data was transferred. This can be done by checking the data transfer bit (byte 2, bit 2) in the DTF. If the data transfer bit is ON,

13 (with RDONLY) should be saved before use and restored afterward.

If HOLD=YES is specified, you must issue the ERET macro to return to ISAM during an ERREXT to free any held tracks.

Bytes	Bits	Contents
0-3	---	DTF address
4-7	---	Main storage address of the record in error.
8-15	---	DASD address of the error (mbbcchhr) where m is the extent sequence number and r is a record number which can be inaccurate if a read error occurred during a read of the highest level index. For more information see Track Index.
16		Record identification:
	0	Data record
	1	Track index record
	2	Cylinder index record
	3	Master index record
		Type of operation:
	4	Not used
	5	Not used
	6	Read
	7	Write
17	---	Command code of failing CCW

Note: If the error occurred on an index record, the user should not IGNORE this record unless it is first checked for accuracy. If the record was read inaccurately, the user should RETRY to read the record.

Figure 44. ERREXT Parameter List

INDEXED SEQUENTIAL FILE (DTFIS)

The DTFIS detail entries that apply to a file when records are processed by the Indexed Sequential Access Method are explained here and summarized in Figure 47. A DTFIS header entry and a series of detail entries describe the file.

The symbolic name of the file, filename, is entered in the name field. DTFIS is entered in the operation field.

CYLOFL=n

This entry must be included if cylinder overflow areas are reserved for a logical file. Do not include this entry if no overflow areas are reserved.

When a file is loaded or when records are added, this operand is required to reserve the areas for cylinder overflow (optional for retrieval operations). It specifies the number n of tracks to be reserved on each cylinder. The maximum number of tracks that can be reserved on each cylinder is 8 for 2311 and 18 for 2314, 2319, or 2321.

If an independent overflow area is specified (by an extent card) along with the CYLOFL entry, overflow records are written in the independent overflow area after a cylinder overflow area becomes filled.

DEVICE={2311|2314|2321}

This entry specifies the unit that contains the prime data area or overflow areas for the logical file.

Note: Specify 2314 for 2319.

DSKXTNT=n

This entry must be included to specify the maximum number n of extents for this file. The number must include all the data area extents if more than one DASD area is used for the data records, and all the index area and independent overflow area extents that are specified by extent statements. Thus the minimum number specified by this entry is 2: one extent for one prime data area, and one for a cylinder index. Each area assigned to an ISAM data file is considered an extent.

Note: Master and cylinder indexes are treated as one area. When there is one master index extent, one cylinder index extent, and one prime data area extent, DSKXTNT=2.

ERREXT=YES

This operand enables a problem program error routine checking filenameC to return to ISMOD with the ERET macro. Also, it enables unrecoverable I/O errors occurring before a data transfer takes place to be indicated to the problem program.

If HOLD=YES and ERREXT=YES, you must issue the ERET macro to return to the ISAM module to free any held tracks.

HINDEX={2311|2314|2321}

This entry specifies the unit containing the highest index.

Note: Specify 2314 for 2319.

HOLD=YES

This operand provides for the track hold option. Track hold prevents two or more programs from updating the same record at the same time. For ISAM, the hold applies to both data records and index records.

Because track hold cannot be performed on a LOAD file, HOLD=YES cannot be specified when IOROUT=LOAD. Also, if HOLD=YES and ERREXT=YES, the problem program must issue the ERET macro to return to the ISAM module to free any held tracks. If the HOLD parameter is omitted, the track hold function is not performed.

For further information, see the DASD Track Protection Macros section.

INDAREA=name

This operand specifies the symbolic name of the area assigned to the cylinder index. If specified, all or part of the cylinder index resides in main storage thereby increasing throughput. If this operand is included, INDSIZE must be included.

If the area assigned to INDAREA is large enough for all the index entries to be read into main storage at one time and the index skip feature (INDSKIP) is not specified, no presorting of records need be done. If the area assigned to INDAREA is not large enough, the records processed should be presorted to fully utilize the resident cylinder index.

INDSKIP=YES

When cylinder index entries reside in main storage, this operand specifies the index skip feature. This feature allows ISFMS to skip any index entries preceding those needed to process a given key.

This feature may only be specified with the INDAREA and INDSIZE operands and increases throughput only when

- The records are presorted.
- The allocated main storage is insufficient for storing all of the cylinder index.
- A large segment(s) of the file is not referenced.

If the index skip operand is omitted, the cylinder indexes are processed sequentially.

INDSIZE=n

This operand specifies the number of bytes reserved at the INDAREA parameter name for the cylinder index entries. The minimum number of bytes (n) must be:

(m+3)(keylength+6)

where m is the number of entries to be read into main storage at a time, 3 is the number of dummy entries, and 6 is an abbreviated pointer to the cylinder. If m is set equal to the number of prime data cylinders+1, all the cylinder index is read into main storage at one time.

The resident index facility is suppressed if this operand is omitted or the minimum requirement is not met at assembly time, or an unrecoverable read error is encountered while reading the index.

IOAREAL=name

This operand must be included when a file is created (loaded) or when records are added to an organized file. It specifies the symbolic name of the output area used for loading or adding records to the file. The specified name must be the same as the name used in the DS instruction that reserves the area of main storage. The ISAM routines construct the contents of this area and transfer records to DASD.

This main-storage output area must be large enough to contain the count area, key area, and data area of records. Furthermore, the data-area portion must provide enough space for the sequence-link field of overflow records whenever records are added to a file (Figure 45).

If IOAREAL is increased to permit the reading and writing of more than one physical record on DASD at a time, the IOSIZE operand must be included when records are added to the file. In this case, IOAREAL must be at least the equivalent number of bytes in IOSIZE.

When simultaneously building two ISAM files using two DTFs, do not use a common IOAREAL. Also, do not use a common area for IOAREAL, R, and S in multiple DTFs.

IOAREAR=name

This operand must be included whenever records are processed in random order. It specifies the symbolic name of the input/output area for random retrieval (and updating). The specified name must be the same as that used in the DS instruction that reserves this area of main storage.

The I/O area must be large enough to contain the data area for records. Furthermore, the data-area portion must provide enough space for the sequence-link field of overflow records (Figure 46).

FUNCTION	OUTPUT AREA REQUIREMENTS (IN BYTES)			
	Count	Key	Sequence Link	Data
Load Unblocked Records	8	Key Length	—	Record Length
Load Blocked Records	8	Key Length	—	Record Length x Blocking Factor
Add Unblocked Records	8	Key Length	10	Record Length
Add Blocked Records	8	Key Length	— OR*	Record Length x Blocking Factor
	8	Key Length	10	Record Length
* Whichever Is Larger				

Figure 45. Output Area Requirements for Loading or Adding Records to a File by ISAM

FUNCTION	I/O AREA REQUIREMENTS (IN BYTES)			
	Count	Key	Sequence Link	Data
Retrieve Unblocked Records	—	Key Length for sequential unblocked records	10	Record Length
Retrieve Blocked Records	—	—	—	Record Length (including keys) x Blocking Factor
	—	—	10 OR*	Record Length
* Whichever is Larger				

Figure 46. I/O Area Requirements for Random or Sequential Retrieval by ISAM

IOAREAS=name

This operand must be included whenever records are processed in sequential order by key. It specifies the symbolic name of the input/output area used for sequential retrieval (and updating). The specified name must be the same as that used in the DS instruction that reserves this area of main storage.

This main-storage I/O area must be large enough to contain the key and data areas of unblocked records and the data area for blocked records. Furthermore, the data-area portion must provide enough space for the sequence-link field of overflow records (Figure 46).

IOAREA2=name

This operand enables overlapping of I/O with indexed sequential processing for either the load (creation) or sequential retrieval functions. The parameter [name] represents the symbolic name of an I/O area used when loading or sequentially retrieving records. The I/O area must be at least the length of the area specified by either the IOAREAL parameter for the load function or the IOAREAS parameter for the sequential retrieval function. If the operand is omitted, one I/O area is assumed. If TYPEFLE=RANSEQ, only one I/O area is supported.

IOREG=(r)

This operand must be included whenever records are retrieved and processed directly in the I/O area. It specifies the register that ISAM uses to indicate which individual record is available for processing. ISAM puts the address of the current record in the designated register (2-12) each time a READ, WRITE, GET, or PUT is executed.

IOROUT={LOAD|ADD|RETRVE|ADDRTR}

This entry must be included to specify the type of function to be performed. One of the following specifications is entered after the = sign:

LOAD To build a logical file on DASD or to extend a file beyond the highest record presently in an organized file.

ADD To insert new records into an organized file.

RETRVE To retrieve records from a file for either random or sequential processing and/or updating.

ADDRTR To both insert new records into a file (ADD) and retrieve records for processing and/or updating (RTR).

IOSIZE=n

This operand specifies the number of bytes in the main-storage area assigned to the ISFMS add function using IOAREAL. Its size (n) is a decimal number and specified by

m(8+keylength+blocksize+32)+24

where: m is the maximum number of physical records that can be read into main storage at one time; 8 is the count field; 32 and 24 are ISAM CCWs; and n must be at least equal to

[8+keylength+blocksize+32+10]+24

The 10 bytes account for a needed sequence link field for unblocked records or short blocks (see Figure 45).

If omitted, or if the minimum requirement is not met, no increase in throughput is realized when adding records to a file.

KEYARG=name

This operand must be included for random READ/WRITE operations and sequential retrieval initiated by key. It specifies the symbolic name of the main-storage key field in which the user must supply the record key to ISAM.

KEYLEN=n

This operand must be included to specify the number, n, of bytes in the record key. All keys must be the same length.

```
|KEYLOC=n                                |
```

This operand must always be specified if an add, load, or retrieve function is performed and FIXBLK is specified in DTFIS RECFORM. This entry must always be included for fixed blocked records. It supplies ISAM with the high-order position of the key field within the data record. That is, if the key is recorded in positions 21-25 of each record in the file, this entry should be 21.

ISAM uses this specification to locate (by key) a specified record within a block. The key area of a DASD record contains the key of the highest record in the block. To search for any other records, ISAM locates the proper block and then examines the key field within each record in the block.

```
|MODNAME=name                            |
```

This operand may be used to specify the name of the logic module used with the DTF table to process the file. If the logic module is assembled with the program, the MODNAME in the DTF macro instruction must specify the same name as the ISMOD macro instruction. If this entry is omitted, standard names are generated for calling the logic module. If two DTF macro instructions call for different functions that can be handled by a single module, only one module is called.

```
|MSTIND=YES                              |
```

This operand is included whenever a master index is used for a file. It is required when a file is loaded (to instruct ISAM to build the index) and when records are added to or retrieved from a file with a master index.

ISAM always builds a track index and a cylinder index, but the master index is optional. The master index, if used, is the highest level index, and includes an index record for each track of the cylinder index. Thus, it points to the cylinder index on a search for a particular record (see Indexes: Master Index). The location of the master index is specified by a job control extent statement.

```
|NRECDS=n                                |
```

This operand specifies the number, n, of logical records in a block (called the blocking factor). If RECFORM=FIXUNB, n is assumed to be 1.

```
|RDONLY=YES                              |
```

This operand is specified if the DTF is used with a read-only module. Each time a read only module is entered, register 13 must contain the address of a 72 byte doubleword aligned save area. Each DTF should have its own uniquely defined save area. Register 13 must contain the address of the save area associated with that DTF each time an imperative macro (except OPEN(R), LBRET, SETL, or SETFL) is issued using a particular DTF. The fact that the save areas are unique, or different for each task makes the module reentrant (that is, capable of being used concurrently by several tasks). For more information see Shared Modules and Files.

If an ERROPT or WLRERR routine issues I/O macro instructions using the same read only module that caused control to pass to either error routine, the problem program must provide another save area. One save area is used for the normal I/O, and the second for I/O in the ERROPT or WLRERR routine. Before returning to the module that entered the ERROPT routine, register 13 must be set to the save area address originally specified for that DTF. If the operand is omitted, the module generated is not reenterable and no save area need be established.

```
|RECFORM={FIXUNB|FIXBLK}                 |
```

This operand specifies the type of records in the logical file. All logical records in the file must be fixed length. However, they may be either blocked or unblocked. One or the other of these specifications must be entered after the = sign:

FIXUNB For unblocked records.

FIXBLK For blocked records. If FIXBLK is specified, the key of the highest record in the block becomes the key for the block and must be recorded in the key area.

The specification that is included when the logical file is loaded into DASD storage must also be included whenever the file is processed.

Records in the overflow area(s) are always unblocked (see Addition of Records and Overflow Areas), but this has no affect on the entry. RECFORM refers to records in the prime data area only.

```
|RECSIZE=n                                |
```

This operand must be included to specify the number, n, of characters in a logical record. This is the length of the data area of each individual record. All logical records must be the same size. Also, this operand should specify the same number for additions and retrieval as indicated when the file was created.

```
|SEPASMB=YES                              |
```

This operand must be included if the DTF is assembled separately. This causes a CATALR card with the filename to be punched ahead of the object deck and defines the filename as an entry point in the assembly.

```
|TYPEFLE={RANDOM|SEQNTL|RANSEQ}           |
```

This operand must be included when a retrieval function is performed. It specifies the type(s) of processing performed by the problem program for the file. One of the following specifications is entered after the = sign:

RANDOM For random processing. Records are retrieved from the file in random order specified by key.

SEQNTL For sequential processing. The problem program specifies the first record retrieved, and thereafter ISAM retrieves records in sequential order by key. The first record is specified by key, ID, or the beginning of the logical file (see SETL Macro).

RANSEQ For both random and sequential processing. Only one I/O area is supported.

TYPEFLE is not required for loading or adding functions.

```
|VERIFY=YES                               |
```

This operand is included if the user wants to check the parity of disk records after they are written. VERIFY is always assumed when 2321 records are written. If this operand is omitted, any records written on 2311, 2314, or 2319 are not verified.

```
|WORKL=name                               |
```

This operand must be included whenever a file is created (loaded) or records are added to an organized file. It specifies the symbolic name of the workarea in which the user must supply the data records to ISAM for loading or adding to the file. The specified name must be the same as the name used in the DS instruction that reserves this area of main storage.

This workarea must provide space for one logical data record when a file is created (for blocked records, data; for unblocked records, key and data).

The original contents of WORKL are changed due to record shifting in the ADD function.

```
|WORKR=name                               |
```

When records are processed in random order, this entry must be included if the individual records are processed in a workarea rather than the I/O area. It specifies the symbolic name of the workarea. This name must be the same as the name used in the DS instruction that reserves this area of main storage. This area must provide space for one logical record (data area). When this entry is included and a READ or WRITE instruction is executed, ISAM moves the individual record to, or from, this area.

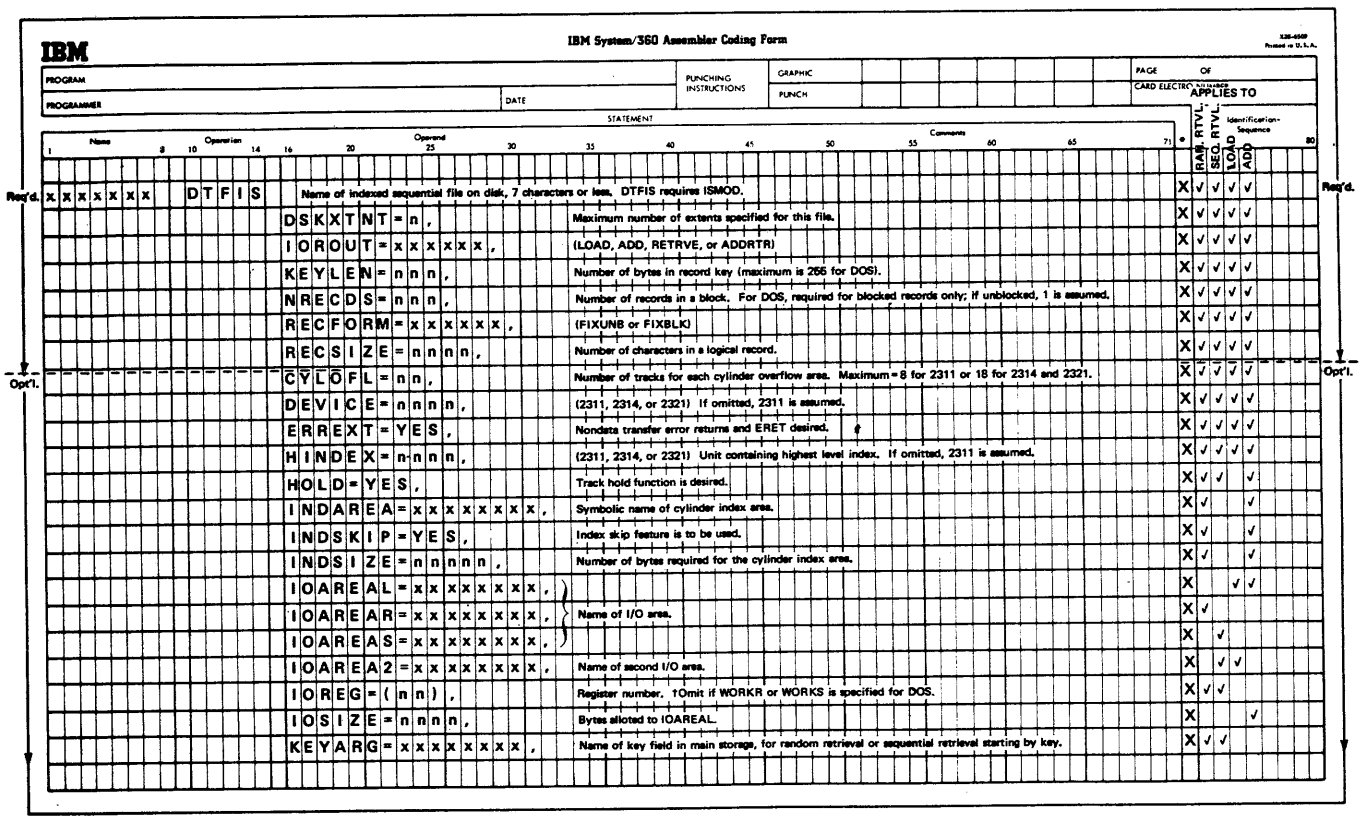
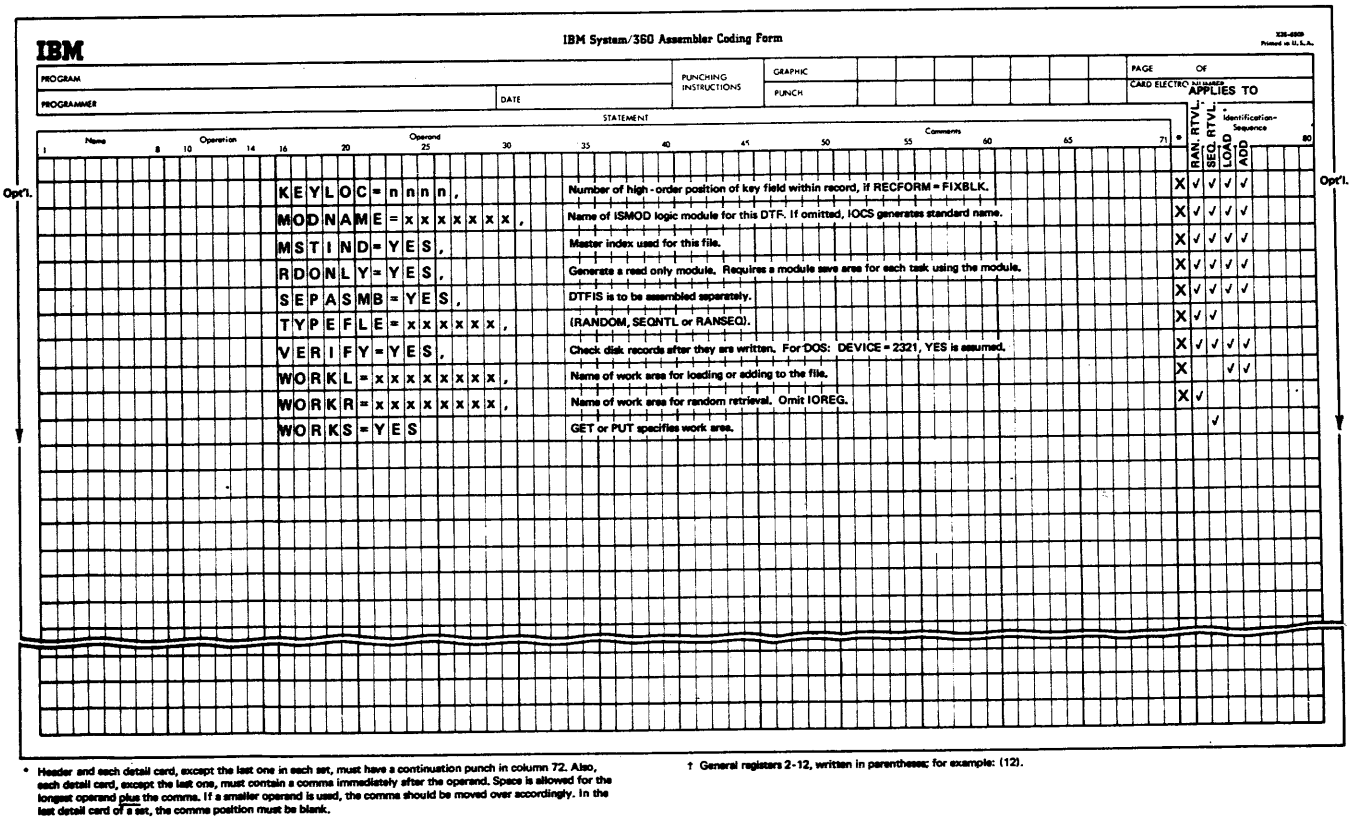


Figure 47. DTFIS Macro (Part 1 of 2)



* Header and each detail card, except the last one in each set, must have a continuation punch in column 72. Also, each detail card, except the last one, must contain a comma immediately after the operand. Space is allowed for the longest operand plus the comma. If a smaller operand is used, the comma should be moved over accordingly. In the last detail card of a set, the comma position must be blank.

† General registers 2-12, written in parentheses; for example: (12).

Figure 47. DTFIS Macro (Part 2 of 2)

```
WORKS=YES
```

When records are processed in sequential order, this entry must be included if the individual records are processed in workareas rather than the I/O area. Each GET and PUT instruction must specify the symbolic name of the workarea to or from which ISAM is to move the record. When processing unblocked records, the area must be large enough for one record (data area) and the record key (key area). For blocked records, the area must be large enough for one logical record (data area) only.

The ISAM workarea requirements are as follows:

	Unblocked Records	Blocked Records
Load	(KL + DL) or 10*	DL or 10*
Add	(KL + DL) or 10*	DL or (KL + 10)*
Random Retrieve	DL	DL
Sequential Retrieve	KL + DL	DL

Where:
K=Key, D=Data, L=Length

*Whichever is greater.

INDEXED SEQUENTIAL MODULE (ISMOD)

A set of ISMOD parameters is included for each logic module necessary to support each DTFIS macro in a particular problem program. The logic modules are described by a ISMOD header entry and a series of operand entries. The header entry contains ISMOD in the operation field, and may contain a user supplied name in the name field. The parameters are explained in this section and shown in Figure 48.

Note: If an ISMOD logic module precedes an assembler language USING statement or follows the problem program, registers 2-12 remain unrestricted even at assembly time. However, if the ISMOD logic module(s) lies within the problem program, the problem programmer should issue the same USING statement [which was issued before the logic module(s)] directly following the logic module(s). This action is necessary because the ISMOD logic module uses registers 1, 2, and 3 as base registers, and the ISMOD CORDATA logic module uses registers 1, 2, 3, and 5 as base registers. Each time either module is assembled, these registers are dropped.

```
CORINDX=YES
```

Include this operand to generate a logic module that can process DTFIS files (add or random retrieve functions) with or without the cylinder index entries resident in main storage. If omitted, the module generated cannot process the resident cylinder index entries.

If an unrecoverable I/O error occurs while reading indexes into main storage, the program does not take advantage of the resident cylinder index entries.

```
CORDATA=YES
```

Include this operand if the module is to add records to files with the IOSIZE DTFIS operand. If this operand is included, the module cannot add records to DTFIS files unless the IOSIZE operand is properly specified. If you omit the CORDATA=YES operand, you will not have an increase in throughput when adding records to a file.

```
ERREXT=YES
```

Include this operand if the ERET macro is to be used with this module or if nondata transfer error conditions are returned in filenameC.

If HOLD=YES and ERREXT=YES, the problem program must issue the ERET macro to return to the ISAM module to free any held tracks.

```
|HOLD=YES                                  |
```

This operand provides for the track hold option. Track hold prevents two or more programs from updating the same record at the same time. For ISAM, the hold applies to both data records and index records.

Because track hold cannot be performed on a LOAD file, HOLD=YES cannot be specified when IOROUT=LOAD. Also, if HOLD=YES and ERREXT=YES, the problem program must issue the ERET macro to return to the ISAM module to free any held tracks. If the HOLD parameter is omitted, the track hold function is not performed.

For further information, see the DASD Track Protection Macros section.

```
|IOAREA2=YES                               |
```

Include this operand if a second I/O area is to be used. This module can process DTFs with one or two I/O areas specified. If TYPEFLE=RANSEQ, only one I/O area is supported.

```
|IOROUT={LOAD|ADD|RETRVE|ADDRTR}          |
```

This operand specifies the type of logic module required to perform a given function. The entries follow.

LOAD

Generates a logic module for creating a file.

ADD

Generates a logic module for adding new records into an existent file.

RETRVE

Generates a logic module to retrieve (randomly/sequentially) records from an organized file.

ADDRTR

Generates a logic module that combines the features of the ADD and RETRVE modules. This module also processes any file in which only ADD or RETRVE is specified in the IOROUT operand statement of the DTF for the file, and in which the TYPEFLE entry contains the corresponding parameter (or a subset of it).

```
|RDONLY=YES                                |
```

This operand generates a read only module. RDONLY=YES must be specified in the DTF. For the programming requirements of this operand, see the DTF RDONLY operand.

```
|RECFORM={FIXUNB|FIXBLK|BOTH}              |
```

This operand generates a detailed logic module that creates, adds to, or processes an unblocked (FIXUNB) or blocked (FIXBLK) data file. If BOTH is specified, a module is generated to process both unblocked and blocked files, and the DTF entry for the file may specify either FIXUNB or FIXBLK in the RECFORM operand statement. The RECFORM operand is required only when IOROUT specifies ADD or ADDRTR. If IOROUT specifies LOAD or RETRVE, a module that handles fixed-length blocked and unblocked files is generated, and the operand is not required.

```
|SEPASMB=YES                               |
```

This operand must be included if the logic module is assembled separately. This causes a CATALR card with the module name (standard or user) to be punched ahead of the object deck.

```
|TYPEFLE={RANDOM|SEQNTL|RANSEQ}            |
```

This operand is required when IOROUT specifies RETRVE or ADDRTR. RANDOM generates a logic module that includes only random retrieval capabilities. SEQNTL generates a logic module that includes only sequential retrieval capabilities. RANSEQ generates a logic module that includes random and sequential capabilities. It also processes any file in which the TYPEFLE parameter statement of the DTF specifies either RANDOM or SEQNTL. If TYPEFLE=RANSEQ, only one I/O area is supported.

When all operands are omitted, the ISMOD logic module can process files where IOROUT=RETRVE, TYPEFLE=RANSEQ, CORINDX, CORDATA, HOLD, and RDONLY are not specified. In this event, the module name assumed is IJHZRBZZ.

Name	Operation	Operand	Remarks
[Modname]	ISMOD		
		ERREXT=YES	Required if nondata transfer error conditions or ERET are desired.
		CORDATA=YES	Required to add records using the DTF IOSIZE operand.
		CORINDX=YES	Required to add or retrieve records with the cylinder index entries in main storage.
		HOLD=YES	Specifies the track hold option.
		IOAREA2=YES	Required if two I/O areas are to be used.
		IOROUT= {LOAD / ADD / RETRVE / ADDRTR}	Specifies function to be performed.
		RDONLY=YES	Required if a read only module is to be generated.
		RECFORM= {FIXUNB / FIXBLK / BOTH}	Describes file. Required if IOROUT specifies ADD or ADDRTR. If IOROUT specifies LOAD or RETRVE, BOTH is assumed.
		SEPASMB=YES	If the module is assembled separately.
		TYPEFLE= {RANDOM / SEQNTL / RANSEQ}	Required if IOROUT specifies RETRVE or ADDRTR.

Figure 48. ISMOD Macro

Recommended Module Name List for ISMOD

Each name begins with a 3-character prefix (IJH) and consists of a 5-character field corresponding to the options permitted in the generation of the module.

ISMOD name = IJHabcde

a = A RECFORM=BOTH, IOROUT=ADD or ADDRTR
  = B RECFORM=FIXBLK, IOROUT=ADD or ADDRTR
  = U RECFORM=FIXUNB, IOROUT=ADD or ADDRTR
  = Z RECFORM is not specified. (IOROUT=LOAD or RETRVE)

b = A IOROUT=ADDRTR
  = I IOROUT=ADD
  = L IOROUT=LOAD
  = R IOROUT=RETRVE

c = B TYPEFLE=RANSEQ
  = G IOAREA2=YES, TYPEFLE=SEQNTL or IOROUT=LOAD
  = R TYPEFLE=RANDOM
  = S TYPEFLE=SEQNTL
  = Z neither is specified (IOROUT=LOAD or ADD)

d = B CORINDX=YES and HOLD=YES
  = C CORINDX=YES
  = O HOLD=YES
  = Z neither is specified

e = F CORDATA=YES, ERREXT=YES, RDONLY=YES
  = G CORDATA=YES and ERREXT=YES
  = O CORDATA=YES and RDONLY=YES
  = P CORDATA=YES
  = S ERREXT=YES and RDONLY=YES
  = T ERREXT=YES
  = Y RDONLY=YES
  = Z neither is specified

Subset/Superset ISMOD Names

The following chart shows the subsetting and supersetting allowed for ISMOD names. Five parameters allow supersetting. For example, the module IJHBABZZ is a superset of the module IJHBASZZ. See Subset/Superset: (Module Names).

			+	+	+	+
I	J	H	A	A	B	B
			B	I	R	O
			Z	+	+	+
			+	A	B	C
			A	R	S	Z
			U	*	+	
			Z	L	G	
					S	
					+	
					G	
					Z	

+ Subsetting/supersetting permitted.
* No subsetting/supersetting permitted.

If two or more modules with the same entry point are included, the linkage editor message 2143I (invalid duplication of entry point label) is generated. (Occasionally these entry points are not obvious when using the preceding chart but the module can perform the indicated functions.) This message can usually be suppressed by including a superset module. However, modules with and without prime data in main storage or modules with TYPEFLE=RANDOM and IOAREA2=YES cannot be combined. Therefore, the user should take either of the following actions:

1. Specify prime data in core for each ADD type DTF in his program. In this case, superset modules are generated.

2. Specify the MODNAME operand in the DTF, and include an ISMOD of that name. The DTF then generates only the specified module.

Initialization - OPEN(R) Macro

Op	Operand
for self-relocating programs	
OPENR	{filename1 | (r1)} [{filename2 | (r2)},...,{filenamen | (rn)}]
for programs that are not self-relocating	
OPEN	{filename1 | (r1)} [,{filename2 | (r2)},...,{filenamen | (rn)}]

Note: To write the most efficient code (in a multiprogramming environment), we recommend that the self relocating form of OPEN be used. (See also Appendix G.)

When the OPEN macro instruction is used, the symbolic address constants that OPEN generates from the parameter list are not self-relocating. When OPENR is specified, the symbolic address constants are self-relocating. The latter form (OPENR) is therefore recommended.

Self-relocating programs using LIOCS must use the OPENR macro instruction to activate all files, including printer-keyboard files. The OPENR macro, in addition to activating files for processing, relocates all address constants (except zero constants) within the DTF tables specified in the operand field(s) in register notation. If symbolic notation is used, the user must establish addressability through a base register.

If OPEN attempts to activate a logical IOCS file (DTF) whose device is unassigned, the job is terminated. If the device is assigned IGN, the OPEN(R) does not activate the file and turns on the DTF byte 16, bit 2, to indicate the file is not activated.

The symbolic name of the file (DTF filename) is entered in the operand field. A maximum of 16 files may be opened with one OPEN (or OPENR) by entering the filenames as additional operands. Alternately, the user can load the address of the DTF filename in a register and specify the register using ordinary register notation. The high-order 8 bits of this register must contain zeros. A filename may be preloaded into any register, 0-15.

Note: If you use register notation, we recommend using only registers 2-12. This will make your programs more compatible with the Operating Systems (OS).

Whenever an input/output DASD is opened, the user must provide the information for checking or building the labels. (See Label Processing.)

When a file is created or extended by using index sequential output processing, those volumes of the file to be written on are opened as output files. If the file consists of more than one volume, all the volumes must be on line and ready when the file is first opened.

For each volume, OPEN checks the standard VOL1 label and performs extensive checks on the extents specified in the extent cards for that volume. The extents must meet the following conditions:

1. All prime data extents must be continuous.
2. The master and cylinder index extents must be continuous and on the same unit.
3. No extents must overlap,
4. Only type 1, 2, or 4 extents are valid.
5. The extent sequence numbers must be in the following order:

   0 for master index, when present.

   1 for cylinder index.

   2, 3, 4,... for the prime data and independent overflow tracks.

   The extent cards for the independent overflow tracks can be placed either before or after all the extent cards for the prime data extents.

OPEN checks all the labels in the VTOC to ensure that the file to be created does not destroy an existing file. Any expired labels are deleted from the VTOC. After the VTOC check, OPEN creates the standard labels for the file and writes the labels in the VTOC. If the DASD device is file protected, all extents specified in the extent cards are available for writing. All volumes containing an indexed sequential file must be on-line and ready when the file is first opened.

For each volume, OPEN checks the extents specified in the extent cards for that volume (for example, checks that the data extents are continuous). OPEN also checks the standard VOL1 label and then goes to the VTOC to check the file label(s) before opening the next volume. After all the volumes are opened, the file is ready for processing. If the DASD device is file protected, all extents specified in extent cards are available to the user.

ISAM MACROS TO LOAD OR EXTEND A FILE

The functions of originally loading a file of presorted records onto DASD, or extending the file by adding new presorted records beyond the previous high record, are the same. Both are considered a LOAD operation (specified by the DTFIS entry IOROUT), and use the same macro instructions in the problem program. However, the type field in the DLAB card must specify ISC for load creation and ISE for load extension.

The areas of the volumes used for the file are specified by job control extent cards. The areas are:

The prime area where the data records are written.

A cylinder index area where the user wants ISAM to build the cylinder index.

A master index area if a master index is to be built (specified by the DTFIS entry MSTIND).

During a load operation, ISAM builds the track, cylinder, and master indexes.

Three different macro instructions are always required in the problem program to load original or extension records into the logical file on DASD.

SETFL Macro to Load or Extend a File

Name	Operation	Operand
[name]	SETFL	{filename / (0)}

The SETFL (set file load mode) macro instruction causes ISAM to set up the file so that the load or extension function can be performed. When loading a file, SETFL preformats the last track of each track index. When extending a file, SETFL preformats only the last track of the last track index plus each new track index for the extension of the file. This allows prime data on a shared track to be referenced even though no track indexes exist on the shared track. The name of the file loaded is the only parameter required in this instruction and is the same as that specified in the DTFIS header entry for the file. It can be specified as a symbol or in register notation. Register notation is necessary to allow use of the macro in a self relocating program. This macro must be issued whenever the file is loaded or extended.

WRITE Macro to Load or Extend a File

Name	Operation	Operand
[name]	WRITE	{filename / (1)} ,NEWKEY

****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

When a WRITE macro instruction with the parameter NEWKEY is issued in the problem program between a SETFL instruction and an ENDFL instruction (the third macro required for loading), ISAM loads a record onto DASD.

A WRITE macro instruction requires two parameters. The first parameter is the name of the file specified in the DTFIS header entry. The filename can be specified as a symbol or in register notation. The second parameter must be the word NEWKEY.

Before issuing the WRITE instruction, the problem program must store the key and data portions of the record in a workarea (specified by DTFIS WORKL). The ISAM routines construct the I/O area (see Figure 35) by moving the data record to the data area, moving the key to the key area, and building the count area. When the I/O area is filled, ISAM transfers the records to DASD storage and then constructs the count area for the next record. The WAITF macro should not be used when loading or extending an ISAM file.

Before records are transferred, ISAM performs both a sequence check and a duplicate-record check. This ensures that the records are in order by key.

After each WRITE is issued, ISAM makes the ID of the record or block available to the problem program. The ID is located in an 8-byte field labeled filenameH, which cannot exceed 7 characters. For example, if the filename in the DTFIS header entry is PAYRD, the ID field is addressed by PAYRDH. The ID of any selected records can be punched or printed for later use by referencing this field. Filename H is required if the user plans to retrieve records in sequential order starting with the ID of a particular record (see SETL Macro).

As records are loaded or added on DASD, ISAM uses the I/O areas to write:

- The new track address each time a track is filled.
- Two track index records (one prime data, one overflow) each time a track is filled.
- A cylinder index record each time a cylinder is filled.
- A master index record (if DTFIS MSTIND is specified) each time a cylinder index is filled.

ENDFL Macro to Load or Extend a File

Name	Operation	Operand
[name]	ENDFL	{filename / (0)}

The ENDFL (end file load mode) macro instruction ends the mode initiated by the SETFL macro. The name of the file loaded is the only parameter required in this instruction and is the same as the name specified in the DTFIS header entry for the file. The filename can be specified either as a symbol or in register notation. Register notation is necessary to allow use of the macro in a self relocating program.

The ENDFL macro performs an operation similar to close for a blocked file. It writes the last block of data records, if necessary, and then writes an end-of-file record after the last data record. Also, it writes any index entries that are needed followed by dummy index entries for the unused portion of the prime data extent.

ISAM MACROS FOR ADDING RECORDS

After a file is organized on DASD, new records can be added. Each record is inserted in the proper place sequentially by key. To provide this function specify ADD or ADDRTR in the DTFIS entry IOROUT.

The file may contain either blocked or unblocked records, as specified by the DTFIS entry RECFORM. When the file contains blocked records, the user must provide ISAM with the location of the key field provided through the DTFIS entry KEYLOC. The inserted records are written one record at a time. The records must contain a key field in the same location as the records already in the file. Whenever the addition of records follows sequential retrieval (ADDRTR), the macro instruction ESETL must be issued before a record is added. Two macro instructions (WRITE and WAITF) are used in the problem program to add records to a file.

WRITE Macro for Adding Records

Name	Operation	Operand
[name]	WRITE	{filename / (1)} ,NEWKEY

****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

The operand filename is the same name that is contained in the DTFIS header entry. The name can be specified either as a symbol or in register notation.

Before the WRITE macro is issued for unblocked records, the program must store the record (key and data) to be added into a workarea specified in the DTFIS entry WORKL. For blocked records, the program must store only the data since the key is assumed to be a part of the data. Before any records transfer, ISAM checks for duplicate record keys. If none are found, ISAM inserts the record into the file.

To insert a record into a file, ISAM performs an index search at the highest level. This search determines if the key of the record to be inserted is lower or higher than the key of the last record in the file. If it is lower, the record can be inserted, and searching of the master index (if available), the cylinder index, and the track index determines the appropriate location to insert the record.

To add an entry to an unblocked file, an equal/high search is performed in the prime data area of the track. When such a condition occurs, the record is read from the track and placed in main storage (in the I/O area). The two records are then compared to check for duplicate records. If a duplication is found, this information is posted to the user in the DTF table at filenameC. If none is found, the appropriate record (in the user's work area) is written directly to the track. The record (just displaced from the track) in the I/O area is moved by ISAM to the user's work area, and the next record on the track is read into the I/O area. Then, the record in the workarea is written on the track. Succeeding records are shifted until the last record on the track is set up as an overflow record.

If the add I/O area (IOAREAL) is increased to permit the reading or writing of more than one record on DASD at a time, an equal/high search is performed in the prime data area of the track. When such a condition occurs, as many records as can fit into the I/O area (specified in the DTFIS operand IOAREAL) are read from the track and placed in the I/O area. The added record is compared with existing records in the I/O area. If a duplicate key is found, the condition is posted to the user in the DTF table filenameC. If no duplicate is found, the records are shifted in main storage, leaving the record with the highest key remaining in the user's workarea. The other records are rewritten directly onto the track. Any remaining record(s) on the track are then read into the I/O area. This process continues until the last record on the track is set up as an overflow record. The last record is then written into the appropriate overflow area, and the track index entries are updated. This area becomes the cylinder overflow area, if CYLOFL is specified for this file and the area is not filled.

If the cylinder overflow area is filled, or if only an independent area is specified by a job control extent card, the end record is transferred to the independent overflow area. If an independent overflow area was not specified (or is filled) and the cylinder area is also filled, no room is available to store the overflow record. ISAM posts this condition in the DTF table at filenameC. In all cases, ISAM determines if room is available before any records are written.

For an entry to a blocked file, the workarea (WORKL) is required in the DTFIS entries. Each added record must contain a key field in the same location as the records already in the file. The user must specify the high-order position of the key field (relative to the leftmost position of the logical record). Use the DTFIS entry KEYLOC for this purpose.

When a WRITE macro is issued in the problem program, ISAM first locates the correct track by referring to the necessary master (if available), cylinder, and track indexes. Then, a search on the key areas of the DASD records on the track is made to locate the desired block of records. The block of records is read into the I/O area. If IOREAL is included for reading and writing more than one record on DASD at a

time, several blocks may be read into the I/O area.

ISAM then examines the key field within each logical record to find the exact position in which to insert the new record and then checks for any duplicate records. If a duplicate key exists the condition is posted in filenameC. If the key of the record inserted (contained in the workarea WORKL) is low, it is exchanged with the record presently in the block. This procedure continues with each succeeding record in the block until the last record is moved into the workarea. ISAM then updates the key area of the DASD record to reflect the highest key in the block. If IOAREAL was included, succeeding blocks in the I/O area are also updated. The block (or blocks) is then written back onto DASD. The remaining blocks on the track are similarly processed until the last logical record on the track is moved into the workarea. This record (set up as an overflow record with the proper sequence-links) is then moved to the overflow area. The indexes are updated and ISAM returns to the problem program for the next record to be added. If the overflow area is filled, the information is posted in filenameC.

If the proper track for a record is an overflow track (determined by the track index), ISAM searches the overflow chain and checks for any duplication. If no duplication is found, ISAM writes the record (preceded by a sequence-link field in the data area of the DASD record) and adjusts the appropriate linkages to maintain sequential order by key. The new record is written in either the cylinder overflow area or an independent overflow area. If these areas are filled, a bit is posted in filenameC.

If the new record is higher than all records presently in the file (end-of-file), ISAM checks tc determine if the last track containing data records is filled. If it is not, the new record is added, replacing the end-of-file record. The end-of-file record is written in the next record location on the track, or on the next available prime data track. Another track must be available within the file limits. If the end-of-file record is the first record on any track, the new record is written in the appropriate overflow area. After each new record is inserted in its proper location, ISAM adjusts all indexes affected by the addition.

ISAM MACROS FOR RANDOM RETRIEVAL

When a file is organized by ISAM, records can be retrieved in random order for processing and/or updating. Retrieval must be specified in the DTFIS entry IOROUT (IOROUT=RETRVE or IOROUT=ADDRTR). Random processing must be specified in the DTFIS entry TYPEFLE=RANDOM or RANSEQ.

Because random reference to the file is by record key, the problem program must supply the key of the desired record. To do this, the key must be stored in the main storage key field specified by the DTFIS entry KEYARG. The specified key designates both the record to be retrieved and the record to be written back into the file in an updating operation. Adding and updating should not be interspersed. Records that are added to a file (between the read and write macro for a particular record to be updated) can result in a lost record and a duplicate key.

The RECSIZE entry in the DTFIS should specify the same value as entered at load time. If these values differ, no error will result; however, the RECSIZE from the load DTFIS is used. The necessary information for a retrieval operation comes from the Format 2 label and not the RETRVE operand in the DTFIS.

READ Macro for Random Retrieval

Name	Operation	Operand
[name]	READ	{filename / (1)} ,KEY

****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

The READ instruction causes ISAM to retrieve the specified record from the file. This instruction requires two parameters. The first parameter specifies

the name of the file from which the record is to be transferred to main storage. This name is the same as the name specified in the DTFIS header entry for the file and can be specified as a symbol or in register notation. The second parameter must be the word KEY.

To locate a record, ISAM first searches the indexes to determine the track on which the record is stored and then searches the track for the specific record. When the record is found, ISAM transfers it to the I/O area specified by the DTFIS entry IOAREAR. The ISAM routines also move the record to the specified workarea if the DTFIS entry WORKR is included in the file definition.

When records are blocked, ISAM transfers the block that contains the specified record to the I/O area. It makes the individual record available for processing either in the I/O area or the workarea (if specified). For processing in the I/O area, ISAM supplies the address of the record in the register specified by DTFIS IOREG. The ID of the record can be referenced using filenameG. A WAITF macro must be issued following a READ macro for random retrieval.

WRITE Macro for Random Retrieval

Name	Operation	Operand
[name]	WRITE	{filename / (1)} ,KEY

****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

The WRITE instruction with the parameter KEY is used for random updating. It causes ISAM to transfer the specified record from main storage to DASD storage. This instruction requires two parameters. The first parameter specifies the name of the file to which the record is transferred. The specified name is the same as that used in the DTFIS header entry and in the preceding READ instruction for the file. The name can be specified as a symbol or in register notation. The second parameter must be the word KEY.

ISAM rewrites the record following a read instruction for the same file. The record is updated from the workarea (if one is specified) or the I/O area. The key need not be specified again ahead of the WRITE instruction. A WAITF macro must be issued following a WRITE macro for random retrieval.

WAITF Macro for Random Retrieval

Name	Operation	Operand
[name]	WAITF	{filename / (1)}

The WAITF macro instruction is issued to ensure that record transfer is completed. The only required parameter is the name of the file to which the record is transferred. The name can be specified as a symbol or in register notation.

This instruction must be issued before the problem program attempts to process an input record or build another output record for the designated file. The program does not regain control until the previous transfer of data is complete, unless ERREXT=YES was specified and an error occurs. In this case, the ERET macro should be issued to handle the error and complete the transfer of data.

The WAITF instruction posts any exceptional information in the DTFIS table at filenameC. The WAITF instruction applies to the functions described in ISAM Macros for Adding Records, and ISAM Macros for Random Retrieval.

ISAM MACROS FOR SEQUENTIAL RETRIEVAL

When a file is organized by ISAM, records can be retrieved in sequential order by key for processing and/or updating. The DTFIS entry IOROUT=RETRVE must be specified. Sequential processing must be specified in the DTFIS entry TYPEFLE=SEQNTL or RANSEQ.

Although records are retrieved in order by key, sequential retrieval can start at a record in the file identified either by key

or by the ID (identifier in the count area) of a record in the prime data area. Sequential retrieval can also start at the beginning of the logical file. The user must specify, in SETL, the type of reference he uses in the problem program.

Whenever the starting reference is by key and the file contains blocked records (RECFORM=FIXBLK), the user must also provide ISAM with the position of the key field within the records. This is specified in the DTFIS entry KEYLOC. To search for a record, ISAM first locates the correct block by the key in the key area of the DASD record. The key area contains the key of the highest record in the block. ISAM then examines the key field within each record in the block to find the specified record. As with random retrieval, the RECSIZE operand should specify the same number as indicated when the DTFIS was loaded.

SETL Macro for Sequential Retrieval

Name	Operation	Operand
[name]	SETL	{filename / (r)}, {idname / (r) / KEY / BOF / GKEY}

The SETL (set limits) macro instruction initiates the mode for sequential retrieval and initializes the ISAM routines to begin retrieval at the specified starting address. The first operand (filename) specifies the name of the file (supplied in the DTFIS header entry) from which records are to be retrieved. The name can be given as a symbol or in register notation. Register notation is necessary to allow the macro to be used in a self-relocating program.

The second operand specifies where processing is to begin. If the user is processing by the record ID, the operand idname or (r) specifies the symbolic name of the main-storage field in which the user supplies the starting (or lowest) reference for ISAM use. The symbolic field contains the following information:

Pointer to First Record to be Processed by Sequential Retrieval:

Byte	Identifier	Contents	Information
0	m	2-245	Number of the extent in which the starting record is located.
1-2	b,b	0,0 (disk)	Always zero for disk.
		0, 0-9 (cell)	Cell number for data cell.
3-4	c,c	0, 1-199 (disk)	Cylinder number for disk.
		0-19,0-9 (cell)	Subcell and strip for data cell. Note: The last four strips on each cell are reserved for alternate tracks.
5-6	h,h	0,0-9 (2311 disk) 0,0-19 (2314 or 2319 disk)	Head position for 2311, 2314, or 2319 disk.
		0-4,0-19 (cell)	Cylinder and head for data cell.
7	r	1-254	Record location.

If processing begins with a key supplied by the user, the second operand is KEY. The key is supplied by the user in the field specified by the DTFIS entry KEYARG. If the specified key is not present in the file, an indication is given at filenameC. The second operand BOF specifies that retrieval is to start at the beginning of the logical file.

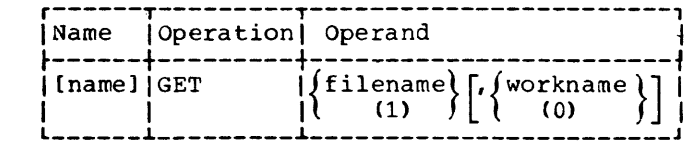
Selected groups of records within a file containing identical characters or data in the first locations of each key can be selected by specifying GKEY in the second operand. The GKEY specification allows processing to begin at the first record (or key) within the desired group. The user must supply a key that identifies the significant (high order) bytes of the required group of keys. The remainder (or insignificant) bytes of the key must be padded with blanks, binary zeros, or bytes lower in collating sequence than any of the insignificant bytes in the first key of the group to be processed. For example, a GKEY

specification of D6420000 would permit file processing to begin at the first record (or key) containing D642xxxx, regardless of the characters represented by the x's. The problem program must determine when the generic group is completed. Otherwise, ISAM continues through the remainder of the file.

Note: If the search key is greater than the highest key on the file, the Filename status byte is set to

X'10' NO RECORD FOUND

GET Macro for Sequential Retrieval



****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

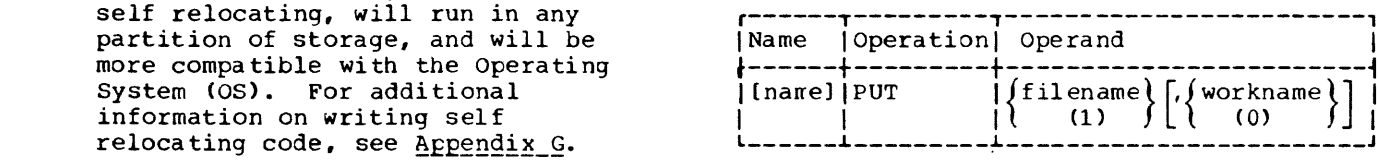
The GET macro instruction causes ISAM to retrieve the next record in sequence from the file. It can be written in either of two forms, depending on where the record is to be processed.

The first form is used if records are to be processed in the I/O area (specified by DTFIS IOAREAS). The only required parameter is the name of the file from which the record is to be retrieved. This is the same name as that specified in the DTFIS header entry and can be specified as a symbol or in register notation. ISAM transfers the record from the file to the I/O area after which the record is available for the execution of the next instruction in the problem program. The key is located at the beginning of IOAREAS and the register (IOREG) points to the data. If blocked records are specified, ISAM makes each record available by supplying its address in the register specified by the DTFIS entry IOREG. The key is contained in the record.

The second form of the GET instruction is used if records are to be processed in a workarea (DTFIS specifies WORKS). It requires two parameters both of which can be specified as symbols or in register notation. The first is the name of the file, and the second is the name of the workarea. When using register notation, workname should not be preloaded into register 1.

If blocked records are specified in the file definition, each GET that transfers a block of records to main storage will also write the preceding block back into the file in its previous location. GET writes the preceding block if a PUT instruction is issued for at least one of the records in the block. If a PUT instruction was not issued, updating is not required for the block and GET does not rewrite the block. Whenever an unblocked record is retrieved from the prime data area, ISAM supplies the ID of that record in the field addressed by filenameH. If blocked records are specified, ISAM supplies the ID of the block.

PUT Macro for Sequential Retrieval



****

Recommendation: To write the most efficient code (in a multiprogramming environment), register notation should be used for this macro in conjunction with the OPENR macro. If this is done, user programs will be self relocating, will run in any partition of storage, and will be more compatible with the Operating System (OS). For additional information on writing self relocating code, see Appendix G.

****

The PUT macro instruction is used for sequential updating of a file, and causes ISAM to transfer records to the file in sequential order. PUT returns a record to a file. It may be written in either of two forms, depending on where records are processed. A GET macro must precede each PUT macro.

The first form is used if records are processed in the I/O area (specified by DTFIS IOAREAS). It requires only the name of the file to which the records are to be transferred. The specified macro is the same as that used in the DTFIS header entry and can be specified in register notation or as a symbol.

The second form of the PUT instruction is used if records are processed in a workarea. It requires two parameters, both of which can be specified either as a symbol or in register notation. The first parameter is the name of the file, and the second is the name of the workarea. When using register notation, workname should not be loaded into register 1. The workarea name may be the same as that specified in the preceding GET for the file, but this is not required. ISAM moves the record from the workarea specified in the PUT instruction to the I/O area specified for the file in the DTFIS entry IOAREAS.

When unblocked records are specified, each PUT writes a record back onto the file in the same location from which it was retrieved by the preceding GET for the file. Thus, each PUT updates the last record that was retrieved from the file. If some records do not require updating, a series of GET instructions can be issued without intervening PUT instructions. Therefore, it is not necessary to rewrite unchanged records.

When blocked records are specified, PUT instructions do not transfer records to the file. Instead, each PUT indicates that the block is to be written after all the records in the block are processed. When processing for the block is complete and a GET is issued to read the next block into main storage, the GET also writes the completed block back into the file in its previous location. If a PUT instruction is not issued for any record in the block, GET does not write the completed block. The ESETL macro instruction writes the last block processed, if necessary, before the end-of-file.

ESETL Macro for Sequential Retrieval

Name	Operation	Operand
[name]	ESETL	{filename / (1)}

The ESETL (end set limit) macro instruction ends the sequential mode initiated by the SETL macro. The name of the file must be the same as the name specified in the DTFIS header entry. It can be specified as a symbol or in register notation. If blocked records are specified, ESETL writes the last block back if a PUT was issued. Register notation is necessary to allow use of the macro in a self relocating program.

Notes: If ADDRTR and/or RANSEQ are specified in the same DTF, ESETL should be issued before issuing a READ or WRITE; another SETL can be issued to restart sequential retrieval. Sequential processing must always be terminated by issuing an ESETL macro. For additional information about ESETL, see the section FREE Macro.

Completion - CLOSE(R) Macro

Op	Operand
for self-relocating programs	
CLOSER	{filename1 / (r1)} [,{filename2 / (r2)}...,{filenamen / (rn)}]
for programs that are not self-relocating	
CLOSE	{filename1 / (r1)} [,{filename2 / (r2)}...,{filenamen / (rn)}]

Note: To write the most efficient code (in a multiprogramming environment), we recommend that the self relocating form of OPEN be used. (See also Appendix G.)

The CLOSE macro instruction deactivates any file that was previously opened in any input/output unit in the system. A file may be closed at any time by issuing this macro instruction. Once a file is closed, no further commands can be issued for the file unless it is reopened.

If a load or load extension file is not closed, the format-2 label associated with the file is not updated with the information that is in the DTF. Further processing of such a file may give unpredictable results.

When the operation CLOSE is used, the symbolic address constants that CLOSE generates from the parameter list are not self-relocating. When CLOSER is specified, the symbolic address constants are self-relocating. This latter form (CLOSER) is therefore recommended.

The symbolic name of the logical file (assigned in the DTF header entry) to be closed is entered in the operand field. A maximum of 16 files may be closed by one instruction by entering additional filename parameters as operands. Alternately, the user can load the address of the filename in a register and specify the register using ordinary register notation. The high-order 8 bits of this register must be zeros. The address of the filename may be preloaded into any register, 0-15. See Label Processing.

Multitasking Macros

When multitasking is specified at system generation time, the multitasking group of macros allows more than one program (task) to execute within a partition. This not only increases overlapping of I/O activity and computer processing, but also increases throughput by eliminating operator intervention to initiate these programs.

The multitasking function enables nine additional subprograms, or subtasks, to be added to the three-partitioned Disk Operating System. Thus, a partition to be multitasked consists of the main task and one to nine subtasks. Each subtask must be initiated (attached) by the main task, at which time control passes to the subtask. The storage protection key and priority of the partition (F1, F2, BG) remain the same, but the priority of a task within a partition is determined according to the order in which a subtask is attached. That is, if subtask 1 through subtask n (n less than or equal to 9) are attached, in ascending order, the priority of the partition is from subtask 1 to subtask n followed by the main task. Thus, to facilitate maximum I/O overlap, the subtask with the most I/O activity should be attached first within the foreground-1 partition.

As an example of multitasking, assume a situation where input/output overlapping is to be achieved by running three programs in one partition. The main task attaches each subtask, checks to ensure each subtask is properly attached, and ends the job when the subtasks are completed. Furthermore, it is assumed that this partition has the highest priority. For this case, processing occurs as follows:

1. The main task initializes the program, attaches subtask 1 and passes control to this subtask.

2. Subtask 1 processes until it issues an I/O command or enters the wait state at which time it passes control to the main task.

3. The main task similarly initiates the second and third subtasks. This achieves not only I/O overlap, but also initiates new jobs without operator intervention.

If an unrecoverable error occurs in the main task, the problem program abnormal termination routine closes the files and terminates the job. If an unrecoverable error occurs in a subtask, a second abnormal termination routine must be provided to close that subtask's files and cancel the subtask. Figure 49 describes this operation more fully.

There are three main types of multitasking within a partition. Subtasks can:

1. Be independent of each other, or

2. Be interrelated with macro instructions used for intertask communication, or

3. Consist of one physical set of coding that is reenterable. This set of coding can be attached up to nine times to operate as if up to nine subtasks were being executed. For multitasking examples of types 2 and 3, see Multitasking Considerations.

Subtasks share main storage with the main task in any way determined by the problem program. The subtask operates independently of the main task and effectively has its own registers. A task need not be affected by other operations and can include its own program check and/or abnormal termination exit through the STXIT macro instruction.

To synchronize tasks, the proper intertask communication macro instructions WAIT, WAITM, and POST must be executed. The WAIT or WAITM enables a task to wait for an event to occur. The task remains waiting until allowed to continue by an I/O completion, a timer interrupt, a task termination, or a POST macro.

If data is manipulated in main storage by one task, it can also be manipulated by a task with a lower priority unless the resource protection macros (RCB, ENQ, and DEQ) are used by all the tasks in the partition. Likewise, shared data manipulated on a DASD device is not protected unless all the files concerned (DTFSD or DTFDA) specify the HOLD option. For additional information about multitasking, see the DOS System Programmer's Guide listed on the front cover.

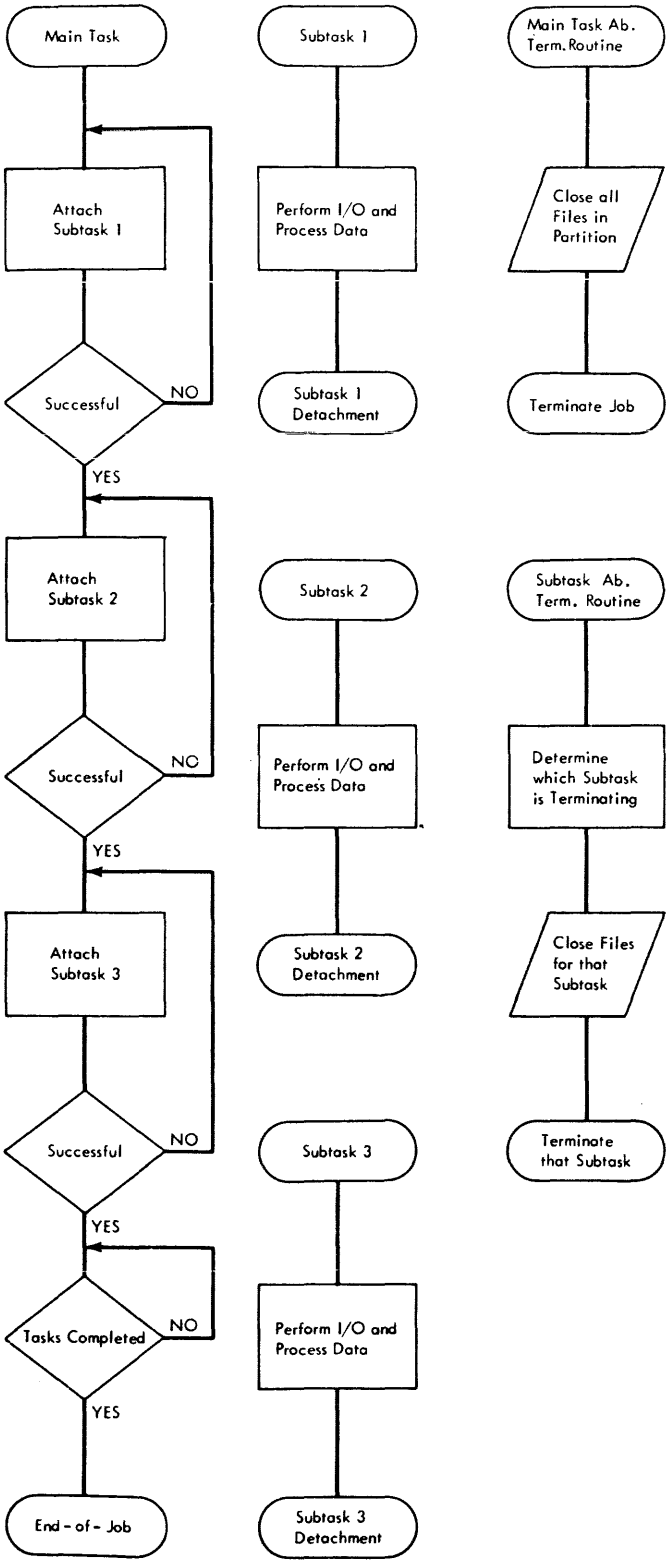


Figure 49. A Multitasking Flowchart Example

Subtask Initiation and Normal Termination Macros

ATTACH Macro

Name	Op	Operands
[name]	ATTACH	{entrypt / (r0)}, SAVE={savearea / (r1)}
		[,ECB={ecbname / (r2)}]
		[,ABSAVE={savearea / (r3)}]

A subtask can only be initiated by issuing the ATTACH macro within the main task. The entry point of the subtask must be in main storage before the subtask can be successfully attached.

The first operand must be the entry point of the subtask and can be specified as a symbol or in register notation. Register 1 should not be used.

The second operand must be the address of the savearea for the subtask. The second operand can be given as a symbol, or in special or ordinary register notation. The save area must be aligned on a doubleword boundary and is 96 or 128 bytes in length depending upon whether the floating point option (CONFG FP=YES) was specified at system generation time.

Savearea without Floating Point Option

NAME	PSW	REGS 9-8	USED BY DOS
0	8	16	80 96

Savearea with Floating Point Option

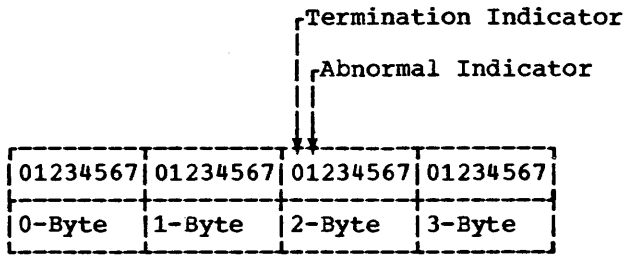
NAME	PSW	REGS 9-8	USED BY DOS
0	8	16	80

FP REGS	USED BY DOS
88	120 128

The area (as shown) contains the subtask's PSW, general purpose registers, floating

point registers (option dependent), and a 16-byte area used by the Disk Operating System. A subtask name should be provided in the first 8 bytes of the save area. This identifies the subtask in the event of a possible abnormal termination condition.

The third operand must be specified if other tasks can be affected by this subtask's termination, or if the ENQ and DEQ macros are used within the subtask. This parameter is the address of the task's event control block (ECB), and is a fullword defined by the problem program. At the time a subtask is attached, byte 2 bits 0 and 1 are set to 0. When a subtask terminates, the supervisor sets byte 2 bit 0 of the ECB to 1. In addition, if AB=YES is generated in the supervisor, byte 2 bit 1 is posted when the subtask abnormally terminates; that is, if task termination is not the result of issuing the CANCEL, DETACH, DUMP, or EOJ macro instructions. The remaining bits of an ATTACH ECB are reserved for future use. However, the intertask communication ECB may be any 4 byte (or larger) field with the following format:



The fourth operand should only be specified if the subtask is to execute the main task abnormal termination routine. See STXIT--Set Linkage to User Routine(s). The problem program can have separate subtask STXIT AB routines with or without a main task STXIT AB routine, or it can have neither. The parameter specified in this operand must be the address of a 72-byte (doubleword aligned) STXIT save area for the subtask. When an abnormal termination occurs, the supervisor saves the old PSW and general registers 0-15 in this area before the exit is taken.

If the ATTACH macro successfully initiates a subtask, control passes to the subtask. Register 1 of the subtask contains the address of the main task save area and the contents of registers 2-15 remain unchanged. The address in register 1 can be used as the second operand of a POST macro later in the job if specific task-to-task communication is desired.

Upon return from a successful ATTACH, the main task register 0 contains the address of the byte immediately following the subtask save area, as determined by the supervisor. Register 0 can be tested to ascertain whether the supervisor contains the floating point option.

If nine subtasks are already attached within the system, any attempt to attach another subtask is unsuccessful. In such an event, the main task keeps control and register 1 (main task) contains the address of an ECB within the supervisor that is posted when the system can initiate another subtask. Register 1 also has the high order bit 0 ON to aid the main task in testing for an unsuccessful ATTACH.

DETACH Macro

Name	Operation	Operands
[name]	DETACH	[SAVE={savearea / (1)}]

A subtask is normally terminated by issuing a DETACH macro, and no operand is required in this case. The main task can also terminate a subtask it initiated by issuing the DETACH macro with an operand. The operand provides the address of the save area specified in the ATTACH macro of the subtask to be terminated.

> Note: If the main task issues the macro without specifying an operand, the whole partition is terminated.

The DETACH macro posts the ECB, if specified in the ATTACH macro, (byte 2 bit 0) to indicate normal termination. All tasks waiting on this ECB are taken out of the wait state, and the highest priority task obtains control.

> Note: For systems operating in a QTAM environment, QTAM files must be closed before issuing this macro.

RCB Macro

Name	Operation	Operand
[name]	RCB	

When more than one task manipulates a resource (data in the same area, an I/O device, a set of instructions, etc), protection should be provided to prevent the resource from being used concurrently by these tasks. If every task within the partition uses the RCB, ENQ, and DEQ macros, such protection is possible.

The RCB macro generates an 8-byte word-aligned Resource Control Block, which protects a user defined resource if the ENQ macro is issued before and the DEQ macro is issued after each use of the resource. Byte 0 of this field is set to binary ones to indicate that the resource is placed in a priority queue by the ENQ macro. RCB bytes 1-3 are reserved for future use.

If bit 0 of the flag byte is ON, it indicates that another task is waiting to use the resource. At this time, RCB bytes 5-7 contain the ECB address of the current resource owner. The format of the RCB is:

Queue Byte	Reserved			Flag Byte	ECB Address of Current Resource Owner		
0	1	2	3	4	5	6	7

ENQ Macro

Name	Operation	Operand
[name]	ENQ	{rcb-name} {(0)}

A task protects a resource by issuing an ENQ (enqueue) macro. When the RCB, (identified by the rcb-name) is enqueued, the task requesting the resource is either queued and executed, or it is placed in a wait condition. When a task using that resource completes, the DEQ (dequeue) macro is issued. All other tasks that were waiting for the dequeued resource are freed from their wait condition, and the highest priority task either obtains or maintains control.

If a task is terminated without dequeuing its queued resources, any task subsequently trying to enqueue that resource is abnormally terminated. If a task issues two ENQs without an intervening DEQ for the same resource, the task is canceled. Also, any task that does not control a resource attempting to dequeue that resource is terminated, unless the DEQ appears in the abnormal termination routine. If DEQ appears in the abnormal termination routine, it is ignored.

Although the main task does not require the problem program to set up an intertask communication ECB to enqueue and dequeue, every subtask using that facility must have the ECB operand in the ATTACH macro, and that ECB must not be used for any other purpose. Also, a resource can only be protected within the partition containing the ECB.

DEQ Macro

Name	Operation	Operand
[name]	DEQ	{rcb-name / (0)}

A task releases a resource by issuing the DEQ macro. If other tasks are enqueued on the same RCB, the DEQ macro frees all other tasks that were waiting for that resource from their wait condition. In such cases, the highest priority task either obtains or maintains control. A task that attempts to dequeue a resource that was not enqueued or that was enqueued by another task is abnormally terminated. Dequeuing under these two conditions (within an abnormal termination routine) results in a no operation instruction.

The operand is the same as that in ENQ and specifies the address of the RCB either by a symbolic name, special register notation, or ordinary register notation.

The following example shows how an RCB can be used to protect an area in main storage:

```
MTASK  START   0      Example
       .
       .
       .
       ATTACH  STASK1,SAVE=SAVE1,ECB=ECB1
       ATTACH  STASK2,SAVE=SAVE2,ECB=ECB2
       .
       .
       .
STASK1 ENQ     RCBA
       update  TOTAL
       DEQ     RCBA
       .
       .
       .
STASK2 ENQ     RCBA
       update  TOTAL
       DEQ     RCBA
       .
       .
       .
RCBA   RCB
TOTAL  DS or DTFxx
```

TOTAL can be simply an area in main storage or a file defined by a declarative macro. In either case, TOTAL is protected from subtask 2 while subtask 1 is operating with it. Thus, if all tasks enqueue and dequeue around all references to TOTAL, TOTAL is protected during the time each task takes to process instructions from the task's ENQ instruction to its DEQ instruction. This is readily apparent if TOTAL is an area in main storage. However, if TOTAL is a file, the record that is being operated upon is protected while in main storage, but it is not necessarily protected on the external storage device. If the file is on a DASD, the HOLD function should be used.

INTERTASK COMMUNICATION MACROS

WAITM Macro

Name	Operation	Operand
[name]	WAITM	{ecb1,ecb2,... / listname / (1)}

If the option is specified at system generation time, the WAITM macro enables the problem program or task to wait for one of a number of events to occur. Control returns to the task when at least one of the ECBs specified in the macro is posted.

The operand provides the address of the ECBs to be waited upon. The symbolic names of ecb1,ecb2... are assumed when at least two operands are supplied. If one operand is supplied, it is assumed to be the symbolic name (listname) of a list of consecutive full word addresses that point to the ECBs to be waited upon. The first byte following the last address in the list must be nonzero to indicate the end of the list. The listname parameter can be given as a symbol, in special register notation, or in ordinary register notation.

When control returns to a waiting task, register 1 points to the posted ECB that had byte 2 bit 0 ON. Other blocks can be used as ECBs if their byte 2 bit 0 indicates a completed event. Examples of these blocks are CCBs and TECBs. However, a task never regains control if it is waiting for a CCB to be posted by another task's I/O completion. A MICR CCB gets posted only when the device stops, not when a record is read. Furthermore, telecommunication ECBs, QTAM control blocks, and all RCBs must not be waited for because their format would never satisfy a WAIT or a WAITM (that is, byte 2 bit 0 would not be posted).

A task that issues the WAITM macro should ensure that the waiting task allows an eventual outlet if it is possible that an event will not occur. (Such a condition could occur if a task posting an event is terminated.) This outlet can also wait for the termination ECB of the task that is to

perform the preferred event. An example of a successful intertask communication is:

```
          WAITM     ECB1A,ECB1
          B         4(1)
          .
          .
          .
ECB1A     DC        F'0'
          B         PEVENT
ECB1      DC        F'0'
          B         TEVENT
```

In this example, the WAITM macro contains a preferred event as the first operand, and a secondary event as the second operand. The preferred event is the posting of ECB1A after subtask 1 completes its processing. If subtask 1 terminates before its processing is completed, the supervisor posts the ATTACH macro ECB of subtask 1, ECB1, and the secondary event can satisfy the WAITM macro. In either case, after the WAITM macro is satisfied, the address of the posted ECB is contained in register 1. This address can be used to select a problem program routine. In this particular case, a branch instruction points to a table containing a list of ECBs with corresponding branch instructions to the routine to be given control when the ECB is posted. This table can easily be expanded to include up to a maximum of 16 ECBs.

POST Macro

Name	Operation	Operand
[name]	POST	{ecbname / (1)}
		[,SAVE={savearea / (0)}]

This macro provides intertask communication by posting an ECB (turn byte 2 bit 0 ON). A POST issued to an ECB removes a task waiting for the ECB from the wait state. The first operand provides the address of the ECB to be posted. It can be provided as a symbol, in special register notation, or in ordinary register notation.

If the SAVE operand is present, only the task identified by the address of its save area is taken out of the wait state. This task normally is waiting for the specified ECB to be posted. Although time is saved by specifying this operand, other tasks waiting for this ECB are not taken out of the wait state for this event unless another POST is issued. When a POST is issued without the SAVE operand, all tasks waiting for the ECB are taken out of the wait state, and the highest priority task regains control.

DASD TRACK PROTECTION MACROS

DASD track protection means that when a record on a DASD track is being modified by a task in main storage, that track is prevented from being accessed by another task. Within a partition, track protection can be accomplished for a particular DASD by the resource protection macros or the intertask communication macros. With the resource protection macros, an RCB can be enqueued before each reference to the DASD. With the intertask communication macros, a subtask can wait for an ECB to be posted before each reference to the DASD.

For programs using the DTFSD-SDMOD (data files with updating, or work files with updating), DTFIS-ISMOD, and/or DTFDA-DAMOD macros, the track hold function can provide DASD track protection. In these cases, DASD track protection within the entire system can be accomplished if the track hold option is specified at system generation time, and if every task uses the DTFSD-SDMOD, DTFIS-ISMOD, and/or DTFDA-DAMOD HOLD=YES macros to access the DASD. If protection is required within a partition, the track hold function must be used for every read within the partition.

The track hold function can be used in four specific situations:

1. DTFSD updating files without work files.
2. DTFSD updating files with work files.
3. DTFDA files.
4. DTFIS files.

In the first situation, the track being held is freed automatically by the system. More specifically, for situation 1, the next GET issued to a new track for the file frees the previous hold. For situation 2, the track is automatically freed by the system if the record that was read and held is then updated. If it is not updated, the problem program must issue the FREE macro. For situation 3, the problem program must issue the FREE macro for each hold placed on the track. A hold is placed on a track each time the track is accessed with a GET or a READ, and each hold is released by issuing either a FREE or CLOSE macro to that file, or a DETACH macro to that task. For situation 4, the method of

implementation depends on the function being performed.

The format of the FREE macro is:

Name	Operation	Operand
[name]	FREE	{filename / (1)}

The maximum number of tracks that can be held within a system is specified at system generation time. The maximum that can be specified is 255, with a system default option of 10. If a task attempts to exceed the limit, the task is placed in the wait state until a previously held track is freed.

The same track can be held more than once without an intervening FREE if the hold requests are from the same task. The same number of FREEs must be issued before the track is completely freed. However, a task is terminated if more than 16 hold requests are recorded without an intervening FREE, or if a FREE is issued to a file that does not have a hold request for that track.

For DTFDA files using WRITE or WRITE AFTER, DAMOD initially places a HOLD on the track. Before returning control to the problem program, DAMOD automatically issues a FREE to that track. However, a WRITE AFTER issued to a track that has the maximum number of HOLDs already in effect cancels the task (or partition).

If a task requests a track that is being held by another task, that task is placed into the wait state at the GET or WAITF macro associated with the I/C request. The request is fulfilled after the track is freed and when control returns to the requestor.

If more than one track is being held, it is possible for the problem program to inadvertently put the entire system in the wait state. This occurs if each task is waiting for a track that is already held by another task. A way to prevent this, is to FREE each track held before another track hold is attempted.

For DTFIS files, bit 2 of byte 72 in the format 2 label is reset to 0 whenever a file is opened for ADD or ADDRTR. If this bit is already 0 when HOLD=YES, the problem program is canceled because another program is already using the file for an ADD or ADDRTR. When the file is closed, the bit is set to 1. This switch prevents two programs from trying to update the same file because it does not allow a second open for ADD or ADDRTR on the same file when HOLD=YES.

If for any reason a file is not closed during execution of a job in which ADD or ADDRTR is specified, this file cannot be opened if the next job using this file specifies ADD or ADDRTR when HOLD=YES. Bit 2 of byte 72 of the format 2 label must first be set to 1 by issuing a CLOSE to that file in any job in which ADD, ADDRTR, or LOAD is specified and HOLD does not equal YES.

The method of implementation for ISAM track hold depends on the function being performed:

- Sequential Retrieval - The track index is held at the beginning of retrieval from each cylinder. A search and hold is issued for the data track, the index track is released, and a wait is issued for the data track. When the system is finished with the data track (prime or overflow), it is released, and the next track is held. The problem program must release the track hold function by issuing either a PUT (if the file is updated) or a GET (no update) for the next record, or an ESETL.

- Random Retrieval - The track index is held while the needed entries from it are read in. The data track is held, and the desired record is searched for. When the record is found, the track index is released. The problem program must release the data track by issuing a WRITE (if the file is updated) or a FREE (no update).

- Add - The track index and the data track are held. If the record is not going onto the prime data track, the track index is released. All tracks being changed are held during modification. The track index is again held while it is updated to reflect the added records. After alteration, the tracks are released by the system.

- SETL Macro - SETL issues a hold on the track index on which processing will begin. This hold is released by the system at the appropriate time.

- ESETL Macro - ESETL frees any tracks that are held by sequential retrieval when the ESETL is issued. Since the ESETL macro issues a FREE whether or not any tracks are held, the user should not issue ESETL if SETL has not been successful.

Shared Modules and Files

The DTF and logic modules for the card, device independent, direct access, indexed sequential, printer, sequential disk, and tape macro instructions must contain the operand RDONLY=YES to generate a read only module. Each time a read-only module is entered, register 13 must contain the address of a 72-byte, doubleword-aligned save area. Each DTF requires its own unique save area. The fact that the module save areas are unique, or different for each task, makes the module reentrant (that is, capable of being used concurrently by several tasks). The 72-byte save area required by the RDONLY modules should not be confused with save areas required for multitasking macros.

If the operand is omitted, the module generated is not reenterable, and no save area need be established. If an ERROPT or WLRERR routine issues I/O macro instructions that use the same read-only module that passed control to either error routine, the problem program must provide another save area. One save area is used for the initial I/O and the second for I/O operations in the ERROPT or WLRERR routine. Before control returns to the module that entered the ERROPT routine, register 13 must contain the address of the save area originally specified for that DTF.

Problem programs using devices such as an optical reader can make use of the multitasking function to increase I/O overlap without reentrant modules. However if the problem program ignores module considerations, two tasks may attempt to use a single nonreentrant module. When this occurs, unpredicatable results occur because values for the first task using the module are modified by the second task. To circumvent this situation, several methods can be used.

One method is to assemble a module with a different module name for each task that could attempt to use the module simultaneously. This method requires each module name to be specified in the corresponding DTF MODNAME operand.

Another method is to link-edit each DTF and module separately for each task that could simultaneously attempt to use the same module. Then, before a task attempts to reference a device through that module, the DTF and module can be fetched or loaded into main storage.

Either of these methods prevents the linkage editor from resolving linkage to one module. Thus, separate modules can be provided to perform each function. For more information on the linkage between the DTF and logic module, see the section Interrelationships of the Macro Instructions.

If several tasks are to share processing or reference data on the same file, not only should reentrant modules be employed but each task must contain its own DTF table for that file (unless you use the ENQ and DEQ macros). Each task can either open its own DTF or the main task in the partition can open all files for the subtasks.

There are two methods that can be used for a shared file. The problem programmer can either supply a separate set of label statements (DLBL-EXTENT, TLBL, etc) for each corresponding DTF filename, or he can assemble each DTF and program (subtask) separately with the same filename and one set of label statements. In the latter case, each separately assembled program must open its DTF.

Special consideration must be made for shared multivolume files on a 2321 data cell if DASD file-protect is specified in the supervisor. Within a partition, each task must have its own logical unit assigned to the data cell unless all tasks switch volumes at the same time.

Multitasking Considerations

When the multitasking macros are used, these considerations may be helpful:

1. Only main tasks can issue checkpoints, attach subtasks, or contain an interval timer (STXIT IT) and/or operator communication (STXIT OC) exit routines.

2. Because only one set of system logical units (SYSIN, SYSLOG, etc) exist per partition, interspersed usage by several independent programs is generally not feasible, unless either the resource protection macros or the intertask communication macros are employed.

3. Shared use of SYSLST must be expected by the use of the DUMP, PDUMP, or DSPLYV (display VTOC) functions.

4. If several tasks are writing on or updating the same file (shared file), use either the hold function or the intertask macros to prevent improper modification of data during I/O operations.

5. When using ENQ and DEQ macros with shared DTFs, timing considerations may result in the higher priority subtask not relinquishing processing control during its I/O operations.

In addition to these considerations Figures 50-53 show the detailed flowcharts and coding necessary for two example programs that use multitasking macros.

Example 1, (Figures 50 and 51) shows the use of the intertask communication macros, POST and WAIT, to indicate to subtask 2 that the control cards for subtask 1 were read. This communication is necessary because both tasks are reading control cards from the same symbolic device (SYSIPT). The abnormal termination routine for subtask 1 enables subtask 2 to be processed even if subtask 1 abnormally terminates.

Example 2, (Figures 52 and 53) shows a reenterable set of coding (SUBTASKR) that is attached three times as three independent subtasks. In this example, 100-byte tape input records, blocked four logical records to a physical record, are used to update 100-byte disk records. The disk record address is always located in the first 8 bytes of each logical record. Each subtask updates its record independently of the other subtasks. For optimum throughput, the number of subtasks that should be attached depends upon record sizes, main storage availability, subtask availability, and the amount of processing to be performed.

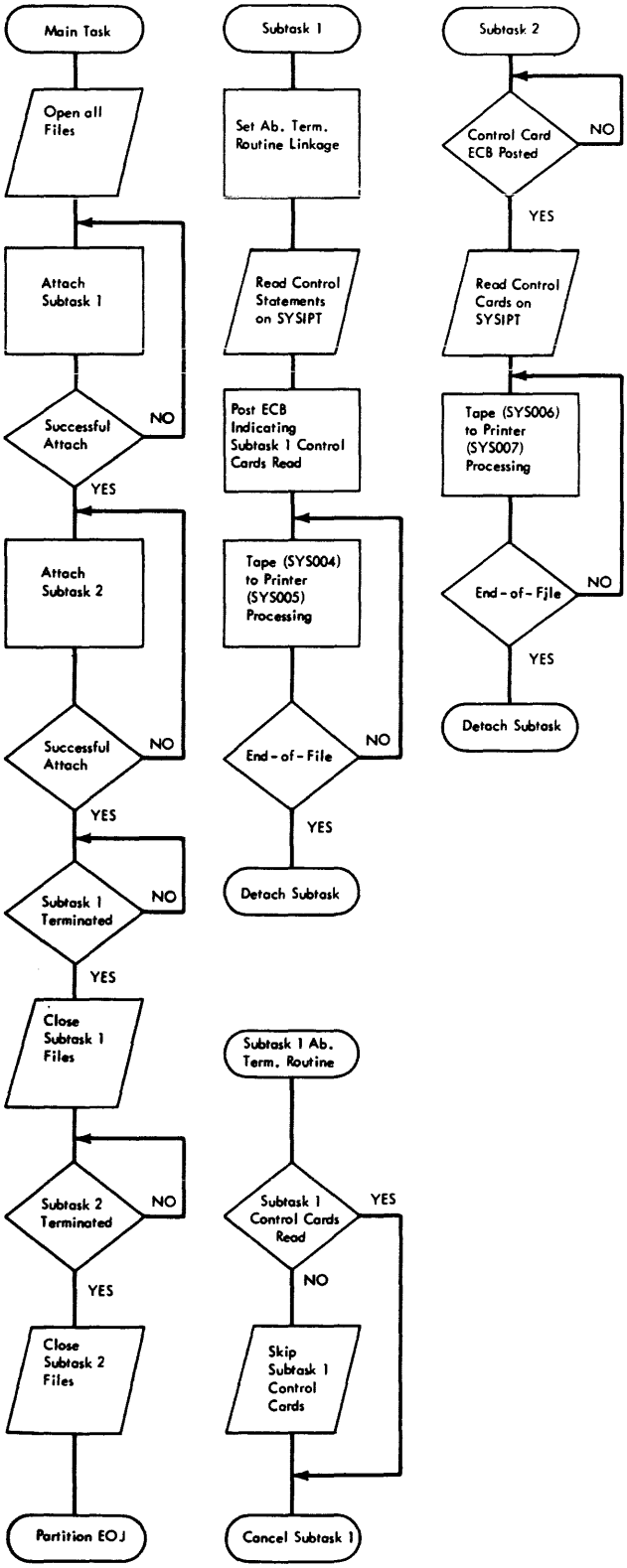
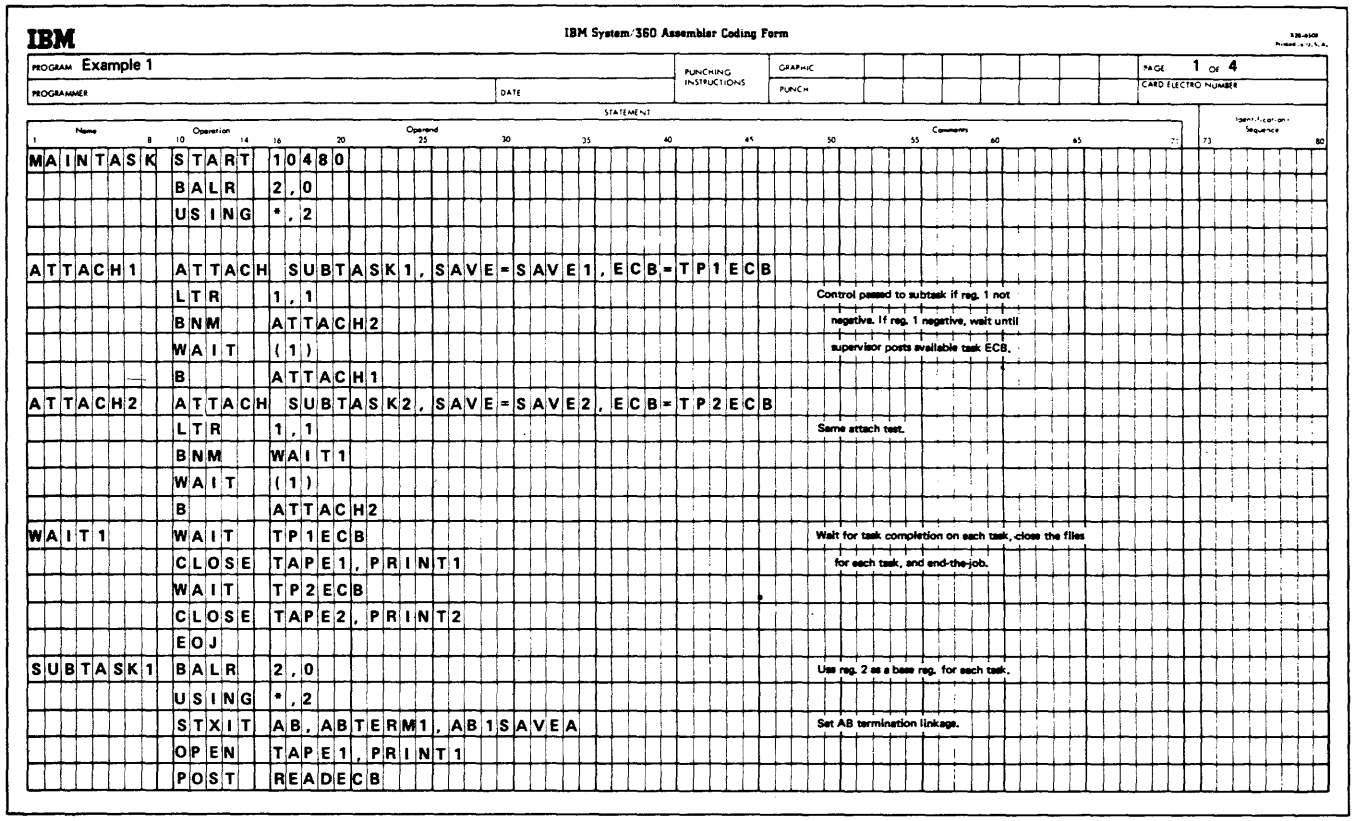


Figure 50. Flowchart for Example 1



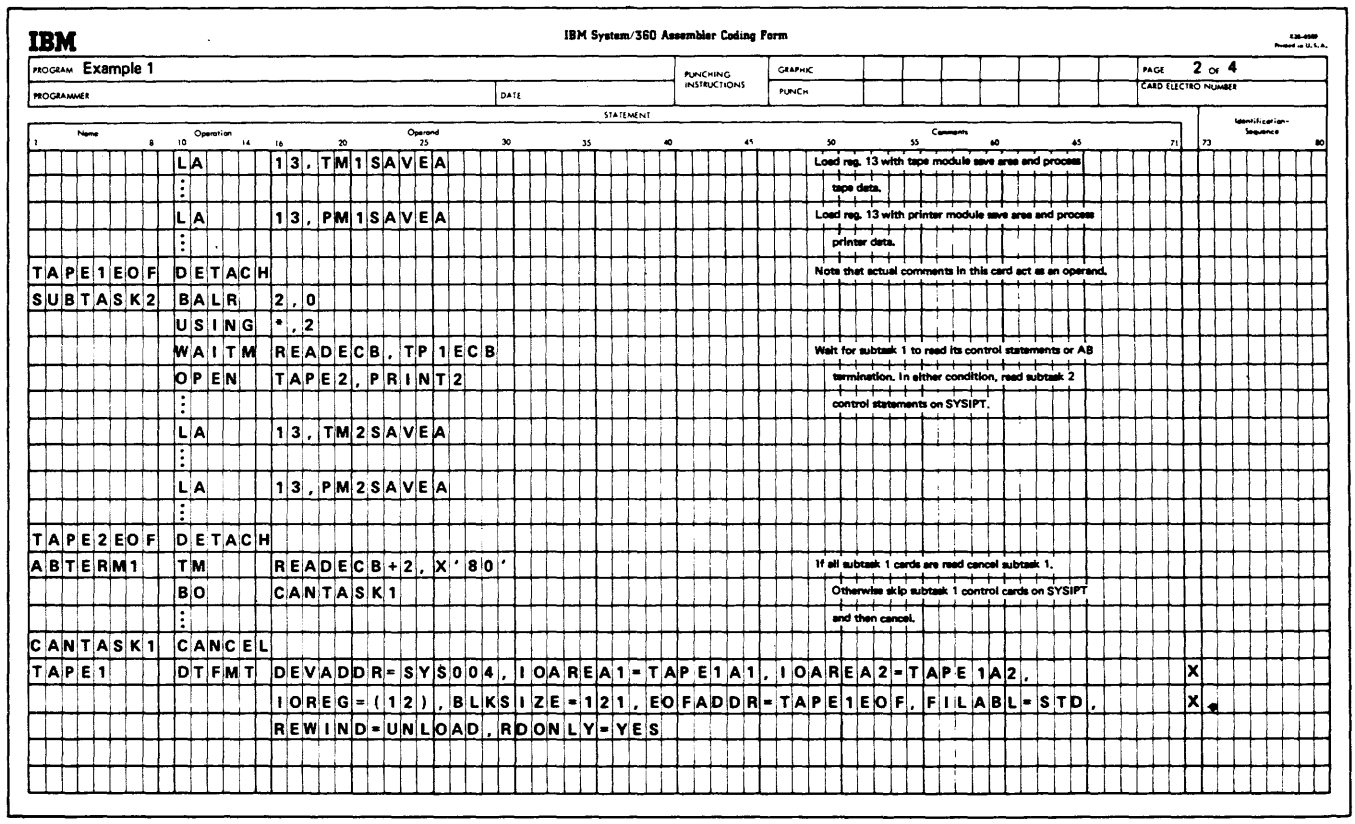
IBM System/360 Assembler Coding Form

PROGRAM Example 1 | PROGRAMMER | DATE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 1 OF 4 | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments
MAINTASK	START	1048 0	
	BALR	2,0	
	USING	*,2	
ATTACH1	ATTACH	SUBTASK1,SAVE=SAVE1,ECB=TP1ECB	
	LTR	1,1	Control passed to subtask if reg. 1 not
	BNM	ATTACH2	negative. If reg. 1 negative, wait until
	WAIT	(1)	supervisor posts available task ECB.
	B	ATTACH1	
ATTACH2	ATTACH	SUBTASK2,SAVE=SAVE2,ECB=TP2ECB	
	LTR	1,1	Same attach test.
	BNM	WAIT1	
	WAIT	(1)	
	B	ATTACH2	
WAIT1	WAIT	TP1ECB	Wait for task completion on each task, close the files
	CLOSE	TAPE1,PRINT1	for each task, and end-the-job.
	WAIT	TP2ECB	
	CLOSE	TAPE2,PRINT2	
	EOJ		
SUBTASK1	BALR	2,0	Use reg. 2 as a base reg. for each task.
	USING	*,2	
	STXIT	AB,ABTERM1,AB1SAVEA	Set AB termination linkage.
	OPEN	TAPE1,PRINT1	
	POST	READECB	

Figure 51. Example 1 (Part 1 of 4)



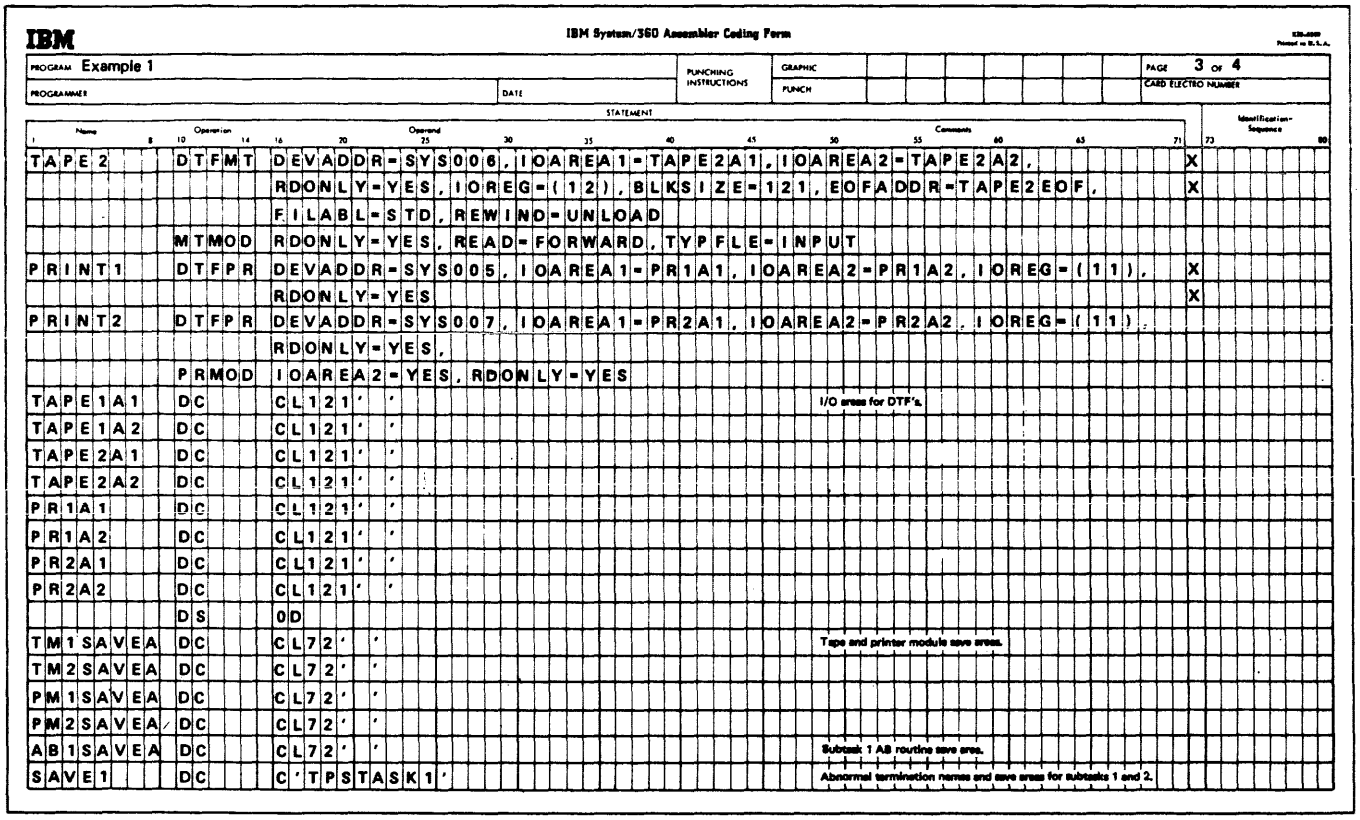
IBM System/360 Assembler Coding Form

PROGRAM Example 1 | PROGRAMMER | DATE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 2 OF 4 | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	
	LA	13,TM1SAVEA	Load reg. 13 with tape module save area and process	
	⋮		tape data.	
	LA	13,PM1SAVEA	Load reg. 13 with printer module save area and process	
	⋮		printer data.	
TAPE1EOF	DETACH		Note that actual comments in this card act as an operand.	
SUBTASK2	BALR	2,0		
	USING	*,2		
	WAITM	READECB,TP1ECB	Wait for subtask 1 to read its control statements or AB	
	OPEN	TAPE2,PRINT2	termination. In either condition, read subtask 2	
	⋮		control statements on SYSIPT.	
	LA	13,TM2SAVEA		
	⋮			
	LA	13,PM2SAVEA		
	⋮			
TAPE2EOF	DETACH			
ABTERM1	TM	READECB+2,X'80'	If all subtask 1 cards are read cancel subtask 1.	
	BO	CANTASK1	Otherwise skip subtask 1 control cards on SYSIPT	
	⋮		and then cancel.	
CANTASK1	CANCEL			
TAPE1	DTFMT	DEVADDR=SYS004,IOAREA1=TAPE1A1,IOAREA2=TAPE1A2,		X
		IOREG=(12),BLKSIZE=121,EOFADDR=TAPE1EOF,FILABL=STD,		X
		REWIND=UNLOAD,RDONLY=YES		

Figure 51. Example 1 (Part 2 of 4)



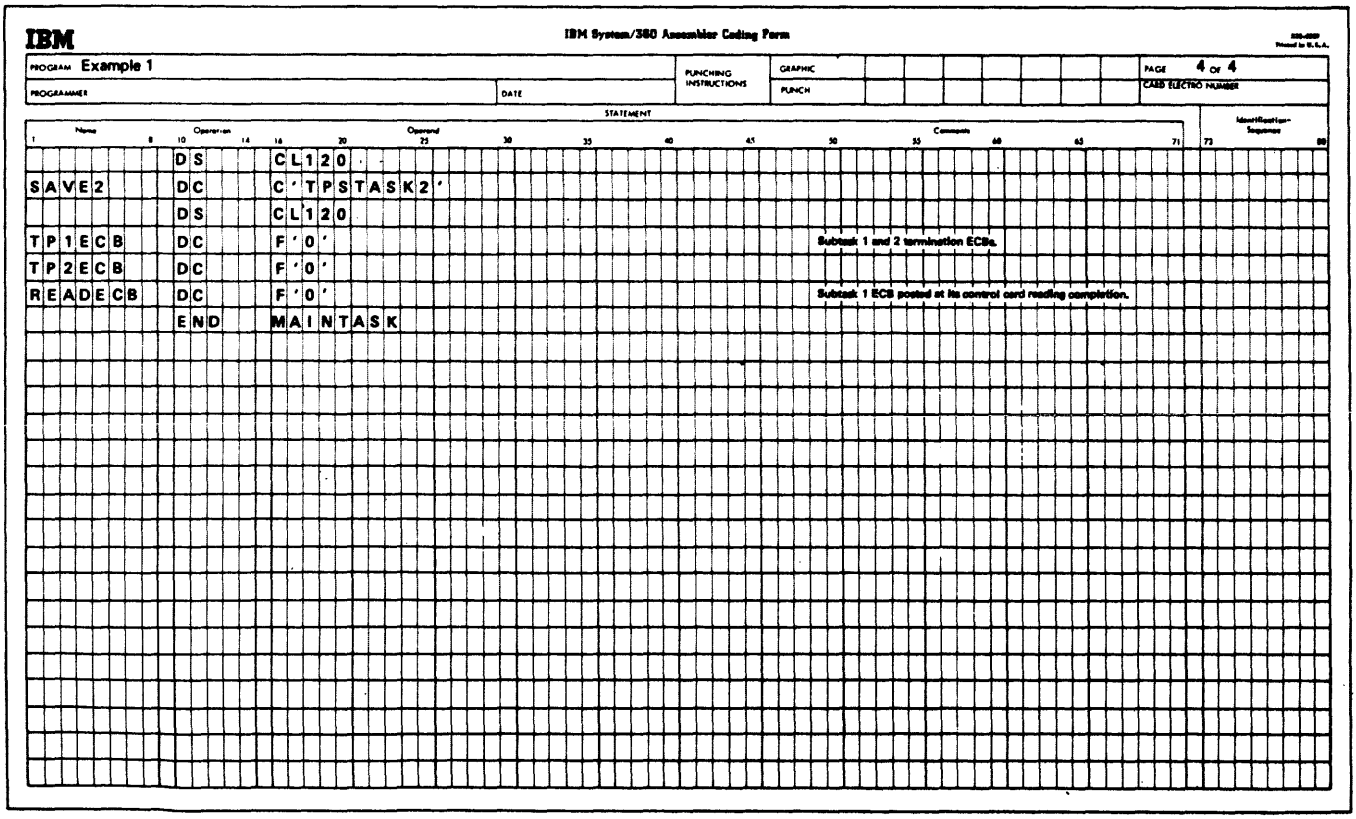
IBM System/360 Assembler Coding Form

PROGRAM Example 1 | PROGRAMMER | DATE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 3 OF 4 | CARD ELECTRO NUMBER

STATEMENT | Name | Operation | Operand | Comments | Identification-Sequence

```
TAPE2    DTFMT  DEVADDR=SYS006,IOAREA1=TAPE2A1,IOAREA2=TAPE2A2,                    X
                RDONLY=YES,IOREG=(12),BLKSIZE=121,EOFADDR=TAPE2EOF,                X
                FILABL=STD,REWIND=UNLOAD
         MTMOD  RDONLY=YES,READ=FORWARD,TYPFLE=INPUT
PRINT1   DTFPR  DEVADDR=SYS005,IOAREA1=PR1A1,IOAREA2=PR1A2,IOREG=(11),             X
                RDONLY=YES                                                         X
PRINT2   DTFPR  DEVADDR=SYS007,IOAREA1=PR2A1,IOAREA2=PR2A2,IOREG=(11),
                RDONLY=YES,
         PRMOD  IOAREA2=YES,RDONLY=YES
TAPE1A1  DC     CL121' '                          I/O areas for DTF's.
TAPE1A2  DC     CL121' '
TAPE2A1  DC     CL121' '
TAPE2A2  DC     CL121' '
PR1A1    DC     CL121' '
PR1A2    DC     CL121' '
PR2A1    DC     CL121' '
PR2A2    DC     CL121' '
         DS     0D
TM1SAVEA DC     CL72' '                           Tape and printer module save areas.
TM2SAVEA DC     CL72' '
PM1SAVEA DC     CL72' '
PM2SAVEA DC     CL72' '
AB1SAVEA DC     CL72' '                           Subtask 1 AB routine save area.
SAVE1    DC     C'TPSTASK1'                       Abnormal termination names and save areas for subtasks 1 and 2.
```

Figure 51. Example 1 (Part 3 of 4)



IBM System/360 Assembler Coding Form

PROGRAM Example 1 | PROGRAMMER | DATE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 4 OF 4 | CARD ELECTRO NUMBER

STATEMENT | Name | Operation | Operand | Comments | Identification-Sequence

```
         DS     CL120
SAVE2    DC     C'TPSTASK2'
         DS     CL120
TP1ECB   DC     F'0'                              Subtask 1 and 2 termination ECBs.
TP2ECB   DC     F'0'
READECB  DC     F'0'                              Subtask 1 ECB posted at its control card reading completion.
         END    MAINTASK
```

Figure 51. Example 1 (Part 4 of 4)

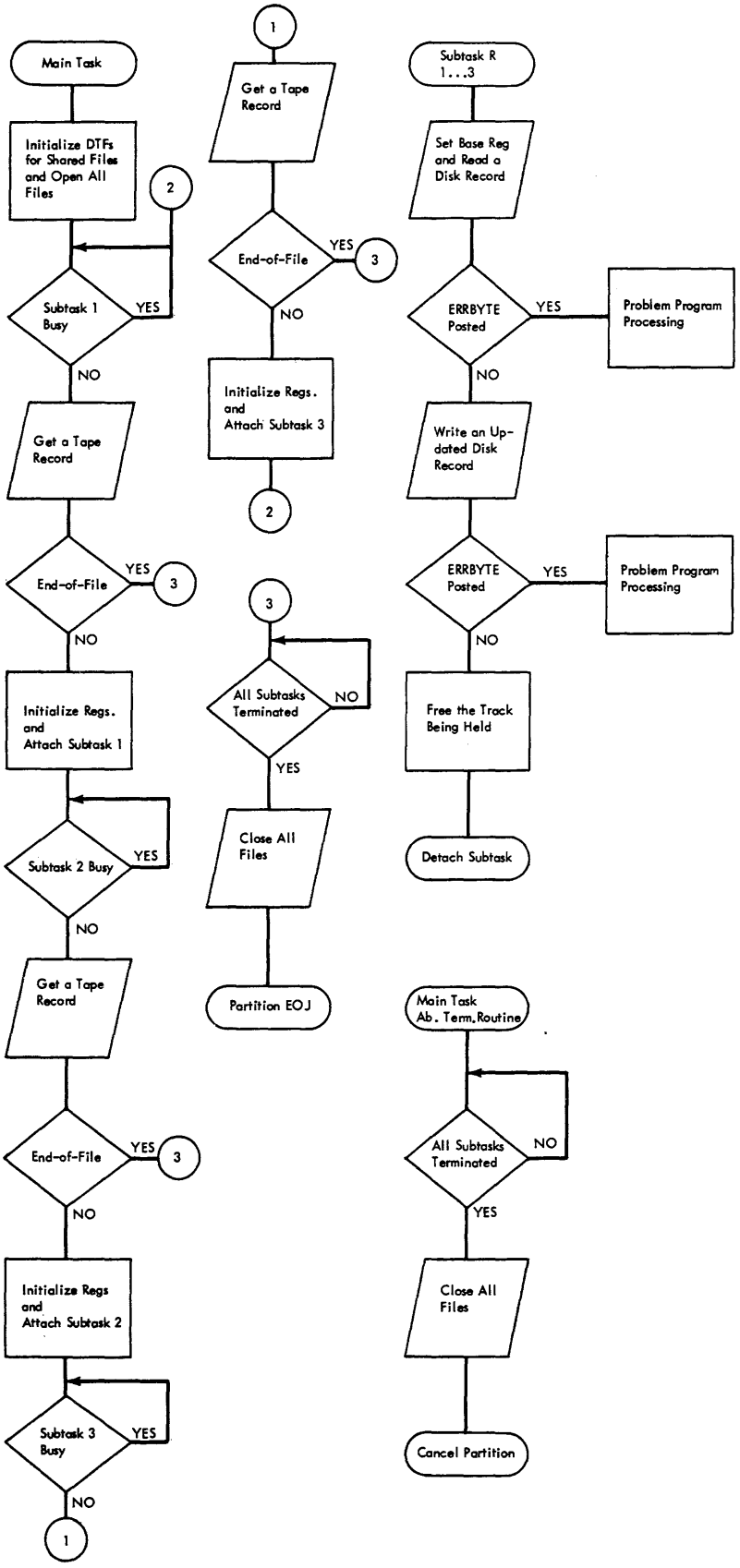
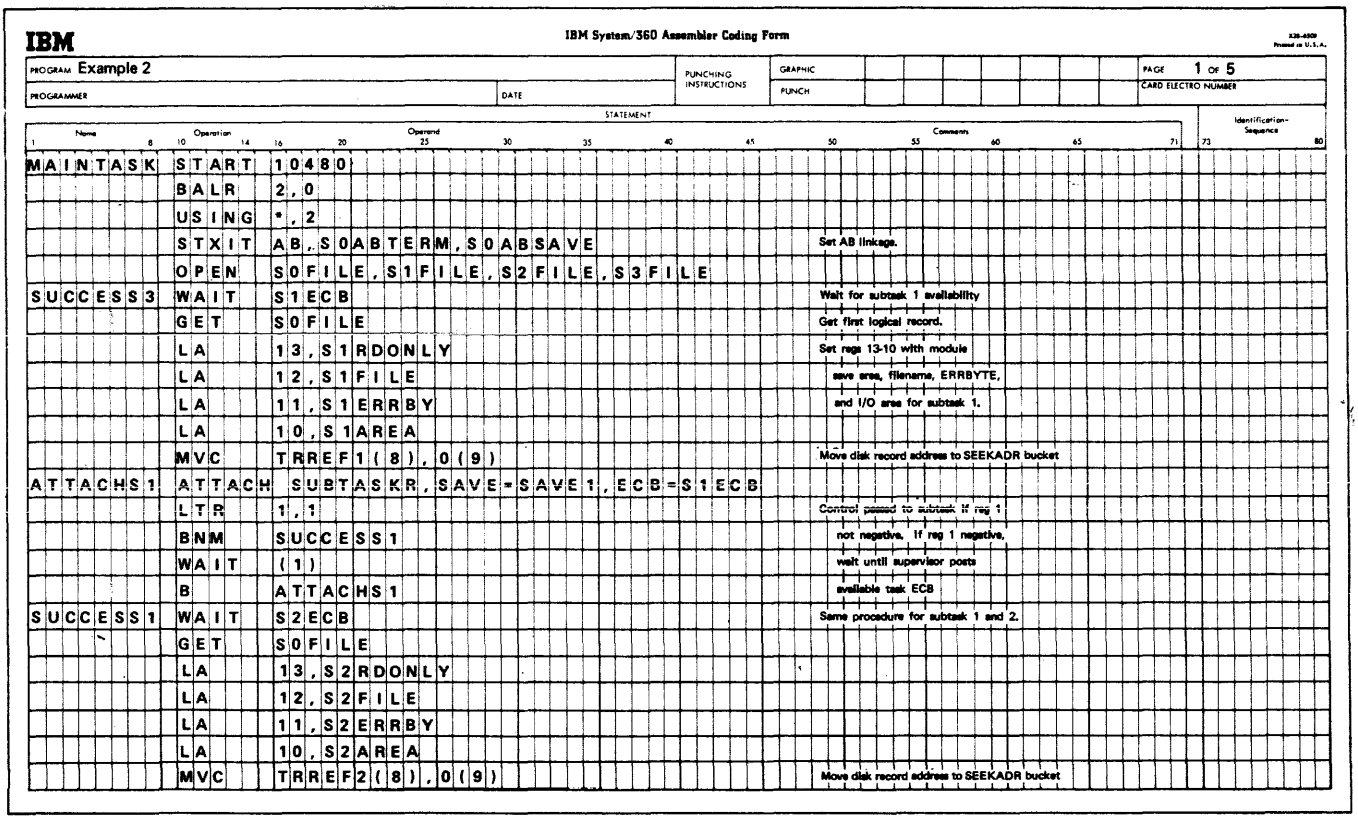


Figure 52. Flowchart for Example 2



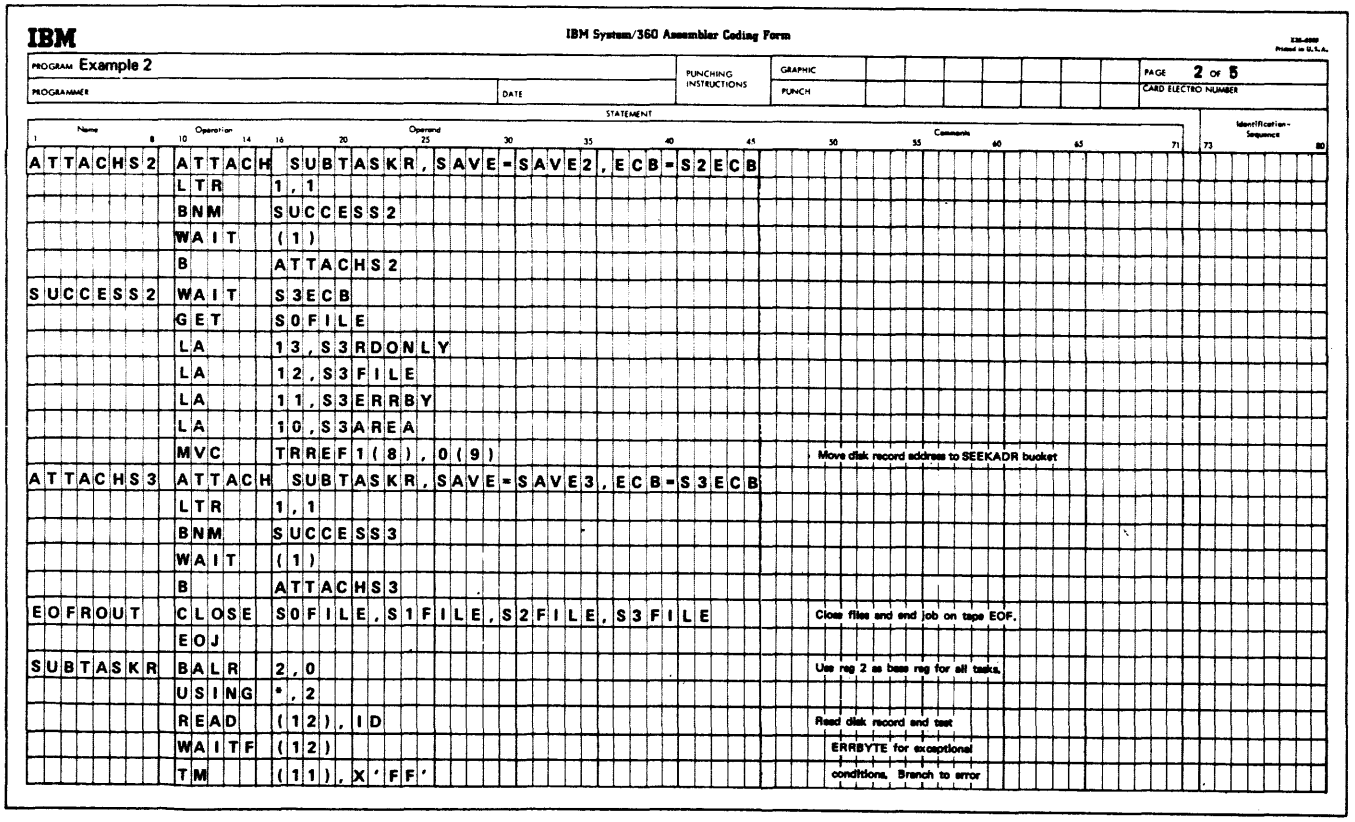
IBM System/360 Assembler Coding Form

PROGRAM Example 2 | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE 1 OF 5
PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT | Identification-Sequence

Name	Operation	Operand	Comments
MAINTASK	START	10480	
	BALR	2,0	
	USING	*,2	
	STXIT	AB,S0ABTERM,S0ABSAVE	Set AB linkage.
	OPEN	S0FILE,S1FILE,S2FILE,S3FILE	
SUCCESS3	WAIT	S1ECB	Wait for subtask 1 availability
	GET	S0FILE	Get first logical record.
	LA	13,S1RDONLY	Set regs 13-10 with module
	LA	12,S1FILE	save area, filename, ERRBYTE,
	LA	11,S1ERRBY	and I/O area for subtask 1.
	LA	10,S1AREA	
	MVC	TRREF1(8),0(9)	Move disk record address to SEEKADR bucket
ATTACHS1	ATTACH	SUBTASKR,SAVE=SAVE1,ECB=S1ECB	
	LTR	1,1	Control passed to subtask if reg 1
	BNM	SUCCESS1	not negative. If reg 1 negative,
	WAIT	(1)	wait until supervisor posts
	B	ATTACHS1	available task ECB
SUCCESS1	WAIT	S2ECB	Same procedure for subtask 1 and 2.
	GET	S0FILE	
	LA	13,S2RDONLY	
	LA	12,S2FILE	
	LA	11,S2ERRBY	
	LA	10,S2AREA	
	MVC	TRREF2(8),0(9)	Move disk record address to SEEKADR bucket

Figure 53. Example 2 (Part 1 of 5)



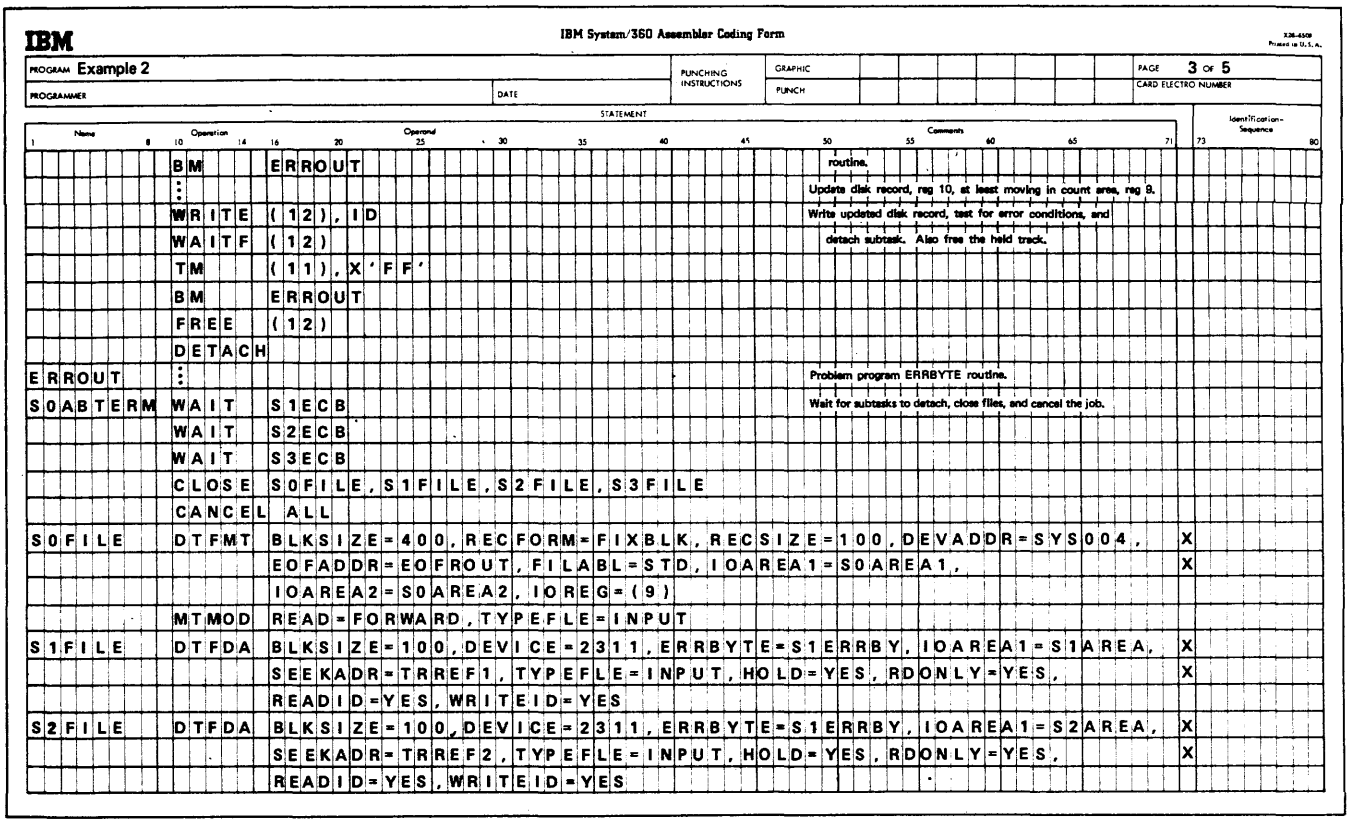
IBM System/360 Assembler Coding Form

PROGRAM Example 2 | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE 2 OF 5
PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT | Identification-Sequence

Name	Operation	Operand	Comments
ATTACHS2	ATTACH	SUBTASKR,SAVE=SAVE2,ECB=S2ECB	
	LTR	1,1	
	BNM	SUCCESS2	
	WAIT	(1)	
	B	ATTACHS2	
SUCCESS2	WAIT	S3ECB	
	GET	S0FILE	
	LA	13,S3RDONLY	
	LA	12,S3FILE	
	LA	11,S3ERRBY	
	LA	10,S3AREA	
	MVC	TRREF1(8),0(9)	Move disk record address to SEEKADR bucket
ATTACHS3	ATTACH	SUBTASKR,SAVE=SAVE3,ECB=S3ECB	
	LTR	1,1	
	BNM	SUCCESS3	
	WAIT	(1)	
	B	ATTACHS3	
EOFROUT	CLOSE	S0FILE,S1FILE,S2FILE,S3FILE	Close files and end job on tape EOF.
	EOJ		
SUBTASKR	BALR	2,0	Use reg 2 as base reg for all tasks.
	USING	*,2	
	READ	(12),ID	Read disk record and test
	WAITF	(12)	ERRBYTE for exceptional
	TM	(11),X'FF'	conditions. Branch to error

Figure 53. Example 2 (Part 2 of 5)



IBM System/360 Assembler Coding Form

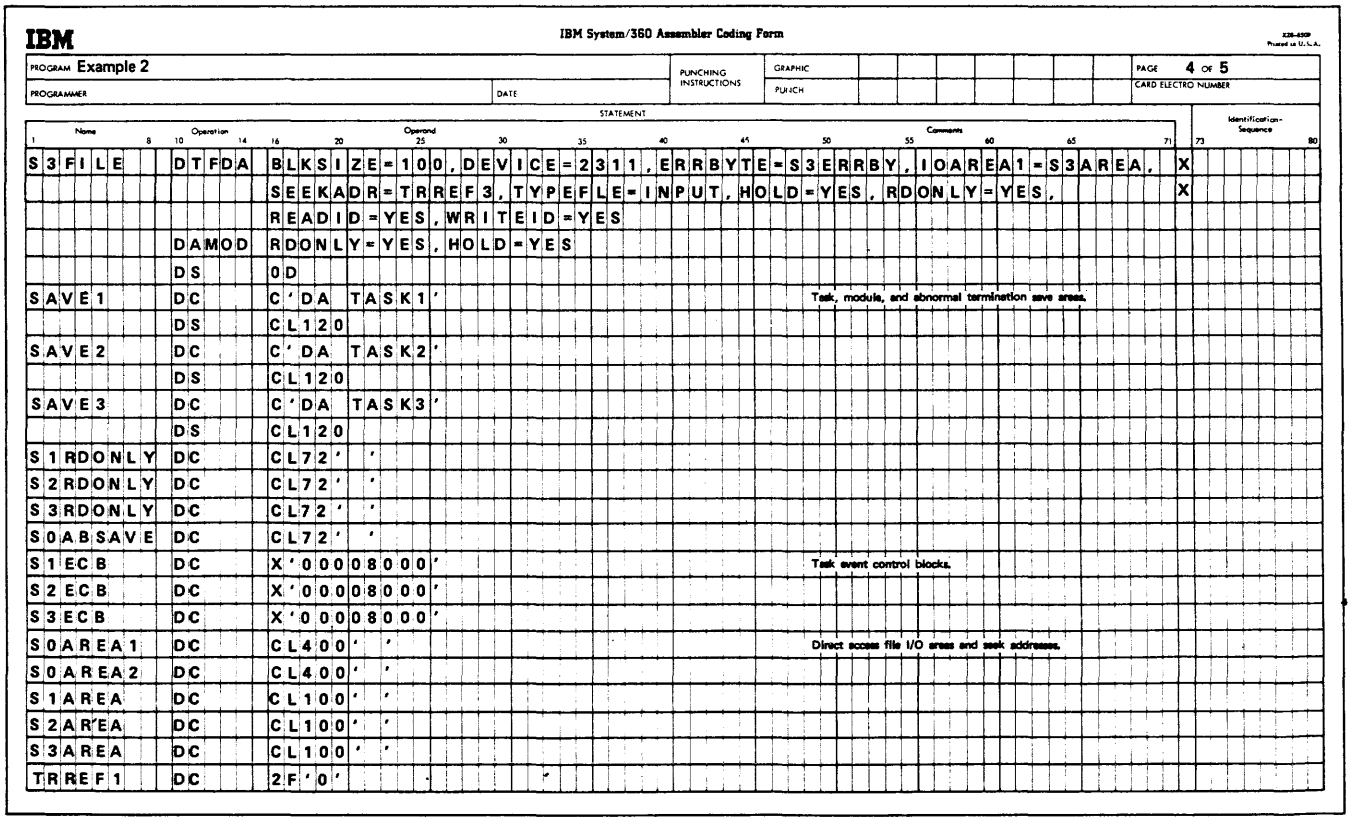
PROGRAM Example 2 | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE 3 OF 5

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
	BM	ERROUT	routine.	
	:			
	WRITE	(12),ID	Update disk record, reg 10, at least moving in count area, reg 9.	
	WAITF	(12)	Write updated disk record, test for error conditions, and	
	TM	(11),X'FF'	detach subtask. Also free the held track.	
	BM	ERROUT		
	FREE	(12)		
	DETACH			
ERROUT	:		Problem program ERRBYTE routine.	
S0ABTERM	WAIT	S1ECB	Wait for subtasks to detach, close files, and cancel the job.	
	WAIT	S2ECB		
	WAIT	S3ECB		
	CLOSE	S0FILE,S1FILE,S2FILE,S3FILE		
	CANCEL	ALL		
S0FILE	DTFMT	BLKSIZE=400,RECFORM=FIXBLK,RECSIZE=100,DEVADDR=SYS004,		X
		EOFADDR=EOFROUT,FILABL=STD,IOAREA1=S0AREA1,		X
		IOAREA2=S0AREA2,IOREG=(9)		
	MTMOD	READ=FORWARD,TYPEFLE=INPUT		
S1FILE	DTFDA	BLKSIZE=100,DEVICE=2311,ERRBYTE=S1ERRBY,IOAREA1=S1AREA,		X
		SEEKADR=TRREF1,TYPEFLE=INPUT,HOLD=YES,RDONLY=YES,		X
		READID=YES,WRITEID=YES		
S2FILE	DTFDA	BLKSIZE=100,DEVICE=2311,ERRBYTE=S1ERRBY,IOAREA1=S2AREA,		X
		SEEKADR=TRREF2,TYPEFLE=INPUT,HOLD=YES,RDONLY=YES,		X
		READID=YES,WRITEID=YES		

Figure 53. Example 2 (Part 3 of 5)



IBM System/360 Assembler Coding Form

PROGRAM Example 2 | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE 4 OF 5

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

STATEMENT

Name	Operation	Operand	Comments	Identification-Sequence
S3FILE	DTFDA	BLKSIZE=100,DEVICE=2311,ERRBYTE=S3ERRBY,IOAREA1=S3AREA,		X
		SEEKADR=TRREF3,TYPEFLE=INPUT,HOLD=YES,RDONLY=YES,		X
		READID=YES,WRITEID=YES		
	DAMOD	RDONLY=YES,HOLD=YES		
	DS	0D		
SAVE1	DC	C'DA TASK1'	Task, module, and abnormal termination save areas.	
	DS	CL120		
SAVE2	DC	C'DA TASK2'		
	DS	CL120		
SAVE3	DC	C'DA TASK3'		
	DS	CL120		
S1RDONLY	DC	CL72' '		
S2RDONLY	DC	CL72' '		
S3RDONLY	DC	CL72' '		
S0ABSAVE	DC	CL72' '		
S1ECB	DC	X'00008000'	Task event control blocks.	
S2ECB	DC	X'00008000'		
S3ECB	DC	X'00008000'		
S0AREA1	DC	CL400' '	Direct access file I/O areas and seek addresses.	
S0AREA2	DC	CL400' '		
S1AREA	DC	CL100' '		
S2AREA	DC	CL100' '		
S3AREA	DC	CL100' '		
TRREF1	DC	2F'0'		

Figure 53. Example 2 (Part 4 of 5)

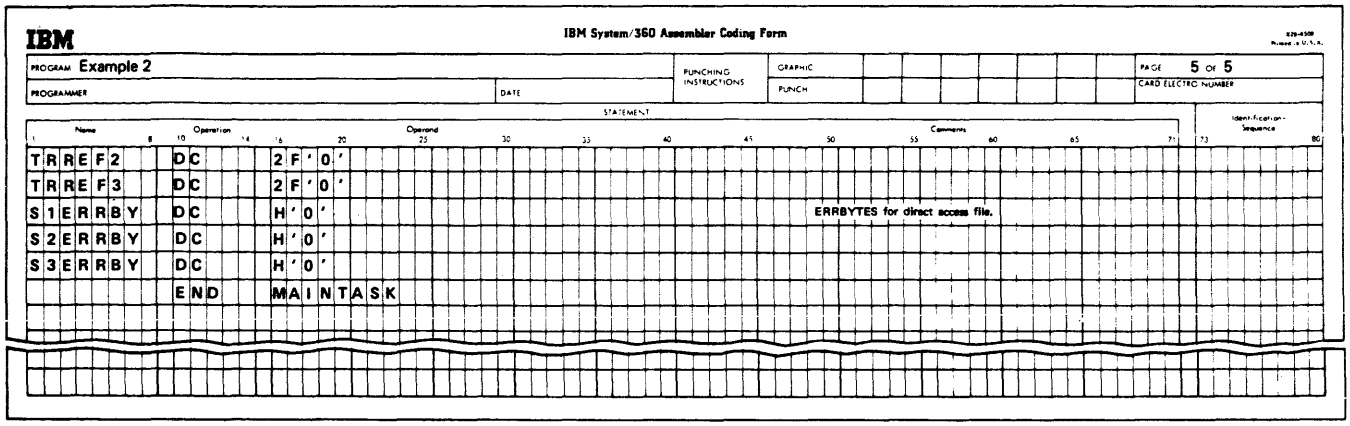


Figure 53. Example 2 (Part 5 of 5)

Records can be transferred to or from an input/output device by issuing physical IOCS macro instructions. These instructions relate directly to the physical IOCS routines and are distinct from logical IOCS routines. See Physical IOCS vs Logical IOCS.

The user must provide any of the functions that are required for a problem program such as blocking or deblocking records, performing programmed wrong-length record checks, testing (the CCB) for certain errors, switching I/O areas when two areas are used, and setting up channel command words (CCW). He must also recognize and bypass checkpoint records if they are interspersed with data records on an input tape.

Physical IOCS routines control the transfer of data to or from the external device. These routines are:

- Start I/O
- I/O Interrupt
- Channel Scheduler
- Device Error

Thus, physical IOCS macro instructions provide the user with the capability of obtaining data and performing nondata operations in I/O devices, with exactly the CCWs that he requests. For example, if he is handling only physical records, he does not need the logical IOCS routines for blocking and deblocking logical records.

Three macro instructions are available to the programmer for direct communication with physical IOCS: CCB (command control block), EXCP (execute channel program), and WAIT. Whenever physical IOCS macro instructions are used, the programmer must construct the channel command words (CCW) for input/output operations. He uses the assembler instruction CCW statement to do this. The CCW instruction is described in Principles of Operation which is listed on the front cover.

Macros normally used with files that are processed by logical IOCS are necessary when standard DASD or magnetic tape labels are processed, or when DASD file protect is present. The DTFPH, OPEN(R), CLOSE(R), LBRET, FEOV, and SEOV macros can be used in this processing. See DTFPH Macro. The OPEN and the DTFPH macros are also necessary when a 2311 or 2314 is used as a checkpoint file.

CCB MACRO

Name	Op	Operand
blockname	CCB	SYSnnn,command-list-name [,X'nnnn'][,sense address]

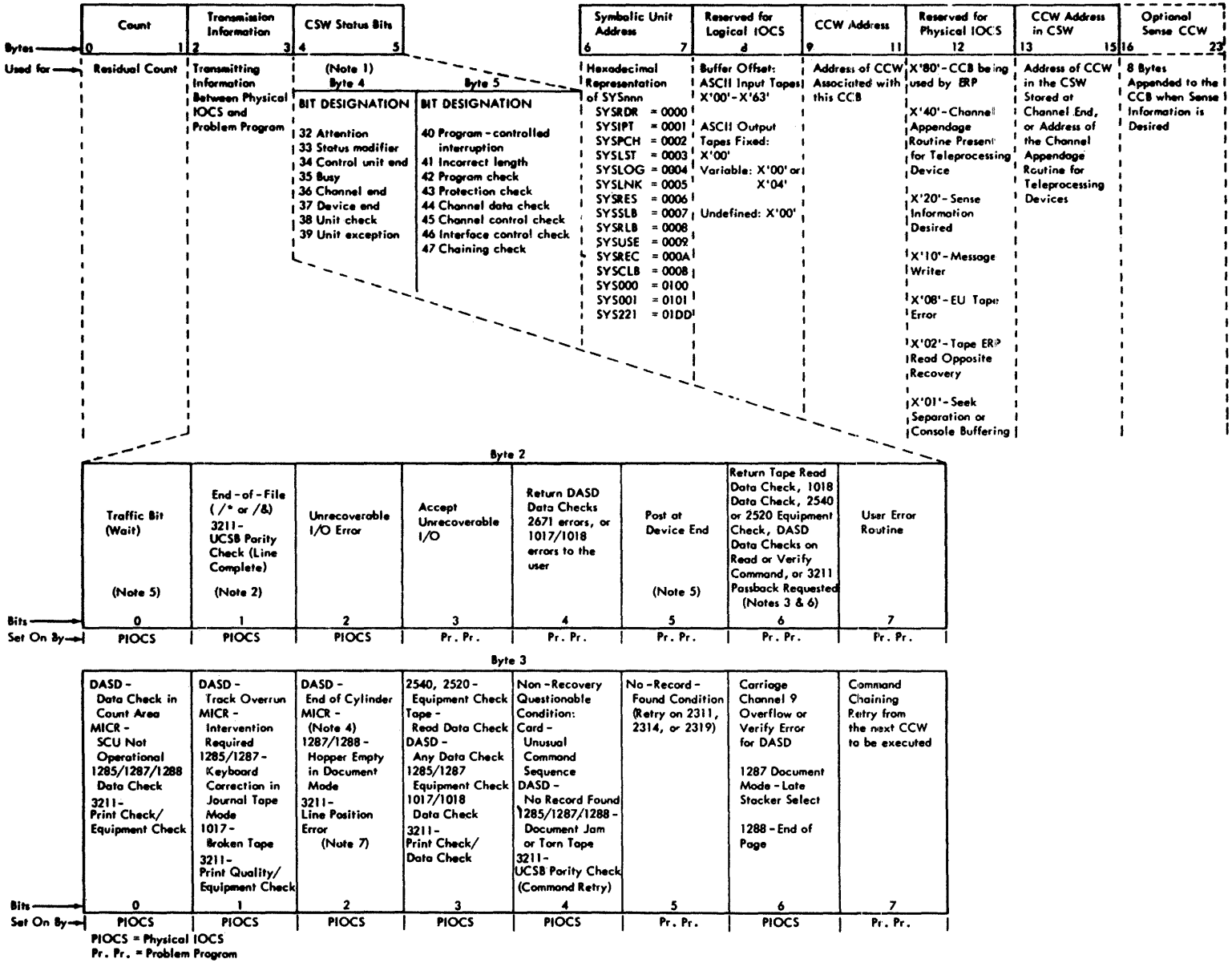
A CCB (command control block) macro instruction must be specified in the problem program for each I/O device that is controlled by physical IOCS macro instructions. The first 16 bytes of all the generated DTF tables, except DTFOR and DTFSR (optical reader), contain the CCB. For optical reader macros, the CCB begins at filename+24 which includes the DTFPH. This block (see Figure 54) is necessary to communicate information to physical IOCS so that it can perform desired operations (for example, notify the problem program of printer channel 9). The command control block also receives status information after an operation and makes this available to the problem program. The user should ensure proper boundary alignment of the CCB if this is necessary for his program.

Blockname: The CCB instruction must be labeled (blockname) with a symbolic name. This name can be used as the operand in the EXCP and WAIT macro instructions that refer to the command control block.

SYSnnn: Two operands are required in this CCB macro instruction. The first operand specifies the symbolic unit (SYSnnn) for the actual I/O unit with which this control block is associated. A list of symbolic units applying to CCB can be found in the Symbolic Unit Addresses section. The actual I/O unit can be assigned to the symbolic unit by a job control ASSGN statement.

Command-list-name: The second operand (command-list-name) specifies the symbolic name of the first CCW used with a CCB. This name must be the same as the name specified in the assembler CCW statement that constructs the channel command word.

Figure 54. Command Control Block (CCB)



PIOCS = Physical IOCS
Pr. Pr. = Problem Program

Note 1. Bytes 4 and 5 contain the status bytes of the Channel Status Word (Bits 32-47). If byte 2, bit 5 is on and device end results as a separate interrupt, device end will be ORed into CCB byte 4.
Note 2. Indicates /* or /& statement encountered on SYSRDR or SYSIPT. Byte 4, bit 7 (unit exception) is also on.
Note 3. DASD data checks on count not returned.
Note 4. For 1255/1259/1270/1275/1412/1419, disengage. For 1275/1419D, I/O Error in external interrupt routine (channel data check or busout check).
Note 5. The traffic bit (Byte 2, bit 0) is normally set on at channel end to signify that the I/O was completed. If byte 2, bit 5 has been set on, the traffic bit and bits 2 and 6 in byte 3 will be set on at device end. Also see Note 1.
Note 6. 1018 ERP does not support the Error Correction Function.
Note 7. This error occurs as an equipment check, data check, or FCB parity check.

X'nnnn': A hexadecimal value sets the CCB user option bits. Column 5 of Figure 55 gives the value used to set a user option bit ON. If more than one bit must be set, the sum of the values is used. For example, to set user option bits 3, 5, and 6 of byte 2 ON, X'1600' is used.

(X'1600'=X'1000' + X'0400' + X'0200')

It is possible for the macro to set ON any of the bits in bytes 2 or 3, but normally, the user need not be concerned with setting the remaining bits ON.

Sense Address: This operand, when supplied, indicates user error recovery (see Figure 55 byte 2, bit 7) and generates a CCW for reading sense information as the last field of the CCB. The name field (sense address) of the area that the user supplies must have a length attribute assigned of at least one byte. Physical IOCS uses this length attribute in the CCW to determine the number of bytes of sense information the user desires.

CCB Format

From the specifications in this CCB instruction, the macro sets up a 16-byte or 24-byte field (Figure 54) as follows:

Bytes	Contents
0-1	After a record is transferred, IOCS places the residual count from the CSW in these two bytes. By subtracting the residual count from the original count in the CCW, the problem program can determine the length of the record that was transferred. This residual count is set at zero for negative values.
2-3	The next two bytes are for transmission of information between physical IOCS and the problem program. The problem program can test any bit in bytes 2 and 3, using the mask given in column 6 of Figure 55. More than one bit can be tested by the hexadecimal sum of the test values. All bits are set at 0 (OFF) when the problem program is assembled unless the third parameter is specified. If the third parameter is specified, it is assembled into these two bytes. The user may turn on bits 5 and 7 in byte 3 and bits 3 through 7 in byte 2. During execution, each bit may be set at 1 (ON) by the problem program or by a condition detected by physical IOCS. Any bits that can be turned ON by physical IOCS during program execution are reset to zero by PIOCS the next time an EXCP macro using the same CCB is executed.

Figure 55 shows the condition indicated by the setting of each bit.

4-5 These two bytes are the status bytes of the CSW. If device end posting is requested (byte 2, bit 5), device end status is ORed in. Byte 4 is set to X'00' at EXCP time.

> Note: For nonteleprocessing devices, a program-controlled interruption (PCI) is ignored by the Channel Scheduler.

6-7 These two bytes are a hexadecimal representation of the symbolic unit for the I/O devices, as specified in the first operand of this CCB.

8-11 These four bytes contain the address of the CCW (or first address of a chain of CCWs) associated with this CCB and specified symbolically in the second operand.

12 This byte must not be modified by the user.

13-15 These bytes contain the address of the CCW in the CSW stored at channel end interrupt for this I/O operation.

> Note: Bytes 13-15 contain the address of the channel appendage routine when bit X'40' is set in byte 12.

16-23 These bytes are allotted only when the sense address operand is supplied in the CCB macro. They contain the CCW for returning sense information to the problem program.

EXCP Macro

Name	Operation	Operand
[name]	EXCP	{blockname / (1)}

The EXCP (execute channel program) macro instruction requests physical IOCS to start an input/output operation for a particular I/O device. The blockname of the CCB established for the device is the only operand required in this instruction. Blockname can be specified as a symbol or in register notation.

Physical IOCS determines the device from the command control block specified by blockname, and places the command control block (CCB) in a queue of such CCBs for this device. If the channel and device are available, the channel program is started and program control is returned to the problem program. I/O interruptions are used to process I/O completion and to start I/O for requests if the channel or device was busy at EXCP time.

WAIT Macro

Name	Operation	Operand
[name]	WAIT	{blockname / (1)}

This macro instruction is issued whenever the program requires that an I/O operation (started by an EXCP instruction), be completed before execution of the problem program continues. For example, transferring data (a physical record) to main storage must be completed before data can be added or moved to another area of main storage, or otherwise processed. When the WAIT instruction is executed in a batched job environment, processing is suspended until the traffic bit (byte 2, bit 0) of the related CCB is turned ON. Then, programming automatically continues and the data can be processed. In a multiprogramming environment, the supervisor gives control to another program until the traffic bit is set ON.

The blockname (as a symbol or in register notation) of the CCB established for the I/O device is the only operand required in this instruction. This is also the same name as that specified in the EXCP instruction for the device.

Physical IOCS Considerations

ALTERNATE TAPE SWITCHING

Alternate tape drives cannot be used on input processed by PIOCS. On output, automatic alternate switching can be accomplished by the DTFPH and FEOV macro instructions. FEOV writes the standard trailer labels, and any user-trailer labels (if DTFPH LABADDR is specified). When the new volume is mounted and ready for writing, IOCS writes the standard header labels, and the user-standard header labels, if any.

Byte	Bit	Condition Indicated		On Values for Third Operand in CCB Macro	Mask for Test Under Mask Instruction
		1 (ON)	0 (OFF)		
2	0 Traffic Bit (WAIT).	I/O Completed. Normally set at Channel End. Set at Device End if bit 5 is ON.	I/O requested and not completed.		X'80'
	1 End of File on System Input.	/* or /& on SYSRDR or SYSIPT. Byte 4, Unit Exception Bit is also ON.			X'40'
	3211 UCSB Parity Check (line complete)	Yes	No		
	2 Unrecoverable I/O Error	I/O error passed back due to program option or operator option.	No program or operator option error was passed back.		X'20'
	3 [1] Accept Unrecoverable I/O Error (Bit 2 is ON)	Return to user after physical IOCS attempts to correct I/O error.[2]	Operator Option: Dependent on the Error	X'1000'	X'10'
	4 [1] 2671 data check.	Operator Options: Ignore, Retry, or Cancel.	Operator Option: Retry or Cancel.	X'0800'	X'08'
	1017/1018 data checks.	Ignore or Cancel.	Cancel.		
	Return any DASD data checks.	Return to user.			
	5 [1] Post at Device End.	Device End condition is posted; that is, byte 2, bit 0 and byte 3, bits 2 and 6 set at Device End. Also byte 4, bit 5 is set.	Device End conditions are not posted. Traffic bit is set at Channel End.	X'0400'	X'04'
	6 [1] Return: Uncorrectable tape read data check (2400-series, 3420, or 2495); 1018 data check; 2540 or 2520 punch equipment check; DASD read or verify data check; 3211 passback requested. (Data checks on count not returned.)	Return to user after physical IOCS attempts to correct 3211, tape, or DASD error. Return to user when 1018 data check.[4]	Operator Option: Ignore or Cancel for tapes, punches, or paper tape punch (1018). Retry or cancel for DASD.	X'0200'	X'02'
	7 [1] User Error Routine	User handles error recovery.[3]	A physical IOCS error routine is used unless the CCB sense address operand is specified. The latter requires user error recovery.	X'0100'	X'01'
3	0 Data check in DASD count Field.	Yes - Byte 3, bit 3 is OFF; Byte 2, bit 2 is ON.	No		X'80'
	Data check - 1285, 1287, or 1288.	Yes	No		
	MICR - SCU not operational.	Yes	No		
	3211 Print Check (equipment check).	Yes	No		
	1 DASD Track overrun.	Yes	No		X'40'
	1017 broken tape.	Yes	No		
	Keyboard correction 1285 or 1287 in Journal Tape Mode.	Yes	No		
	3211 print quality error (equipment check).	Yes	No		
	MICR intervention required.	Yes	No		
	2 End of DASD Cylinder.	Yes	No		X'20'
	Hopper Empty 1287/1288 Document Mode.	Yes	No		
	MICR - 1255/1259/1270/1275/1412/1419, disengage.	Document feeding stopped.	No		
	- 1275/1419D, I/O error in external interrupt routine.	Channel data check or Busout check.			
	3211 line position error.[5]	Yes	No		

Figure 55. Conditions Indicated by CCB Bytes 2 and 3 (Part 1 of 2)

Byte	Bit	Condition Indicated		On Values for Third Operand in CCB Macro	Mask for Test Under Mask Instruction
		1 (ON)	0 (OFF)		
3	3 Tape read data check (2400-series or 2495); 2540 or 2520 punch equipment check; or any DASD data check.	Operation was unsuccessful. Byte 2, bit 2 is also ON. Byte 3, bit 0 is OFF.	No		X'10'
	1017/1018 data check.	Yes	No		
	1285, 1287, or 1288 equipment check.	Yes	No		
	3211 data check (print check).	Yes	No		
	4 Questionable Condition.	Card: Unusual command sequence (2540). DASD: No record found.			X'08'
	Nonrecovery	1285/1287/1288: Document jam or torn tape.			
	UCSB parity check (command retry).	Yes	No		
	5[1] No record found condition	Retry command if no record found condition occurs (disk).	Set the questionable condition bit ON and return to user.	X'0004'	X'04'
	6 Verify error for DASD or Carriage Channel 9 overflow	Yes. (Set ON when Channel 9 is reached only if Byte 2, bit 5 is ON).	No		X'02'
	1287 document mode - late stacker select.	Yes	No		
	1288 End-of-Page (EOP).	Yes	No		
	7[1] Command Chain Retry	Retry begins at last CCW executed.	Retry begins at first CCW of channel program.	X'0001'	X'01'

[1] User Option Bits. Set in CCB macro. Physical IOCS sets the other bits OFF at EXCP time and ON when the condition specified occurs.
[2] I/O program check, command reject, or tape equipment check always terminates the program.
[3] For System/360, the user must handle all error or exceptional conditions except Channel Control Check, Interface Control Check, I/O Program Check, and I/O Protection Check. For System/370, the user may handle Channel Control Checks and Interface Control Checks. The occurrence of a channel data check, unit check, or chaining check causes a byte 2, bit X'20' of the CCB to turn on, and completion posting and dequeuing to occur. I/O program and protection checks always cause program termination. Incorrect length and unit exception are treated as normal conditions (posted with completion). Also, the user must request device end posting (CCB byte 2, bit X'04') in order to obtain errors after channel end.
[4] Error correction feature for 1018 is not supported by physical IOCS. When a 1018 data check occurs and CCB byte 2, bit X'02' is on, control returns directly to the user with CCB byte 3, bit X'10' turned on.
[5] A line position error can occur as a result of an equipment check, data check, or FCB parity check.

Figure 55. Conditions Indicated by CCB Bytes 2 and 3 (Part 2 of 2)

BYPASSING EMBEDDED CHECKPOINT RECORDS ON TAPE

The checkpoint information saved is written as a set of magnetic tape records: a 20-byte header record, as many core-image records as required to save the necessary parts of core, and a 20-byte trailer record identical to the header. The format of the header and trailer record is:

Bytes	Contents
0-11	/// CHKPT //
12-13	The number, in binary, of core image records following the header.
14-15	The total number, in unpacked decimal, of records following the header.
16-19	The serial number of the checkpoint.

If checkpoint sets are embedded in a file being read with physical IOCS, they must be recognized and bypassed. For any mode on an input tape, checkpoint sets may be identified by the first 12 bytes of the header or trailer records. When reading backwards, the checkpoint header occupies the 20 low-order bytes of the input area.

When bypassing checkpoint sets, three methods are possible:

1. Go into a read loop (forward or backward) until the checkpoint trailer (header if backward) is encountered.
2. Extract the count from bytes 12-13 of the header (or trailer if backwards), add 2 to this, and forward space (or

backspace) that number of records. Read commands could also be used.

3. Extract bytes 14-15 of the header (or trailer if backwards), pack and convert the field to binary, and forward space (or backspace) that number of records. Read commands could also be used.

When bypassing checkpoint sets on 7-track tapes in translate mode, only method 3 can be used and only forward space (or backspace) record commands (not reads) can be used. Reads would create data checks.

CCW ROUTINE CONSIDERATIONS

Printer-Keyboard Buffering

If the console buffering option is specified at system generation time and the printer-keyboard is assigned to SYSLOG, physical IOCS can increase throughput on each actual output record not exceeding 80 characters. This increase in throughput is the result of starting the output I/O command and returning to the problem program before the output is complete. Regardless of whether output records are buffered (queued on an I/O completion basis) or not, they are always printed in a first-in first-out (FIFO) order.

Console buffering is performed on output only when the following conditions are maintained.

1. The actual record to be written does not exceed 80 characters.
2. Data or command chaining is not performed.
3. The CCB associated with this operation does not indicate the acceptance of unrecoverable I/O errors, posting at device end, or user error routines (that is, CCB byte 2 bits 3, 5, and 7 posted respectively).
4. The CCB does not request sense information (CCB byte 12 bit 2).

Command Chaining Retry

If the user generates his system to support command chaining retry, he can use this option for his physical IOCS channel programs by setting CCB bit 7, byte 3 ON. If this bit is ON and an error involving retry occurs, the retry begins with the last CCW executed. If the bit is OFF, the entire channel program is reexecuted.

If a command chain is broken by a condition (such as wrong-length record or unit exception) that does not result in device error recovery by physical IOCS, the user can determine the address of the last CCW executed and, if necessary, restart at that point. To obtain the address of the last CCW executed, subtract 8 from the address stored in bytes 13-15 of the CCB.

When the command chaining retry bit is ON, the user must move the address of the first CCW in the channel program to bytes 9-11 of the CCB before each EXCP issued. This ensures that the correct address is there, because physical IOCS modifies this field when retrying after an I/O error and never restores it to the original value.

The command chaining retry bit should not be used to read multiple records from SYSIPT or SYSRDR. The bit should never be ON for DASD channel programs. If command chaining, or data chaining, is employed on an IBM 2495 Tape Cartridge Reader, the problem programmer must specify command chaining retry at system generation time and in the CCB.

> Note: A chain is broken by a 1403 printer after sensing channel 9 or 12. In such cases, the problem program must determine if the entire chain was printed.

Channel Command Word

The format of the Channel Command Word is:

Bits	Contents
0-7	Command code.
8-31	Data address.
32	The address portion of next CCW is used.
33	Command code and data address of next CCW is used.
34	Suppresses incorrect length.
35	Suppresses transfer of information to main storage.
36	Program control interrupt.
48-63	Byte count.

Data Chaining

When using data chaining, each CCW should contain the command code of the operation being executed to ensure proper I/O error recovery. If the CCWs were formed by IOCS, they contain this code automatically. Because recovery frequently depends on the command being executed, the command in the last CCW executed is often examined. See Command Chaining Retry.

DASD Channel Programs

The user must begin his DASD channel programs with a full seek (command code X'07') and if embedded seeks are used, they must also be full seeks.

If embedded seeks are used, a program can never run under DASD file protection (system generation option DASDFP) nor can it take full advantage of the seek separation feature (system generation option SKSEP). With DASD file protection, an embedded seek causes cancellation of the errant program (see DASD File Protection for more information).

The seek separation feature initiates a seek and separates it from the channel command chain. In this manner, the channel is available for other input or output operations on the same channel. This feature, however, applies only to the first seek in the channel command chain.

When executing a channel program (Figure 56), the supervisor sets up three commands in the channel program that it builds: a seek that is identical to the user's seek, a set file mask that prevents any other 07 seeks from being executed, and a transfer in channel (TIC) that transfers control to the command following the user's seek.

DTFPH Macro

When physical IOCS macro instructions (EXCP, WAIT, etc) are used in a program, DASD or tape files with standard labels need to be defined by DTFPH entries (DTF for a file handled by physical IOCS). DTFPH must also be used for a checkpoint file on a 2311, 2314, or 2319.

Checkpoint File on a 2311 or 2314: The following parameters can be used:

CCWADDR=name	optional
DEVADDR=SYSnnn	optional
DEVICE=2311 or 2314	required
LABADDR=name	optional
MOUNTED=SINGLE	required
TYPEFLE=OUTPUT	required
XTNTXIT=name	does not apply

If a DASD or tape file with standard volume and file labels is processed, a DTFPH header card and detail cards may be used (Figure 57). This indicates to IOCS that labels are to be read and checked (on input) or written (on output). The header card is punched with DTFPH in the operation field and the symbolic name of the file in the name field. The symbolic name may be seven characters long.

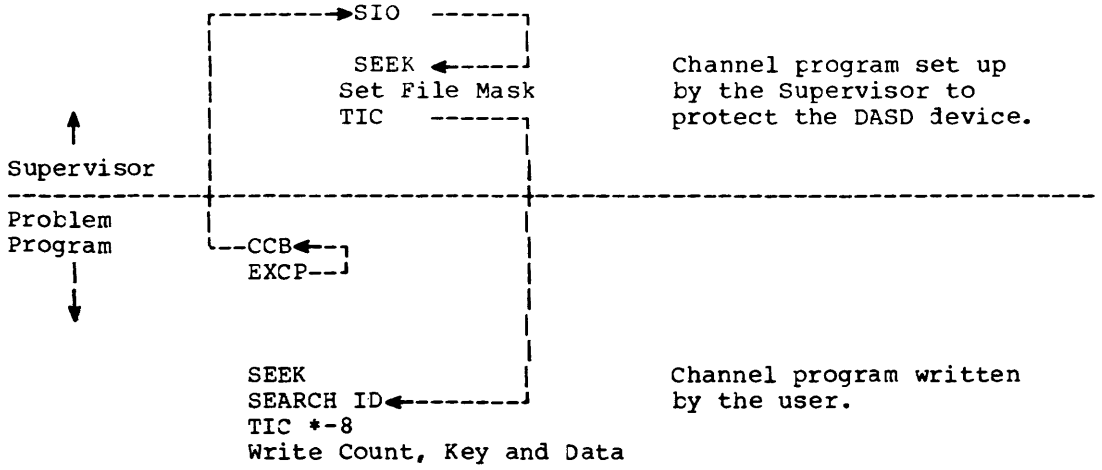
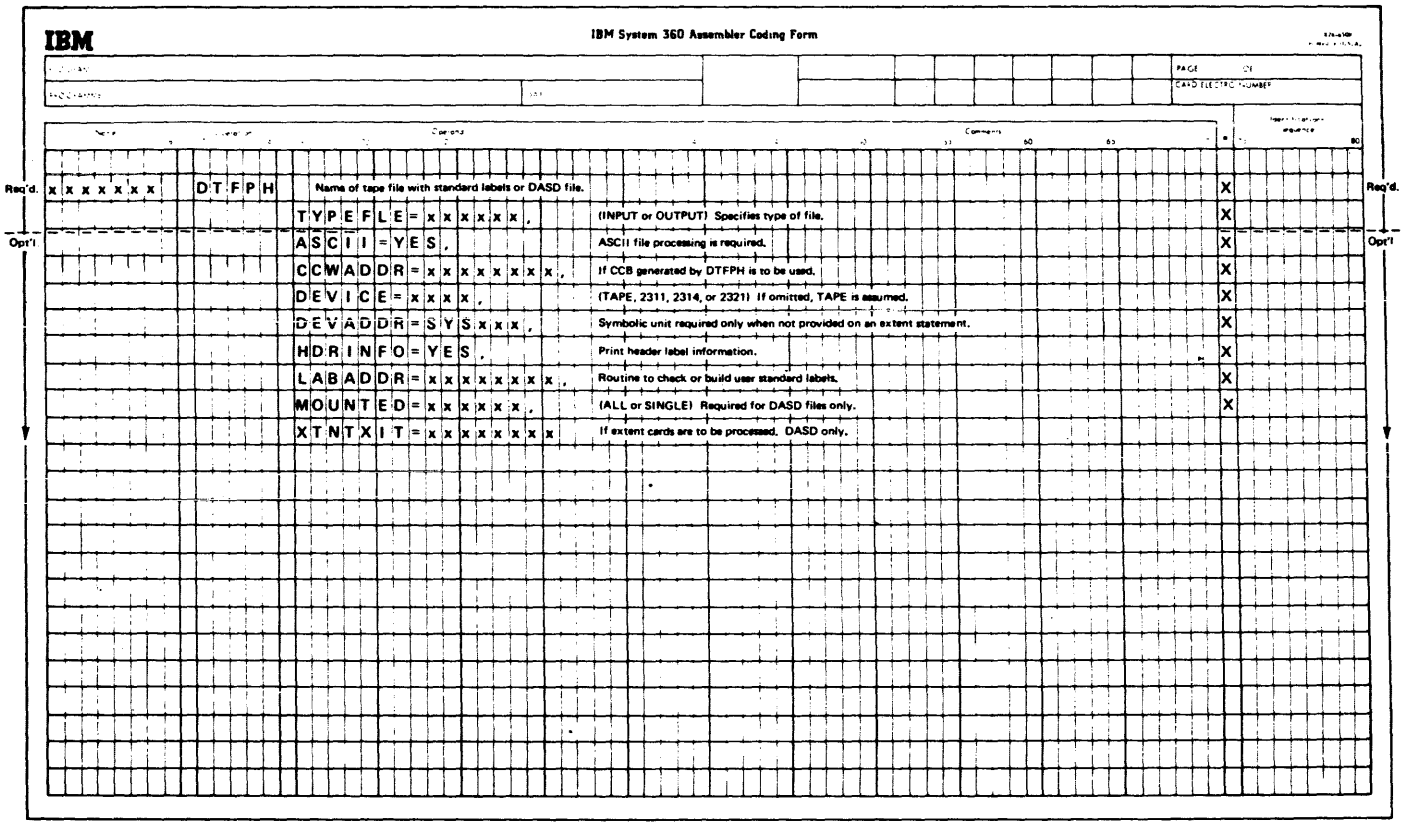


Figure 56. Channel Programming a File Protected DASD



* Header and each detail card, except the last one in each set, must have a continuation punch in column 72. Also, each detail card, except the last one, must contain a comma immediately after the operand. Space is allowed for the longest operand plus the comma. If a smaller operand is used, the comma should be moved over accordingly. In the last detail card of a set, the comma position must be blank.

† General registers 2-12, written in parentheses; for example: (12).

Figure 57. DTFPH Macro

```
|ASCII=YES                                |
```

This operand is required to process ASCII tape files. If this operand is omitted, EBCDIC file processing is assumed.

```
|CCWADDR=name                             |
```

This operand allows the user to use the CCB generated within the first 16 bytes of the DTFPH table. CCWADDR specifies the symbolic name of the first CCW used with the CCB generated within the DTFPH macro. This name must be the same as the name specified in the assembler CCW statement that constructs the channel command word.

If this parameter is omitted, the location counter value of the CCB-CCW table address constant is substituted for the CCW address.

```
|DEVICE={TAPE|2311|2314|2321}             |
```

If the file is contained on DASD, enter the proper identification: 2311, 2314, or 2321; TAPE applies to any 3420 or 2400-series tape unit, and is the only valid entry in this operand for ASCII files.

Note: Specify 2314 for 2319.

```
|DEVADDR=SYSxxx                           |
```

This operand must specify the symbolic unit (SYSxxx) associated with the file if an extent statement symbolic unit is not provided. If an extent statement symbolic unit is provided, its specification overrides a DEVADDR specification. This specification, or symbolic unit, represents an actual I/O address, and is used in the job control ASSGN statement to assign the actual I/O device address to this file.

For a list of symbolic units applying to DTFPH, see Symbolic Unit Addresses. The

only symbolic unit in that section that is not applicable is SYSLOG.

If SYSLST or SYSPCH are used as output tape units and alternate tape switching is desired upon detecting a reflective spot, the SEOV macro instruction must be used. (See SEOV Macro.) When processing ASCII tape files, the only valid specification is a programmer logical unit (that is SYSnnn).

```
HDRINFO=YES
```

This operand, causes IOCS to print standard header label information (fields 3-10) on SYSLOG each time a standard label file is opened. Likewise, the filename, symbolic unit, and device address are printed each time an end-of-volume condition is detected. If HDRINFO=YES is omitted, no header or end-of-volume information is printed.

```
LABADDR=name
```

The user may require one or more DASD or tape labels in addition to the standard file labels. If so, he must include his own routine to check (on input) or build (on output) the user label(s). The symbolic name of the user's routine is specified in this entry. IOCS branches to this routine after the standard label is processed.

LABADDR may be included to specify a user routine for user header or trailer labels as follows:

- DASD input or output file: header labels only
- Tape input or output file: header and trailer labels

Thus, if LABADDR is specified for the file, user header labels can be processed for an input/output DASD or tape file, and user trailer labels can be built for a tape output file. Physical IOCS reads input labels and makes them available to the user for checking, and writes output labels after they are built. This is similar to the functions performed by logical IOCS. For a complete discussion of the LABADDR routine, see Label Processing.

If physical IOCS macros are used for a tape file, an OPEN instruction must be issued for the new volume. This causes IOCS to check the HDR1 label and provides for user checking of user standard labels, if any.

When physical IOCS macros are used and DTFPH is specified for standard tape label processing, FEOV may not be issued for an input file.

```
MOUNTED={ALL|SINGLE}
```

This entry must be included for a DASD file to specify how many extents (areas) for the file are available for processing when the file is initially opened. The entry must not be included for a tape file. One of the following specifications is entered after the = sign:

ALL

Is specified if all extents are available for processing. When a file is opened, IOCS checks all labels on each disk pack and makes available all extents specified by the user's control statements. Only one OPEN is required for the file. ALL should be specified whenever the user plans to process records in a manner similar to the direct access method. In any case, the user must supply a LBLTYP statement.

After an OPEN is performed, the user must be aware that the symbolic unit address of the first volume containing the file is in bytes 30 and 31 of the DTFPH table rather than in the CCB. Before executing any EXCPs the user must place the symbolic address in bytes 6 and 7 of the CCB.

SINGLE

Is specified if only the first extent on the first volume is available for processing. SINGLE should be specified when the user plans to process records in sequential order. IOCS checks the labels on the first pack and makes the first extent specified by the user's control cards available for processing. The user must keep track of the extents and issue a subsequent OPEN whenever another extent is required for processing. The user will find the information in the DTFPH table helpful in keeping track of the extents:

DTFPH table (referenced by filename)

Bytes	Contents
0-15	CCB (Symbolic unit has been initialized in the CCB)
54-57	Extent Upper Limit (cchh)
58-59	Seek Address (bb-bin or cell number, if a one-celled device such as disk it must be zero.
60-63	Extent Lower Limit (cchh).

On each OPEN after the first, IOCS makes available the next extent specified by the control cards. When the user issues a CLOSE for an output file, the volume on which he is currently writing records is indicated, in the file label, as the last volume for the file.

```
TYPEFLE={INPUT|OUTPUT}
```

This entry must be included to specify the type of file (input or output). One specification or the other is entered immediately after the = sign.

```
XTNTXIT=name
```

This entry is included if the programmer wants to process label extent information. It specifies the symbolic name of the user's extent routine. The DTFPH entry MOUNTED=ALL must also be specified for the file.

Whenever XTNTXIT is included, IOCS branches to the user's routine during the initial OPEN for the file. It branches after each specified extent is completely checked and after conflicts, if any, resolved.

Upon entry to the user's routine, IOCS stores the address (in register 1) of a 14-byte area from which the user can retrieve label extent information (in binary form). This area contains:

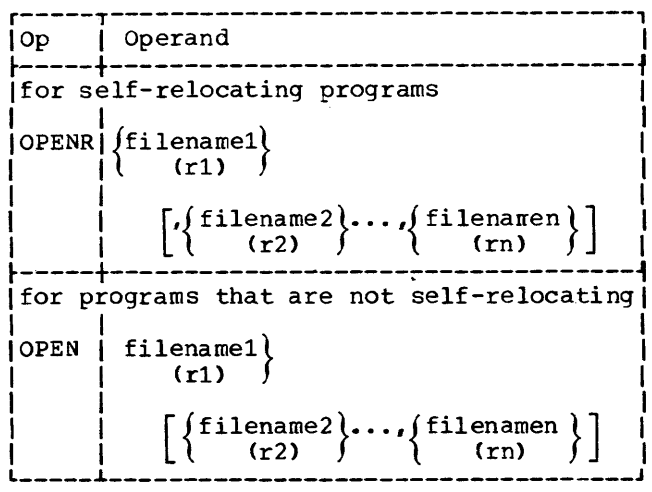
Bytes	Contents
0	Extent type code (as specified in the extent statement). See Extent Type Hexadecimal Codes.
1	Extent sequence number
2-5	Lower limit of the extent (cchh)
6-9	Upper limit of the extent (cchh)
10-11	Symbolic unit (see Figure 54)
12	Old bin (cell) number. If a one-celled device such as disk, byte 12 contains zero.
13	Present bin number of the extent (b2)

The user returns to IOCS by using the LBRET macro instruction.

Extent Type Hexadecimal Codes:

Code	Meaning
00	Next three fields do not indicate any extent.
01	Prime area (indexed sequential), or consecutive area, etc (that is, the extent containing the user's data records.)
02	Overflow area of an indexed sequential file.
04	Cylinder index or master index of an indexed sequential file.
40	User label track area.
80	Shared cylinder indicator.

OPEN(R) Macro



Note: To write the most efficient code (in a multiprogramming environment), we recommend that the self relocating form of OPEN be used. (See also Appendix G.)

The OPEN macro instruction activates files processed with the DTFPH macro. When OPEN is used, the symbolic address constants generated from the parameter list are not self-relocating. When OPENR is specified, the symbolic address constants are self-relocating. For this reason, use of OPENR is recommended.

Self-relocating programs using LIOCS must use the OPENR macro instruction to activate all files, including printer-keyboard files. The OPENR macro also relocates all address constants within the DTF tables. If symbolic notation is used, the user must establish addressability through a base register.

Note: Zero constants are not relocated except when they constitute the module address.

If OPEN attempts to activate a logical IOCS file (DTF) whose device is unassigned, the job is terminated. If the device is assigned IGN, the OPEN(R) does not activate the file and turns ON the DTF byte 16, bit 2, to indicate the file is not activated.

The symbolic name of the file (DTF filename) is entered in the operand field. A maximum of 16 files may be opened with one OPEN (or OPENR) by entering the filenames as additional operands. Alternately, the user can load the address of the DTF filename in a register and specify the register using ordinary register notation. The high-order 8 bits of this register must contain zeros. A filename may be preloaded into any register, 0-15.

Note: If you use register notation, we recommend using only registers 2-12. This will make your programs more compatible with the Operating System (OS).

Whenever an input/output DASD or magnetic tape file is opened and a user plans to process user-standard labels (UHL or UTL), or nonstandard tape labels, he must provide the information for checking or building the labels. If this information is obtained from another input file, that file must be opened, ahead of the DASD or tape file. Do this by specifying the input file ahead of the tape or DASD file in the same OPEN, or by issuing a separate OPEN preceding the OPEN for the file.

If an output tape (specified to contain standard labels) is opened and does not contain a volume label, a message is issued to the operator. He can then enter a volume serial number allowing the volume label to be written on the output tape.

PIOCS--Single Volume Mounted--Output

When processing with physical IOCS, OPEN is used only if the user wants to build standard labels. When the first OPEN for the volume is issued, OPEN checks the standard VOL1 label and the extents specified in the extent cards for the mounted volume:

1. The extents must not overlap each other.
2. If user standard header labels are written, the first extent must be at least two tracks.
3. Only types 1 and 8 extents are valid.

OPEN checks all the labels in the VTOC to ensure that the file and file label created does not destroy an existing file whose expiration date is still pending. After this check, OPEN creates the standard label(s) for the file and writes the label(s) in the VTOC.

If the user wishes to create his own user standard header labels (UHL) for the file, he must include the DTF entry LABADDR. OPEN reserves the first track of the first extent for these labels and gives control to the user's label routine. After this, the first extent of the file can be used. Each time the user determines that

all processing for an extent is completed, he issues another OPEN for the file to make the next extent available. When the last extent on the last volume of the file is processed, OPEN issues a message. The system operator has the option of canceling the job, or typing in an extent on the printer-keyboard and continuing the job. If the system provides DASD file protection, only the extents opened for the mounted volume are available to the user.

PIOCS--Single Volume Mounted--Input

When processing with physical IOCS, OPEN is used only if the user wants to check standard labels.

When the mounted volume is opened for the first time, OPEN checks the extents specified in the extent cards (for example, checks that the extent limit address for the device being opened is valid). OPEN also checks the standard VOL1 label and then checks the file label(s) in the VTOC. If the system provides DASD file protection, only the extents opened for the mounted volume are available to the user.

If LABADDR is specified, OPEN makes the user standard header labels (UHL) available to the user one at a time for checking. Then, OPEN makes the first extent available for processing.

Each time the user determines that all processing for an extent is completed, he issues another OPEN for the file to make the next extent available. If another extent is not available, OPEN stores the character F (for EOF) in byte 31 of the DTFPH table. The user can determine the end of file by addressing and checking the byte at filename+30.

PIOCS--All Volumes Mounted--Output

If all output volumes are mounted when creating an output file with physical IOCS, each volume is opened before the file is processed. OPEN is used only if standard labels are checked or written.

For each volume, OPEN checks the standard VOL1 label and checks the extents specified in the extent cards:

1. The extents must not overlap each other.
2. Only type-1 extents can be used.
3. If user standard header labels are created, the first extent must be at least two tracks long.

OPEN checks all the labels in the VTOC to ensure that the created file does not destroy an existing file with an expiration date still pending. After this check, OPEN creates the standard label(s) for the file and writes the label(s) in the VTOC.

If the user wishes to create his own user standard header labels for the file, he must include the DTF entry LABADDR. OPEN reserves the first track of the first extent for these labels and gives control to the user's label routine.

If the XTNTXIT entry is specified, OPEN stores the address of a 14-byte extent information area in register 1. (See Physical IOCS (PIOCS): DTFPH Macro for the format of this area.) Then, OPEN gives control to the user's extent routine. The user can save this information for later use in specifying record addresses. If the user's DASD file is file protected, he cannot write on any extents while in the XTNTXIT routine. When checking is complete, the user returns control to OPEN by issuing the LBRET 2 macro. The next volume is then opened. After all the volumes are opened, the file is ready for processing.

PIOCS--All Volumes Mounted--Input

When all volumes containing the file are on-line and ready at the same time, each volume is opened one at a time before any processing is done. OPEN is used only when standard labels are processed. For each volume, OPEN checks the extents specified in the extent cards, and checks the standard VOL1 label on track 0 and the file label(s) in the VTOC. If LABADDR is specified, OPEN makes the user standard labels available, one at a time, for checking.

If XTNTXIT is specified, OPEN stores the address of a 14-byte extent information area into register 1. (See Physical IOCS (PIOCS): DTFPH Macro for the format of this area.) Then OPEN gives control to the user's extent routine. For example, the user can save this area and use the information for later use in specifying record addresses. If the DASD file is file protected, the user cannot write on any extents while in the XTNTXIT routine.

LBRET Macro

Name	Operation	Operand
[name]	LBRET	{1|2|3}

The LBRET macro is issued in user subroutines when processing is completed and the user wishes to return control to IOCS. LBRET applies to subroutines that write or check DASD or magnetic tape user standard labels, write or check tape nonstandard labels, or check DASD extents. The operand used depends on the function to be performed. See Label Processing.

CHECKING USER STANDARD DASD LABELS: IOCS passes labels to the user one at a time until the maximum allowable number are read and updated, or until the user signifies he wants no more. If the user issues LBRET 3 in his label routine, IOCS updates (rewrites) the label read and passes him the next label. If LBRET 2 is issued, IOCS reads and passes the next label to the user. If an end-of-file record is read when LBRET 2 or LBRET 3 is used, label checking is automatically ended. If the user wants to eliminate the checking of one or more remaining labels, he should issue LBRET 1.

WRITING USER STANDARD DASD LABELS: The user builds the labels one at a time and uses LBRET to return to IOCS to write the labels. LBRET 2 is used if the user wishes to regain control after IOCS writes the label. If, however, IOCS determines that the maximum number of labels has been written, label processing is terminated. LBRET 1 is used to stop writing labels before the maximum number is written.

CHECKING DASD EXTENTS: When using physical IOCS on an input file with all volumes mounted, the user can process his extent information. After each extent is processed, the user should issue a LBRET 2 to receive the next extent. When extent processing is complete, the LBRET 1 macro returns control to IOCS.

CHECKING USER STANDARD TAPE LABELS: IOCS reads and passes the labels to the user one at a time until a tapemark is read, or until the user signifies he does not want any more labels. LBRET 2 is used if the user wants to process the next label. If IOCS reads a tapemark, label processing is automatically terminated. LBRET 1 is used if the user wants to bypass any remaining labels.

WRITING USER STANDARD TAPE LABELS: The user builds the labels one at a time and returns to IOCS, which writes the labels. When LBRET 2 is used, IOCS returns control to the user (at the address specified in LABADDR) after writing the label. LBRET 1 must be used to terminate the label set.

WRITING OR CHECKING NONSTANDARD TAPE LABELS: The user must process all his nonstandard labels at once. LBRET 2 is used after all label processing is completed and the user wants to return control to IOCS. For an example see Appendix D.

FEOV Macro

Name	Operation	Operand
[name]	FEOV	{filename { (1) }

The FEOV (forced end-of-volume) macro instruction is for files on magnetic tape (programmer logical units only) to force an end-of-volume condition before sensing a reflective marker. This indicates that processing of records on one volume is considered finished, but that more records for the same logical file are to be read from, or written on, the following volume. For system units, see SEOV Macro.

The name of the file, specified in the header entry, is the only parameter required in the operand. The name can be specified either as a symbol or in register notation.

When physical IOCS macro instructions are used and DTFPH is specified for standard label processing, FEOV may be issued for an output file only. In this case, FEOV writes a tapemark, the standard trailer label, and any user-standard trailer labels if DTFPH LABADDR is specified. When the new volume is mounted and ready for writing, IOCS writes the standard header label and user-standard header labels, if any.

SEOV Macro

Name	Operation	Operand
[name]	SEOV	filename

The SEOV (System End-Of-Volume) macro instruction must only be used with physical IOCS to switch automatic volume for magnetic tape output files if SYSLST or SYSPCH are assigned to a tape output file. The routine writes a tapemark, rewinds, unloads the file, and checks for an alternate tape. If none is found, a message is issued to the operator who can mount a new tape on the same drive and continue. If an alternate unit is assigned, the macro fetches the alternate switching routine to promote the alternate unit, opens the new tape, and makes it ready for processing. When using this macro, the user must detect the end-of-volume condition in the CCB.

CLOSE(R) Macro

Op	Operand
for self-relocating programs	
CLOSER	{filename1 / (r1)} [,{filename2 / (r2)}...,{filenamen / (rn)}]
for programs that are not self-relocating	
CLOSE	{filename1 / (r1)} [,{filename2 / (r2)}...,{filenamen / (rn)}]

Note: To write the most efficient code (in a multiprogramming environment, we recommend that the self relocating form of OPEN be used. (See also Appendix G.)

The CLOSE macro instruction is used to deactivate any file that was previously opened in any input/output unit in the system. (Console files cannot be closed.) A file may be closed at any time by issuing this macro instruction. No further commands can be issued for the file unless it is opened.

When the operation CLOSE is used, the symbolic address constants that CLOSE generates from the parameter list are not self-relocating. When CLOSER is specified, the symbolic address constants are self-relocating. For this reason, the self-relocating form of CLOSE is recommended.

The symbolic name of the logical file (assigned in the DTF header entry) to be closed is entered in the operand field. A maximum of 16 files may be closed by one instruction by entering additional filename parameters as operands. Alternately, the user can load the address of the filename in a register and specify the register by using ordinary register notation. The high-order 8 bits of this register must be zeros. The address of the filename may be preloaded into any register, 0-15.

Notes:

- If you use register notation, we recommend using only registers 2-12. This will make your programs more compatible with the Operating System (OS).
- If CLOSE is issued to an unopened magnetic tape input file, the option specified in the DTF rewind option is performed. If CLOSE is issued to an unopened magnetic tape output file, no tapemark or labels are written.

Supervisor-Communication Macros

The supervisor is a control program that provides specialized services to problem programs. These services differ slightly, depending on the execution environment. In the batch-job environment the supervisor processes interruptions, I/O requests, and program retrieval. In addition to these functions, the supervisor also determines which program (Foreground1, Foreground2, or Background) is to be executed in a multiprogramming environment.

The interruptions handled by the supervisor result from five conditions:

- Input/Output
- Program Check
- Machine Check
- External Signal (including timer)
- Supervisor Call

The user can request the supervisor to set up linkages so that his routines can handle program check, and operator communication and/or timer interrupts.

If a batched-job foreground environment is specified at system generation time, individual communication regions (Figure 58) are defined for each of the three partitions. This facility allows each partition to modify and check its respective communication region between job steps. However, if the batched-job foreground environment is not specified, only one communication region is generated and only the background program can modify its contents. Any program can read from the communication region of its choice. If a program attempts to modify another partition's communication region, the program is canceled.

Several macro instructions are available to the programmer to enable him to communicate with the supervisor. Thus, he can utilize the functions performed by the supervisor or have access to the communication region in the supervisor. To make use of the supervisor functions requires switching from problem state to supervisor state. Therefore, most macro instructions used for this purpose generate a Supervisor Call (SVC) instruction. The macro instructions included in this section are:

CANCEL	Terminates all remaining steps of the job.
CHKPT	Takes checkpoints in a batched partition.
COMRG	Obtains the address of the appropriate communication region.
DUMP	Terminates the job step and dumps main storage.
EOJ	Informs the supervisor that the current problem-program job step is completed.
EXIT	Returns to the point of interruption from a user routine for interval timer, program check, or operator communication.
FETCH	Loads and gives control to a program phase.
GETIME	Obtains the time of day.
LOAD	Loads a program phase and returns control to the calling phase.
MVCOM	Modifies the content of the user's portion of the appropriate communication region.
PDUMP	Obtains a selective (snapshot) dump of main storage.
RELEASE	Releases devices at execution time. Changes a temporary assign to its permanent assign, or to unassigned status if there is no permanent assignment.
SETIME	Requests the control program to take a program exit or set a bit in the TECB after a specific time interval.
STXIT	Activates a problem program's abnormal termination program check, interval timer, or operator communication routine, or cancels the use of such a routine.
TECB	Generates a timer event control block.
WAIT	Yields control until the expiration of the interval timer.

Multiprogramming Restrictions on Use of Supervisor Macros: If MVCOM is used to modify another partition's communication region, the program executing MVCOM is canceled. Thus, if SUPVR MPS=YES is specified at system generation time, the MVCOM macro cannot be used in a foreground program. The interval timer macros SETIME,STXIT IT, and EXIT IT can be used in only one program at a time (Foreground1, Foreground2, or Background). This timer macro may be specified at system-generation time but can be changed by the system operator. CHKPT is ignored in a foreground job in the single program mode.

Program Loading

Phases may be loaded into main storage from the system core image library or a private core image library with the FETCH and LOAD macro instructions. FETCH gives control to the phase that was loaded while LOAD returns control to the phase that issued the macro instruction.

All IBM supplied $$A, $$B, and $$R phases (transients) must be placed in the system core image library. When a phase is requested, the system core image library and the private core image library are searched, if necessary. If the phase starts with $, the system core image library is searched first. If the phase requested is an IBM supplied $$A or $$B transient which is not located in the system core image library, the system enters the wait state with an error message of 04W (X'04E6') in bytes 0 and 1 of lcw main storage. For other phases starting with $, if the phase is not located in the system core image library, the private core image library assigned to the partition is then searched. If the requested phase does not begin with $, the private core image library is searched first, followed by a search of the system core image library.

Support of a private core image library is a system generation option.

FETCH--FETCH A PHASE

Name	Op	Operand
[name]	FETCH	{phasename (1)} , {entryname (0)}

The FETCH macro instruction loads the phase specified in the first parameter. The phase name can be 1-8 characters long. Control is passed to the address specified by the second operand. If the second operand is not specified, control is passed to the entry point determined at linkage edit time.

The parameters can be specified either as symbols or in register notation. When register notation is used for phasename, the register must be preloaded with the address of an 8-byte field that contains the phasename as alphameric characters. If necessary, the phasename should be left-adjusted and padded with blanks.

If ordinary register notation is used for entryname, the absolute address of the entry point of the phase should not be preloaded into register 1. If, instead, a symbolic name is used for entryname, the macro expansion results in a V-type address constant. The entryname does not have to be identified by an EXTRN statement.

If the physical transient overlap option is specified at system generation time, an increase in throughput can result by overlapping FETCH I/O operations of one partition with problem program processing in another partition.

LOAD--LOAD A PHASE

Name	Op	Operand
[name]	LOAD	{phasename (1)} [,{loadaddr (0)}]

The LOAD macro instruction loads the phase specified in the first parameter and returns control to the calling phase. The phasename can be 1-8 characters long. The user should code his LOAD in such a manner that it cannot be overlaid by the new phase.

After execution of the macro, the entry-point address of the called phase is returned to the programmer in register 1. This entry-point address is determined at linkage-edit time.

If an optional address parameter is provided, the load-point address specified to the linkage editor is overridden, and the phase is loaded at the specified address. The address used must be outside the supervisor area. When an overriding address is given, the entry-point address is relocated and returned in register 1.

None of the other addresses in the phase are relocated.

The parameters can be specified either as symbols or in register notation. When register notation is used for phasename, the register must be preloaded with the address of an 8-byte field that contains the phasename. If necessary, the phasename should be left-justified and padded with blanks. If ordinary register notation is used for loadaddr, this parameter should not be preloaded into register 1.

If the physical transient overlap option is specified at system generation time, an increase in throughput can result by overlapping LOAD I/O operations of one partition with problem program processing in another partition.

Communication Region

As Figure 58 shows, the communication region is a storage area within the supervisor. Programs can check any communication region available (three in a batched-job foreground environment). However, a program can only modify its own communication region. Single program mode programs in a batched job foreground environment do not have their communication regions updated.

Figure 58 shows the portion of the communication region that contains information of interest to the user. For a complete layout of the communications region, see the System Programmer's Guide listed on the cover of this manual.

Field Length	Information
8 bytes	Calendar date. Supplied from system date whenever the JOB statement is encountered. The field can be two forms: mm/dd/yy or dd/mm/yy where mm is month, dd is day, yy is year. It can be temporarily overridden by a DATE statement.
2 bytes	Address of the first byte of the problem program area.
2 bytes	Address of the first byte following the supervisor area. If storage protect is specified, the address is that of the first byte with a storage protect key of 1. If storage protect is not specified, the address is that of the problem program area.
11 bytes	User area for interjob or intrajob step communications. All 11 bytes set to zero when the JOB statement for the job is encountered.
1 byte	UPSI (user program switch indicators). Set to binary zero when the JOB statement for the job is encountered. Initialized by UPSI job control statement.
8 bytes	Job name as found in the JOB statement for the job.
4 bytes	Address of the uppermost byte of the program area. When the first phase of a foreground or background program is loaded and given control, register 2 contains the address of the uppermost byte of the respective program area.
4 bytes	Address of the uppermost byte of the current phase placed in the problem program area by the last FETCH or LOAD macro instruction in the job.
4 bytes	Highest ending main storage address of the phase among all the phases having the same first four characters as the operand on the EXEC statement. For the background partition only, job control builds a phase directory of these phases. The address may be incorrect (1) if the program loads any of these phases above or below its link edited origin address, (2) if programs that have the first four characters in common are in the private core image library as well as in the system core image library. If the EXEC statement has no operand, job control places in this location the ending address of the phase just link edited.
2 bytes	Length of batch or background program label area.

Macro instructions (COMRG and MVCOM) allow the problem program to communicate with the supervisor and the appropriate communication region.

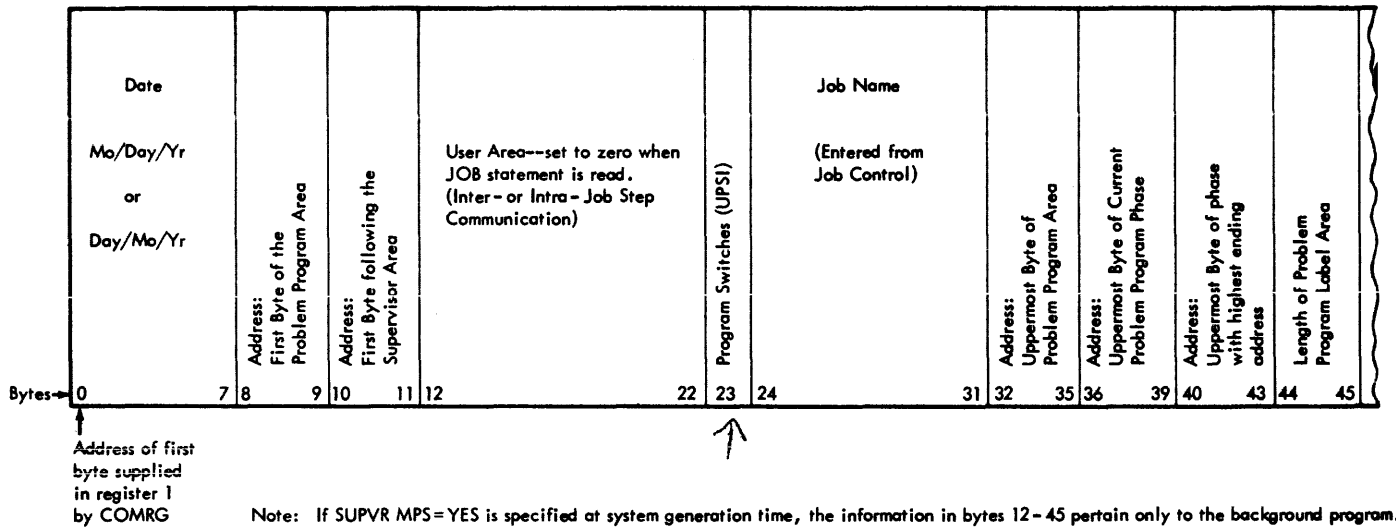


Figure 58. Communication Region (in Supervisor)

COMRG--GET ADDRESS OF COMMUNICATION REGION

Name	Operation	Operand
[name]	COMRG	

When COMRG is issued, the address of the appropriate communication region is placed in register 1. A problem program can read any portion of its own communication region by using register 1 as a base register.

MVCOM--MOVE TO COMMUNICATION REGION

Name	Operation	Operand
[name]	MVCOM	to,length,{from / (0)}

The MVCOM macro instruction modifies the content of bytes 12-23 of the appropriate communication region. This macro cannot modify another partition's communication region.

The operand from represents the address (either as a symbol or in register notation) of the bytes to be inserted. The operand length represents the number of bytes (1-12) inserted. The operand to is the relative address of the first communication region byte modified (12-23). (The to address is relative to the first byte of the region.)

The following shows how to move three bytes from the symbolic location DATA into bytes 16-18 of the communication region. The execution of this macro fetches a routine into the physical transient area.

Name	Operation	Operand
[name]	MVCOM	16,3,DATA

RELEASE -- RELEASE TEMPORARY ASSIGNS (BATCHED PROCESSING ONLY)

Note: This macro not valid for Single Program Initiation.

The problem program supplies the names of the units (SYSxxx) to be released. The format of the RELEASE macro is:

Name	Operation	Operand(s)
[name]	RELEASE	(SYSxxx,...,n) [,savearea]

n may be up to a maximum of 16 units to be released.

The savearea parameter is optional. If provided, it should be the name of an 8-byte word aligned area where registers 0 and 1 are saved for the problem program. If not provided, the contents of registers 0 and 1 are destroyed.

All the units in the operand sublist are checked to assure that no system logical

units are requested for release. If there are system logical units specified, an MNOTE is issued and the unit is ignored.

After all checking is done, a unit table is set up and register 0 is loaded with the table address. If the savearea option is specified, registers 0 and 1 are saved.

If there is no permanent assignment, the device is unassigned. If the device is at permanent assignment level, no action is taken on the unit.

Before any release is attempted, a check is made for ownership of the unit. If the requesting partition does not own the unit, or if the unit is already unassigned, the request is ignored.

> Recommendation: The user should inform the system operator via a message that the assignment was released.

Time of Day Macro

GETIME--GET TIME OF DAY IN REGISTER 1

Name	Operation	Operand
[name]	GETIME	{STANDARD|BINARY|TU}

The GETIME macro instruction obtains the time of day at any time during program execution. STANDARD is assumed if no operand is given.

If STANDARD is specified, the time of day is returned in register 1 as a packed decimal number: hhmmss where h is hours, m is minutes, and s is seconds with low-order sign. The time of day may be stored, unpacked, or edited.

> Note: Lengthy conversion routines are generated (in line) each time STANDARD is used. Therefore, this function should be put into a subroutine if it is used frequently.

If BINARY is specified, the time of day is returned in register 1 as a binary integer in seconds.

If TU is specified, the time of day is returned in register 1 as a binary integer in units of 1/300 second.

GETIME can be used only if the timer feature was specified at system generation time and if the CPU has the timer feature.

> Note: The timer feature is independent of the interval timer options (SETIME and STXIT). GETIME can be used by any area in a multiprogramming environment, regardless of which area is using the timer.

Interval Timer and User Exit Macros

Programs using the interval timer macros (SETIME, WAIT, TECB, STXIT IT, EXIT IT) must be executed with a supervisor containing the optional interval timer routines and on a CPU with the timer feature. The user specifies at system generation time whether the supervisor is to be generated with the interval timer routines. Any other STXIT or EXIT macros also require the option to be specified at system generation time.

In a multiprogramming environment, only one program or task at a time can use the interval timer macros. The partition of the program or task is specified at system generation time but can be temporarily changed by the operator. There are two distinct methods of using the interval timer macros. Only one method can be used at a time.

The first method allows the problem program to set the timer and enter a routine in the problem program when the time elapses. The SETIME, STXIT, and EXIT macros do this. However, only the main task of the partition controlling the timer can issue these instructions.

If the problem program uses QTAM, refer to the QTAM Message Control Program publication listed in the Preface.

In the second method, a given routine can be performed at timed intervals. The time set is a real time interval and is not stopped or adjusted when the program using the timer does not have control. The SETIME, TECB, and WAIT macros are used. However, if multitasking, only one task in the partition can use the method and the WAIT on the TECB must appear in that task. In this case, the priority of the program or task assigned to process interval timer interruptions must be considered.

SETIME--SET INTERVAL TIMER

Name	Operation	Operand
[name]	SETIME	{seconds / (1)}

The SETIME macro instruction sets the interval timer to the value that is specified in the operand. The largest allowable value is 55918 (equivalent to 15 hours, 31 minutes, 58 seconds). A register may be specified as the operand. The register must contain the number of seconds in binary. When the specified timer interval has elapsed, the interval timer routine supplied by the user is entered.

If a routine is not supplied to the supervisor (via the STXIT macro instruction) by the time of the interruption, the interruption is ignored. When a program is restarted from a checkpoint, any timer interval set by a SETIME macro is not restarted.

STXIT--SET LINKAGE TO USER ROUTINE(S)

Name	Operation	Operand
To establish linkage		
[name]	STXIT	{AB / IT / PC / OC}[,{rtnaddr / (0)}[,{savearea / (1)}]]
To terminate linkage		
[name]	STXIT	{AB|IT|PC|OC}

The STXIT (set exit) macro establishes or terminates linkage from the supervisor to a problem program routine for abnormal task termination, interval timer, program check, or operator communication interrupts processing. To return from these routines, refer to the EXIT macro. This linkage must be established for a program or task by issuing the STXIT macro at the inception of main program or task execution. If only the first operand is present, linkage to the problem program routine is terminated.

AB

An abnormal task termination routine is entered if a job or task is terminated for some reason other than a CANCEL, DETACH, DUMP, or EOJ macro issued by either the problem program or the supervisor. Upon entry to the task's abnormal termination routine:

- Byte 2 bit 1 is posted in the task's attachment ECB, if AB=YES is generated in the supervisor.
- Register 0 contains the abnormal termination code in its low order byte (Figure 59).
- Register 1 contains the address of task's abnormal termination save area, which contains the PSW and the contents of registers 0-15 at the time of the abnormal termination.

The abnormal termination routine can then examine this data and take whatever action is necessary.

Instructions that might be used in this routine are the DEQ, POST, and CLOSE. However, if an abnormal termination condition occurs in an abnormal termination routine, the job or task is abnormally terminated without regard to an abnormal termination exit. Thus, the problem program abnormal termination routine should avoid instructions such as ENQ, CHKPT, and any other I/O instructions that may cause an abnormal termination.

Note: For systems operating in a QTAM environment, QTAM files must be closed before issuing this macro.

Hexadecimal Representation	Specific Abnormal Termination Code Meaning
17	Main task issued a CANCEL macro with subtask still attached
18	Main task issued a DUMP macro with subtask still attached
19	Operator replied cancel as the result of an I/O error message
1A	An I/O error has occurred (see PSW)
1C	CANCEL ALL macro issued in another task
1D	Main task terminated
1E	A DEQ macro was issued for a resource but tasks previously requesting a resource cannot be found because their save areas (containing register 0) were modified
20	A program check occurred
21	An invalid SVC was issued by the problem program or macro instruction
22	Phase not found in the core image library
23	CANCEL macro issued
24	Canceled due to an operator request
25	Invalid main storage address given (outside partition)
26	Device not assigned
27	Undefined logical unit
30	Read past a /& statement
31	I/O error queue overflow during system error recovery procedure
32	Invalid DASD address
33	No long seek on a DASD
Others	Reserved for future systems use

Figure 59. Abnormal Termination Codes

After the appropriate action is taken, the problem program abnormal termination routine should end with a CANCEL DETACH, DUMP, or EOJ macro instruction. However, DUMP forces a storage map of the partition even if option NODUMP was specified. At this time, the subtask's attachment ECB bit 0 of byte 2 is posted, all held tracks are freed, messages to identify the reason for abnormal termination are given, and the subtask is detached. If the main task issued the CANCEL macro, the entire partition is terminated with every subtask abnormal termination exit taken in order of priority.

If the system was generated with the multitasking option, each task may require its own abnormal termination routine. A main task can attach a subtask with an ABSAVE operand. This assumes the subtask will use its abnormal termination routine.

However, the subtask may override this specification by issuing its own STXIT AB macro.

IT

An Interval Timer interruption routine is entered when the interval timer elapses. If multitasking, only the main task can process the condition.

OC

An Operator Communication interruption routine is entered in a background job when the external interrupt key on the console is pressed. In a foreground program, the OC routine is entered when you press the request key on the 1052 and request the foreground OC routine. If multitasking, only the main task can process this condition.

PC

A Program Check interruption routine is entered when a program check occurs. If a program check occurs in a routine being executed from the logical transient area, the job containing the routine is abnormally terminated. A program check interruption routine can be shared by more than one task within a partition. Do this by executing the STXIT macro in each subtask with the same routine address but with separate save areas. To successfully share the same PC routine, the routine must be reenterable. That is, it must be capable of being used concurrently by two or more tasks.

rtnaddr

Entry point address of the routine that processes the condition described in the first operand.

savearea

Address of a 72-byte area in which the supervisor stores the old PSW and general registers 0-15 in that order. The problem program must have a separate save area for each routine that is included.

The routine address and the savearea address can be supplied as a symbol or in special or ordinary register notation.

If a STXIT macro is issued and the supervisor is not generated to handle the requested facility, the job is abnormally terminated. This also applies to a program that requests the timer interrupt and the timer is not allocated.

If an abnormal termination condition occurs and linkage has not been established to a problem program or main task abnormal termination routine, the partition is abnormally terminated. However, if the abnormal termination condition occurs in a subtask without exit linkage, only the subtask is terminated. An interval timer, or operator communication condition occurring without exit linkage is ignored.

If a program check condition occurs in a main task without exit linkage, the partition is terminated. However, if this same condition occurs in a subtask, only the subtask is terminated.

The following shows what happens when a condition occurs where a STXIT routine is being processed within a particular partition:

Routine Being Processed	Condition Occurring			
	AB	IT	OC	PC
AB	T	I	I	I
IT	S	I	H	H
OC	S	H	Ib Ef	H
PC	S	H	H	T

Ef Error message issued in foreground program, and control returns to interrupted OC routine.

H Condition honored. When processing of new routine completes, control returns to interrupted routine.

I Condition ignored for all partitions.

Ib Interrupt ignored in a background partition.

S Execution of the routine being processed is suspended, and control transfers to the AB routine.

T Job abnormally terminated. If AB routine present and there has not been an interruption in the AB routine, its exit is taken. Otherwise, a system abnormal termination occurs.

Note 1: When restarting a program from a checkpointed position, any STXIT linkages established prior to the checkpoint are destroyed.

Note 2: If a program is using a logical transient routine when a timer interrupt occurs, the user timer routine is not entered until the logical transient routine is released.

Note 3: Each routine should provide its own addressability by initializing its base register.

Note 4: If a program issues a QTAM SVC WAIT, the routine specified as linkage must store register 1 in the save area (savearea + 12) specified in the third operand.

EXIT--EXIT FROM USER'S INTERRUPT ROUTINE(S)

Name	Operation	Operand
[name]	EXIT	{PC|IT|OC|MR}

The EXIT macro instruction is used to return from the user routine, to the instruction in the interrupted program immediately after the instruction where the interruption occurred. The user routine is specified in the STXIT macro (except for MR).

For PC, IT, and OC, the PSW and registers are restored from the save area; thus, the save area contents should not be destroyed. The operands have the following meanings:

PC Exit from the user's program check routine.

IT Exit from the user's interval timer routine.

OC Exit from user's routine that handles the operator attention interrupt.

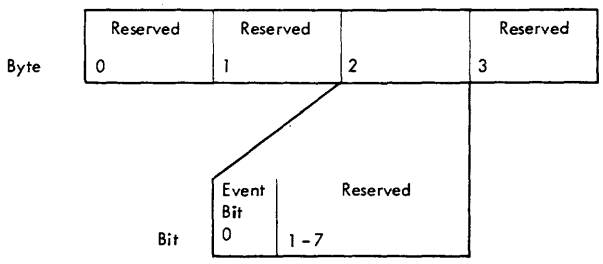
MR The MR indicates that the user stacker selection routine (MICR document processing) is exiting to the external interrupt routine of the supervisor. The name of the user stacker selection routine is specified in the DTFMR macro instruction.

Method 2 - Macros

TECB - BUILD TIMER EVENT CONTROL BLOCK

Name	Operation	Operand
tecbname	TECB	

The TECB generates a timer event control block (Figure 60) at the address of tecbname. This block contains an event bit that indicates when the time interval specified in SETIME has elapsed.



The Event Bit is set ON by the Supervisor's Timer Routines

Value	Indication
0	time specified in SETIME has not elapsed
1	time specified in SETIME has elapsed

Figure 60. Time Event Control Block (TECB)

SETIME--SET INTERVAL TIMER

Name	Operation	Operand
[name]	SETIME	{seconds / (1)}, {tecbname / (r)}

The SETIME macro sets the amount of time that must elapse before the TECB event bit is set to 1 and the routine following the WAIT macro can be processed. When SETIME is issued, the event bit is set to 0.

The number of seconds can be specified directly or in register notation. The largest allowable value is 55918, (equivalent to 15 hours, 31 minutes, 58 seconds). If a register is specified, the register must contain the number of seconds in binary.

The user can specify the tecbname or specify the register (r) (r cannot be 0 or 1) in which is placed the address of the corresponding TECB. After SETIME is executed, the supervisor returns the TECB address in register 1.

WAIT--WAIT FOR TIMER ELAPSE

Name	Operation	Operand
[name]	WAIT	{tecbname / (1)}

The WAIT macro ensures that the time interval specified in SETIME has elapsed (event bit turned ON) before execution of the program issuing the wait continues. When a WAIT macro is processed in a multiprogramming environment, control is given to the supervisor, which makes the time available to another partition.

The user can either specify the tecbname or use register notation. The WAIT macro instruction loads the TECB address into register 1 unless register (1) is specified.

Note: The SETIME macro instruction leaves the TECB address in register 1.

Dump Macros

PDUMP--PARTIAL DUMP OF MAIN STORAGE

Name	Operation	Operand
[name]	PDUMP	{address1 / (r)}, {address2 / (r)}

This macro instruction provides a hexadecimal dump of the general registers and of the main storage area contained between the two address expressions (address1 and address2). One or both of the addresses can be given in registers. The contents of registers 0-1 are destroyed, but the CPU status is retained. Thus, PDUMP furnishes a dynamic dump (snapshot) useful for program checkout. Processing continues with the next user instruction.

The dump is always directed to SYSLST with 121 byte records. The first byte is an ASA control character. When SYSLST is a disk drive, the user must issue an OPEN macro to any DTF assigned to SYSLST after each PDUMP that is executed. The OPEN macro updates the disk address maintained in the DTF table to agree with the address where the PDUMP output ends. If the OPEN is not issued, the address is not updated, and the program is canceled when the next PUT is issued.

DUMP--DUMP PARTITION

Name	Operation	Operand
[name]	DUMP	

This macro terminates the job step and gives a hexadecimal dump of the supervisor, the partition that issued the macro, and the general registers, if the program or main task issued the macro. If a subtask issues the macro, the subtask is detached, the partition is not terminated, and the dump, as described, is given. The dump is always directed to SYSLST upon DUMP macro execution. SYSLST, if disk or tape, must be opened, and if tape, it must be positioned as desired.

Cancel and EOJ Macros

CANCEL--CANCEL THE JOB

Name	Operation	Operand
[name]	CANCEL	[ALL]

The CANCEL macro issued by a subtask abnormally terminates the subtask without branching to any abnormal termination routine. A CANCEL ALL macro issued in a subtask, or a CANCEL issued in the main task, abnormally terminates the partition (job). Job termination in multitasking causes all abnormal termination exits (via STXIT AB) to be taken for each task except the task that issued the CANCEL macro. Once these exits are taken, the job is terminated. Upon task termination, system messages (using the first 8 bytes of each subtask save area) are issued to identify each subtask terminated.

> Note: If the CANCEL macro is issued without an operand, the card cannot contain a comment unless the comment begins with a comma. If CANCEL ALL is issued, the card may contain a comment.

If the DUMP option was specified, and SYSLST is assigned, a system dump will occur

- If a CANCEL ALL macro is issued by a subtask, or
- If a CANCEL macro is issued by a maintask with subtasks attached.

EOJ--END-OF-JOB STEP

Name	Operation	Operand
[name]	EOJ	

The EOJ macro is issued in the main task, or only program within a partition, to inform the system that the job step is finished. If a subtask issues an EOJ, the subtask is detached and the remainder of the partition continues. The operand field is ignored.

Checkpointing a Problem Program

Checkpoint records the status of a problem program at designated intervals. If processing is terminated for any reason before the normal end of program, restart resumes the execution of the program from one of the checked points rather than from the beginning. For example, a job of higher priority may require immediate processing, or some malfunction such as a power failure may occur and cause an interruption. The checkpoint ability is provided through the CHKPT macro while the restart ability is provided through job control. (For information on restarting a checkpointed program, see the System Control and Service publication listed in the Preface.)

> Note: The checkpoint facility is not available for ASCII tape files.

A checkpoint cannot be taken following an end-of-file condition because both checkpoint and close routines are transients. If a user wishes to take advantage of checkpointing, a good time to take a checkpoint is immediately after opening files. If the checkpoint is not successful at this time, the program will cancel before any processing of files begins. The following illustrates the principle:

```
        START
          .
          .
          .
        OPENR  (RTAPEOUT),(RCHKPTF)
        BAL    RLINK, CHECKPT (Trial
               checkpoint)
          .
          .
          .
CHECKPT CHKPT  SYS000, ...
        LTR    0,0 (Test for success)
        BNZ    0(,RLINK) (Return if
               success)
        CANCEL ALL (Checkpoint failed)
          .
          .
        EOJ
```

USE OF THE CHKPT MACRO

Any partition, except a foreground partition in a single program mode, can issue the CHKPT macro successfully. CHKPT cannot be issued for a BTAM program. If multitasking, only the main task can

successfully checkpoint. CHKPT is ignored when issued by a subtask, a foreground partition in single program mode, or in any of the following additional conditions:

1. The device on which the checkpoint records are written is not a magnetic tape or a disk pack. (The device must be a 2311, 2314, or 2319 disk if the filename operand is present.)

2. End-of-reel is detected while writing the checkpoint on tape.

3. The area on disk is not large enough for a single checkpoint.

4. The macro is issued by a telecommunications program that has any I/O operation(s) pending on a telecommunications device.

5. The user-specified end address is greater than the end of the problem program area.

6. The CHKPT macro is issued before the disk checkpoint file is opened.

7. Any of the required DTFPH parameters for the disk checkpoint file contain errors.

8. If a subtask is attached in the partition being checkpointed.

9. If any DASD track for the partition being checkpointed is in the HOLD condition.

If a checkpoint is ignored, control returns to the user with binary zeros in register 0. Otherwise, register zero contains the appropriate checkpoint number (in unpacked decimal).

Checkpoints are usually taken after a specified period of time has elapsed, or after a certain volume of input is processed.

> Note: The maximum number of checkpoints in any one job is 9999 (decimal). If the user wishes additional checkpoints, the job must be forced to end-of-job and then restarted.

If multitasking, the method to use is not readily apparent. However, use the following two situations as a guide.

1. The multitasking operation requires checkpoints to be taken on a time interval basis. Therefore, at main task execution time, a STXIT macro establishes linkage for an interval timer interrupt. In the main task interval timer routine, the problem program issues WAIT macros to wait for the detachment of each subtask in the partition, and then takes the checkpoint. If the main task must take an immediate checkpoint, the interval timer routine must first detach all subtasks before issuing the CHKPT macro.

2. The multitasking operation requires checkpoints to be taken on a volume basis. Therefore, the main task attaches the subtasks necessary to perform the job, and then issues WAIT macros to wait for each subtask in the partition to detach. Each subtask keeps a count on the unit of work to be performed and detaches when it is finished. When all subtasks are detached the main task can take the checkpoint.

After the checkpoint is taken, the main task can then either attach more, or the same, subtasks to continue processing.

CHKPT MACRO

Name	Op	Operand
[name]	CHKPT	SYSnnn,{restart address / (r1)} [,{end address / (r2)}] [,{tpointer / (r3)}] [,{dpointer / (r4)}] [,{Filename / (r5)}]

Special register notation cannot be used with any of these operands.

SYSnnn

Specifies the logical unit on which the checkpoint information is stored. It must be an EBCDIC magnetic tape or a disk pack. (See Checkpoint File.)

Restart address (or r1)

Specifies a symbolic name of the problem program statement (or register containing the address) at which execution is to restart if processing must be continued later.

End address (or r2)

A symbolic name (or register containing the address) of the uppermost byte of the problem program area required for restart (see Communication Region). This address must follow the logic modules being included from the relocatable library. If this operand is omitted, all main storage allocated to the partition is checkpointed.

This operand has two advantages:

1. Less time and space is required for recording the checkpoint record set.
2. If a program using 24K of storage is being run in a larger system and only 24K is checkpointed, that program can be restarted either on a 24K or larger system.

In a multiprogramming environment, checkpoints must be restarted in the same partition that was checkpointed.

Tpointer (or r3)

The symbolic name of an 8-byte field contained in the problem program area. (See Repositioning Magnetic Tape.)

Dpointer (or r4)

The symbolic name of a DASD operator verification table that the user can set up in his own area of main storage. (See DASD Operator Verification Table.)

Filename (or r5)

Used only for checkpoint records on disk. It is the name of the associated DTFPH macro. (See Checkpoint on Disk.)

Information That Is and Is Not Saved

When the CHKPT macro is issued, the following information is saved:

- Information for the restart and other supervisor or job control routines.
- The general registers.
- Bytes 8-10 and 12-45 of the communication region.
- The problem program area (see End-Address operand).
- All DASD file protection extents attached to logical units belonging to the checkpointed program.

The following information is not saved:

- The floating-point registers. (If needed, these registers should be stored in the problem program area before issuing CHKPT, and restored in a user restart routine.)
- Any linkages to user routines set by the STXIT macro. (If needed, STXIT should be used in user's restart routine.)
- Any timer values set by the SETIME macro. (If needed, SETIME should be used in user's restart routine.)
- The program mask in problem program PSW. (If other than all zeros is desired, the mask should be reset in user's restart routine.)

  Notes for DASD and MICR Files: DASD system input or output files (SYSIPT, SYSLST, etc) must be reopened at restart time. In the user's restart routine, the programmer must be able to identify the last record processed before the checkpoint.

  For MICR files, the problem program must disengage the device and process all follow-up documents in the document buffer before taking each checkpoint. MICR files require the DTFMR supervisor linkages to be initiated at restart time. Do this by reopening the MICR file in the user's restart routine which clears the document buffer.

CHECKPOINT FILE

The checkpoint information must be written on a disk pack or an EBCDIC magnetic tape, either 7- or 9-track. The 7-track tape can be in either data conversion or translation mode. However, the magnetic tape unit must have the data conversion feature. On 7-track tapes, the header and trailer labels are written in the mode of the tape and the data records are written in data convert mode, odd parity.

Checkpoints on Tape

The programmer can either establish a separate file for checkpoints or embed the checkpoint records in an output data file. When the data file is read at a later time using logical IOCS, the checkpoint records are automatically bypassed. If physical IOCS is used, the user must program to bypass the checkpoint record sets. See Physical IOCS (PIOCS).

If a separate magnetic tape checkpoint file with standard labels is maintained, the labels should be either checked by an OPEN or bypassed by an MTC command before the first checkpoint is taken.

Checkpoints on Disk

If checkpoints are written on disk, the following must be observed:

- One continuous area on a single pack must be defined at execution time by the job control cards necessary to define a DASD file.
- The number of tracks required is computed as follows:

$$n\left\lceil \frac{\frac{x}{1+30+20}\ \frac{y}{18}}{} + \frac{c}{z} \right\rceil$$

where

n = The number of sets of checkpoint records to be retained. (When the defined extent is full, the first set of checkpoint records is overlaid.)

c = The number of bytes to be checkpointed in the user's problem program up to the end address specified in the CHKPT macro operand.

x = The number of disk extents including nonoverlapping split-cylinder extents. If split-cylinder extents overlap on the same cylinder, the number of extents counted is one used by the program. (This number is zero if DASD file protect is not used.)

y = For 2321, same as x.

z = 3625, if checkpoint records are to be written on a 2311.

7249, if checkpoint records are to be written on a 2314 or 2319.

For each division, the remainder is rounded up to the next highest whole number before multiplying by n.

- Each program can use a common checkpoint file or define a separate one. If a common file is used, only the last program using the file can be restarted.
- The checkpoint file must be opened before the CHKPT macro can be used.
- A DTFPH macro must be included for use by OPEN and the checkpoint routine. See Physical IOCS (PIOCS): DTFPH Macro.

REPOSITION I/O FILES

The I/O files used by the checkpointed program must be repositioned on restart to the record the user wants to read or write next. Checkpoint provides no aids for repositioning unit-record files. The programmer must establish his own repositioning aids and communicate these to the operator when necessary. Some suggested ways are:

- Taking checkpoints at a logical break point in the data, such as paper tape end of reel.
- Switching card stackers after each checkpoint.
- Printing information at checkpoint to identify the record in process.
- Issuing checkpoints on operator demand.

User sequential DASD input, output, and work files require no repositioning.

When updating DASD records in an existing file, the programmer must be able to identify the last record updated at checkpoint in case he needs to restart. This can be done in various ways, such as:

- Creating a history file to record all updates.
- Creating a field in updated records to identify the last transaction record that updated it. This field can be compared against each transaction at restart time.

Repositioning Magnetic Tape

Checkpoint provides some aid in repositioning 3420 or 2400-series magnetic tape files at restart. Files can be repositioned to the record following the last record processed at checkpoint.

This section and Figure 61 present the procedure. The fourth operand of the CHKPT macro points to two V-type address constants that the user specifies in his coding. The order of these constants is important.

1. The first constant points to a table containing the filenames of all logical IOCS magnetic tape files to be repositioned.
2. The second constant points to a table containing repositioning information for physical IOCS magnetic tape files to be repositioned.
3. If the first, second, or both constants are zero, no tapes processed by logical, physical, or both types of IOCS, respectively, are repositioned.

If the tables are contained in the same CSECT as the CHKPT macro, the constants may be defined as A-type constants.

The user must build the tables discussed. Each filename in the logical IOCS table points to the corresponding DTF table where IOCS maintains repositioning information.

- Magnetic tapes with nonstandard labels should be repositioned past the labels at restart time (presumably the labels are followed by a tapemark so that forward-space file may be used).
- If either a nonstandard label or unlabeled magnetic tape file is to be repositioned for reading backwards, the problem programmer must position the tape immediately past the tapemark following the last data record.
- Restart does not rewind magnetic tapes when repositioning them.
- A multifile reel should be prepositioned to the beginning of the desired file.
- The correct volume of a multivolume file must be mounted for restart.
- For tapes with a standard VOL label, restart writes the file serial number and volume sequence number on SYSLOG, and gives the operator the opportunity to verify that the correct reel is mounted.
- IOCS can completely reposition files on system logical units (SYSIPT, SYSLST, etc), if the tape is not shared with any other program and if the user keeps a physical IOCS repositioning table. However, if a system logical unit file is shared with other programs, a problem exists. Output, produced after the checkpoint, is duplicated at restart. Input records must be reconstructed from the checkpoint, or the user restart routine must find the last record processed before checkpoint.

The entries in the physical IOCS table are:

- First halfword. Hexadecimal representation of the symbolic unit address of the tape (copy from CCB).
- Second halfword. Number of files within the tape in binary notation. That is, the number of tapemarks between the beginning of tape and the position at checkpoint.
- Third halfword. Number (in binary notation) of physical records between the preceding tapemark and the position at checkpoint.

DASD OPERATOR VERIFICATION TABLE

If the Dpointer operand is used, the user can build a table (in his own area of main storage) to provide the symbolic unit number and the bin (cell) number of each DASD file used by his program. At restart, the volume serial number of these files is printed on SYSLOG for operator verification.

The entries in the DASD operator verification table must consist of the following two halfwords, in the order stated:

1. The symbolic unit in hexadecimal notation copied from the CCB bytes 6 and 7.

2. The bin (cell) number in hexadecimal notation. The bin number is always zero, except for a 2321, in which case the bin number varies with the cell (0-9) being verified.

There must be one entry for each DASD unit to be verified by the operator.

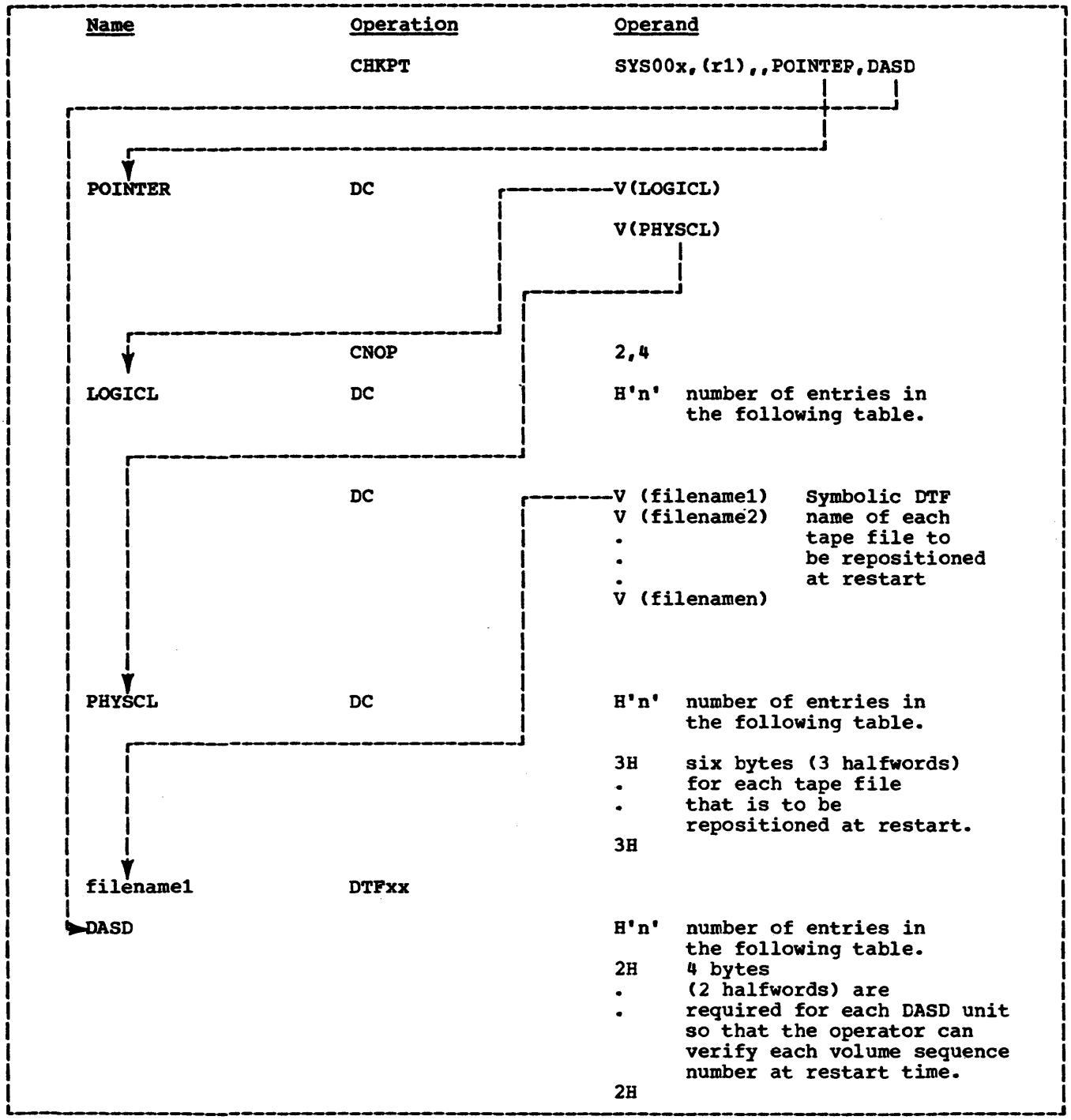


Figure 61. Repositioning Magnetic Tape

Additional Macro Instructions: Call, Save, and Return

A program may consist of several phases or routines produced by language translators and then combined by the linkage editor. The CALL, SAVE, and RETURN macro instructions are used for linkage between routines. These macros, with conventional register and save area usage, allow branching from phase to phase, and delivery of parameters. Also, the parameters can be delivered to another program. Passing control from one routine to another within the problem program is referred to as direct linkage.

Figure 63 shows linkage between a main program and two subroutines. Linkage can proceed through as many levels as necessary, and each routine may be called from any level. The routine given control during the job step is initially a called program. During execution of a program, the services of another routine may be required, at which time the current program becomes a calling program. For example, when main program passes control to B, B is a called program. When control is passed from B to C, B is the calling program and C is the called program.

Linkage Registers

To standardize branching and linking, registers are assigned specific roles (Figure 62). Registers 0, 1, 13, 14, and 15 are known as the linkage registers. Before a branch to another routine, the calling program is responsible for the following calling sequence:

1. Loading register 13 with the address of a register save area in that program which the called program is to use.
2. Loading register 14 with the address to which the called program will return control.
3. Loading register 15 with the address from which the called program will take control.
4. Loading registers 0 and 1 with parameters, or loading register 1 with the address of a parameter list. A typical calling sequence could read:

```
         CNOP  2,4
CALSEQ   LA    13,SAVAR      Load save
                             area address
         LA    1,PARLST      Load address of
                             a parameter list
         L     15,=V(SUBR)   Load entry
                             point address
         BALR  14,15         Load return
                             address
         .
         .
         .
SAVAR    DS    9D
PARLST   DC    A(PAR1,PAR2)
```

The address of save area (SAVAR) and the parameter list (PARLST) containing two parameters (PAR1 and PAR2) are passed to a subroutine (SUBR). SUBR returns control to this program at the next sequential instruction after the BALR instruction.

REGISTER NUMBER	REGISTER NAME	CONTENTS
0	Parameter register	Parameters to be passed to the called program.
1	Parameter register or Parameter list register	Parameters to be passed to the called program. Address of a parameter list to be passed to either the control program or a user's subprogram.
13	Save area register	Address of the register save area to be used by the called program.
14	Return register	Address of the location in the calling program to which control should be returned after execution of the called program.
15	Entry point register	Address of the entry point in the called program.

Figure 62. Linkage Registers

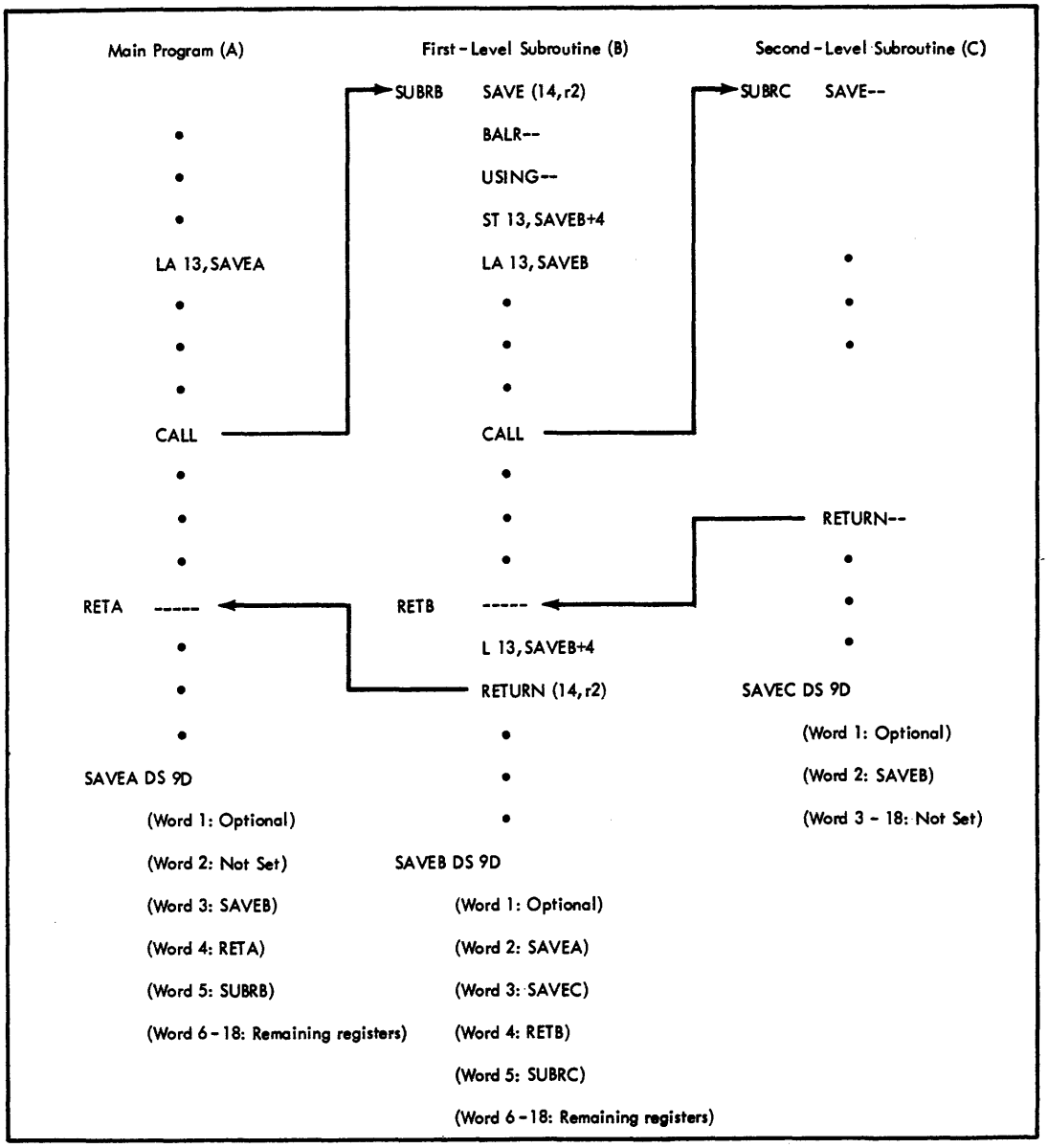


Figure 63. Direct Linkage

After execution of the calling sequence, the following should occur as a result of called program execution:

1. The contents of registers 2 through 14, and the program mask are unchanged.

2. The contents of registers 0, 1, and 15, and the contents of the floating point registers, and the condition code may be changed.

3. The parameter list addresses contains the results obtained from called program execution.

Save Areas

A called program should save and restore the contents of the linkage registers, as well as the contents of any register that it uses. The registers are stored in a save area that the higher level (calling) program provided. This procedure conserves main storage because the instruction to save and restore registers need not be in each calling sequence.

Every program must provide a save area and place its address in register 13 before it executes a direct linkage. This address is then passed to the called routine. A save area occupies nine doublewords and is

WORD	DISPLACEMENT	CONTENTS
1	0	Indicator byte and storage length; used by PL/I language program.
2	4	The address of the previous save area; that is, the save area of the subprogram that called this one (used for tracing purposes).
3	8	The address of the next save area; that is, the save area of the subprogram to which this subprogram refers.
4	12	The contents of register 14 containing the address to which return is made.
5	16	The contents of register 15 containing the address to which entry into this subprogram is made.
6	20	(The contents of) register 0.
7	24	(The contents of) register 1.
8	28	(The contents of) register 2.
9	32	(The contents of) register 3.
10	36	(The contents of) register 4.
11	40	(The contents of) register 5.
12	44	(The contents of) register 6.
13	48	(The contents of) register 7.
14	52	(The contents of) register 8.
15	56	(The contents of) register 9.
16	60	(The contents of) register 10.
17	64	(The contents of) register 11.
18	68	(The contents of) register 12.

Figure 64. Save Area Words and Contents in Calling Programs

aligned on a doubleword boundary. For programs to save registers in a uniform manner, the save area has a standard format (Figure 64) as follows:

- Word 1: An indicator byte followed by three bytes that contain the length of allocated storage. Use of these fields is optional, except in programs written in the PL/I language.
- Word 2: A pointer to word 1 of the save area of the next higher level program. The address passed to a routine in register 13. The contents of register 13 must be stored by a calling program before the calling program loads register 13 with the address of the current save area that is passed to a lower level routine (Figure 64, ST 13,SAVEB+4).
- Word 3: A pointer to word 1 of the save area of the next lower level program, unless this called program is at the lowest level and does not have a save area. (The called program requires a save area only if it is also a calling program.) Thus, the called program, if it contains a save area, stores the save area address in this word.

- Word 4: The return address, which is register 14, when control is given to the called program. The called program may save the return address in this word.

- Word 5: The address of the entry point of the called program. This address is in register 15 when control is given to the called program. The called program stores the entry-point address in this word.

- Words 6 through 18: The contents of registers 0 through 12, in that order. The called program stores the register contents in these words if it is programmed to modify these registers.

In any routine, the contents of register 13 are saved so that the registers may be restored upon return. For purposes of tracing from save area to save area, the address of the new save area is stored. Only the registers to be modified in the routine need be saved. However, the safest procedure is to store all registers to ensure that later changes to the program do not result in the modification of the contents of a register that was not saved.

CALL--CALL A PROGRAM

The CALL macro instruction passes control from a program to a specified entry point in another program. The program issuing the call macro instruction is the calling program. The program receiving control is the called program. The called program must be in main storage when the CALL macro instruction is executed. The called program is brought into main storage in one of two ways:

1. As part of the program issuing the CALL. In this case, the CALL macro instruction must specify an entry point by symbolic name. The phase containing that entry point is included in the phase containing the CALL macro instruction by the linkage editor.

2. As the phase specified by a LOAD macro instruction. In this case, the CALL macro instruction specifies register 15 (the entry-point register) into which was loaded the address of the program to be called. The LOAD macro instruction must precede the first CALL for that program.

Name	Operation	Operand
[name]	CALL	{entrypoint} {(15)} [,(parameter,...)]

entrypoint

Specifies the entry point to which control is passed. If the symbolic name of an entry point is written, an instruction

```
L 15,=V(entrypoint)
```

is generated as part of the macro expansion. The linkage editor makes the called program part of the calling program phase. The symbolic name must be either the name of a control section (CSECT) or an assembler language ENTRY statement operand in the called program. Control is given to the called program at this address.

If register 15 is specified, the entrypoint address should have been loaded into that register previously. The operand may be written as a self-defining value equal to 15 and enclosed in parentheses, inwhich case, the V-type address constant instruction is not generated. Control is given to the called program at the address in register 15. The (15) entrypoint operand and LOAD macro instruction combination is most useful when the same program is called many times during execution of the calling program, but is not needed in main storage throughout execution of the calling program.

If the CALL macro instruction is used and a symbolic name is written for the entry operand, the called program resides in storage throughout execution of the calling program. This wastes main storage if the called program is not needed during all of the calling program execution.

parameter

Specifies an address (relocatable or absolute expression) to be passed as a parameter to the called program. Terms in the address must not be indexed. The parameter operands must be written in a sublist, as shown in the format description. If one or more parameter operands are written, a problem program parameter list is generated. It consists of a fullword for each operand. Each fullword is aligned on a fullword boundary and contains the address to be passed in its three low-order bytes. The addresses must appear in the parameter list in the same order as in the macro instruction. When the called program is entered, register 1 (the parameter list register) contains the address of the problem program parameter list.

In the following examples, EX1 gives control to an entry point named ENT. EX2 gives control to an entry point whose address is contained in register 15. Two parameters, ABC and DEF, are passed.

Examples

```
EX1  CALL ENT
EX2  CALL (15),(ABC,DEF)
```

A typical macro expansion for the macro instruction CALL SUBR,(P1,P2...,Pn) is:

```
CNOP  2,4
NAME
L     15,=V(SUBR)
LA    14,*+6+4*n (return address)
BALR  1,15
DC    A(P1,P2...,Pn)
ORG   *-4
DC    X'80'
ORG
```

NAME is the symbol in the name field of the macro instruction. n is the number of fullwords in the parameter list. SUBR is the symbolic name of the entry point of the called program. P1 through Pn are the addresses to be passed to the called program.

SAVE--SAVE REGISTER CONTENTS

The SAVE macro stores the contents of specified registers in the savearea provided by the calling program. It is written at the entry point of a program, before any registers can be modified by the new program.

Name	Operation	Operand
[name]	SAVE	(r1[,r2])

The operands r1,r2 specify the range of the registers to be stored in the save area of the calling program. The address of this area is passed to the program in register 13. The operands are written as self-defining values so that they cause desired registers in the range of 14 through 12 (14, 15, 0 through 12) to be stored when inserted in an STM instruction. Registers 14 and 15, if specified, are saved in words 4 and 5 of the save area. Registers 0 through 12 are saved in words 6 through 18 of the save area. The contents of a given register are always stored in a particular word in the save area. For example, register 3 is always saved in word 9 even if register 2 is not saved. If r2 is omitted, only the register specified by r1 is saved.

RETURN--RETURN TO A PROGRAM

The RETURN macro instruction restores the registers whose contents were saved and returns control to the calling programs.

Name	Operation	Operand
[name]	RETURN	(r1[,r2])

The operands r1,r2 specify the range of the registers to be reloaded from the save area of the program that receives control. The operands are written as self-defining values. When inserted in an LM (load multiple) instruction, the operands cause the registers in the range from 14 through 12 (14, 15, 0 through 12) to be restored from words 4 through 18 of the save area. If r2 is omitted, only the register specified by r1 is restored. To access this save area, register 13 must contain the save area address. Therefore, the address of the save area is loaded into register 13 before execution of the RETURN macro instruction.

Appendix A: Label Formats

DASD Labels

Whenever files of records are written on DASD, each volume must contain standard labels to identify the pack or cell and the logical file(s) on it. When logical IOCS is used for a file, the IOCS routines read, check, and/or write standard labels. When physical IOCS is used, IOCS processes the labels if the DTFPH macro instruction is included in the user's program. The entry TYPEFLE must be specified to indicate whether the file is an input file (read and/or write labels) or an output file (check old labels and write new labels).

The standard labels include one volume label for each pack or cell and one or more file labels for each logical file on the DASD. The section describes briefly the organization of labels on disk packs or data cells. Additional information about labels is given in the Data Management Concepts publication.

VOLUME LABELS

The standard volume label identifies the entire volume and offers volume protection. For systems residence, the volume label is always the third record on cylinder 0, track 0. The first two records on this track of SYSRES are initial program loading (IPL) records. On all other volumes, these records contain binary zeros. The volume-label record consists of a count area, a 4-byte key area, and an 80-byte data area. Both the key area and the first four bytes of the data area contain the label identifier VOL1. The remaining 76 bytes of the data area contain other identifying information such as the volume serial number, and the address of the set of file labels for the pack or cell (see Standard File Labels). The volume label is generally written once, when the DASD is received, by an IBM-supplied utility program.

The standard volume label may be followed by one to seven additional volume labels (starting with record 4 on cylinder 0, track 0). These labels must contain the label identifier VOL2, VOL3, etc in the 4-byte key areas and in the first four bytes of the data areas. The other 76 bytes may contain whatever information the user requires. The additional volume labels are also written by the utility program that writes the standard volume label. However, IOCS does not make them available to the user for checking or rewriting when problem programs are executed. These labels are used by OS and are always bypassed by the Disk Operating System OPEN routines.

STANDARD FILE LABELS

The standard file labels identify the logical file, give its location(s) on the disk pack or data cell, and offer file protection. The labels for all logical files on a volume are grouped together and stored in a specific area of DASD called the Volume Table of Contents (VTOC).

The number and format of labels required for any one logical file depends on the file organization (see Standard File Label Formats) and the number of separate areas (extents) of the pack or cell used by the file. The data records for a logical file may be contained within one area of the pack or cell, or they may be scattered in different areas of it. The limits (starting and ending addresses) of each area used by the file are specified by the standard file label(s).

Because each file label contains file limits, the group of labels on the volume is essentially a directory of all files on the volume. The VTOC itself becomes a file of records (one or more standard-label records per logical file) and, in turn, has a label. The label of the VTOC is the first record in the VTOC. This label identifies the file as the VTOC file, and gives the file limits. The Volume Table of Contents is contained within one cylinder of a disk pack or data cell. It does not overflow onto another cylinder.

If a logical file of data records is recorded on more than one volume, standard labels for the file must be included in the VTOC of each volume used. The label(s) on each volume identifies the portion of the logical file on the pack or cell and specifies the extent(s) used on it.

STANDARD FILE LABEL FORMATS

All standard file label records have a count area and a 140-byte key/data area. Five standard label formats are provided.

Format 1. This format is for all logical files, and has a 44-byte key area and a 96-byte data area. It is always the first of the series of labels when a file requires more than one label on a disk pack or cell (see Format 2 and Format 3).

The format-1 label identifies the logical file (by a file name assigned by the user and included in the 44-byte key area), and contains file and data record specifications. It also provides the addresses for three separate DASD areas (extents) for the file. If the file is scattered over more than three separate areas on one pack or cell, a format-3 label is also required. In this case, the format-1 label points to the second label set up for the file on this volume. If a logical file is recorded on more than one volume, a format-1 label is always created in the VTOC for each volume.

Format 2. This format is required for any file that is organized by the Indexed Sequential Access Method. The 44-byte key area and the 96-byte data area contain specifications unique to this type of file organization.

If an indexed sequential file is recorded on two or more volumes, the format-2 label is used only on the volume containing the cylinder index. This volume may, or may not, contain data records. The format-2 label is not repeated on the additional packs (as the format-1 label is).

Format 3. If a logical file uses more than three extents on any pack or cell, this format specifies the addresses of the additional extents. It is used only for extent information. It has a 44-byte key area and a 96-byte data area that provide for 13 extents.

The format-1 label for the logical file points to the format-3 label. In a DTFSD file, a format-3 label may also point to another format-3 label. It is included, as required, on the first pack or cell, or on additional volumes if the logical file is recorded on two or more volumes.

Format 4. The format-4 label defines the VTOC itself and is always the first label in the VTOC. This label also provides the location and number of available tracks in the alternate track area.

Format 5. The format-5 label is used by the Operating System for Direct Access Device Space Management.

USER-STANDARD DASD FILE LABELS

The user may include additional labels to defines his file further, provided the file is processed sequentially (DTFSR or DTFSD macro specified), by the direct access method (DTFDA macro specified), or by physical IOCS (DTFPH macro specified). User standard file labels are not processed in a file organized and processed by the Indexed Sequential Access Method (DTFIS specified). A file that is processed in sequential order (using DTFSR or DTFSD) may have up to eight user header labels and trailer labels for a 2311, 2314, or 2319 file. It may have up to five user header labels and trailer labels for a 2321 file. The trailer labels can be written to indicate an end-of-volume or end-of-file condition. That is, when the end of an extent or end of file on one volume is reached and the next extent is on a different volume, user trailer labels can be included to contain whatever trailer information the user desires (for example, a record count for the completed volume).

User-standard labels are not stored in the Volume Table of Contents. Instead, they are written on the first track of the first extent allotted for the logical-file data records. In this case, the user's data records start with the second track in the extent, regardless of whether the labels require a full track. If a file is written on two or more packs or cells, the additional labels are written on each of the packs or cells.

All user-standard labels must be 80 bytes long, and must contain standard information in the first four bytes. The remaining 76 bytes may contain whatever information the user wants. A four-byte key area supplied by IOCS precedes all labels.

The standard information in the first four bytes is used for the record key when reading or writing header labels. The header labels are identified by UHL1, UHL2,...,UHL8. The trailer labels, when applicable, are identified in the key field by UTL0, UTL1,...,UTL7 (or UTL4) although the first four bytes of the labels contain

UTL1-UTL8 (or UTL5). Each user-label set (header or trailer) is terminated by a filemark (a record with data length 0 preceded by a four-byte key area), which is written by IOCS. For example, if a file has five header labels and four trailer labels, the contents of the user-label track are:

```
R0      Standard information
R1      UHL1--user's 1st header label
R2      UHL2--user's 2nd header label
R3      UHL3--user's 3rd header label
R4      UHL4--user's 4th header label
R5      UHL5--user's 5th header label
R6      UHL6--filemark
R7      UTL1--user's 1st trailer label
R8      UTL2--user's 2nd trailer label
R9      UTL3--user's 3rd trailer label
R10     UTL4--user's 4th trailer label
R11     UTL5--filemark
```

If only header labels are used, the user-label track contains:

```
R0      Standard information
R1      UHL1--user's 1st header label
R2      UHL2--user's 2nd header label
          .
          .
          .
R(n)    UHL(n)--user's nth header label
        where n is ≤ 8
R(n+1)  UHL(n+1)--filemark
R(n+2)  UTL0--filemark
```

The user's label routine can determine if a label is a header or trailer label by testing the first four bytes of the label (see Label Processing).

Standard Tape Labels

When an EBCDIC or ASCII tape input or output file with standard labels is opened, IOCS can handle the label checking (on input) or writing (on output). When logical IOCS macros are used, the entry FILABL=STD must be included to specify IOCS processing of labels. When physical IOCS macros are used, the DTFPH entry TYPEFLE must be included to indicate whether this is an input file (check labels) or an output file (write labels).

The standard labels for a tape file are: a volume label, a file header label, and a file trailer label. The volume label (the first record, 80 characters on a reel of tape) identifies the entire volume (reel) and offers volume protection. It contains the label identifier VOL1 in the first four positions, and other identifying information such as the volume serial number. This is a unique number generally assigned to the reel when it is first received in the installation. The volume label is generally written once, when the reel of tape is received, by an IBM-supplied utility program. The standard volume label may be followed by a maximum of seven additional volume labels, if desired. These must be identified by VOL2, VOL3, etc in the first four positions of each succeeding label. However, IOCS does not permit the checking or writing of additional volume labels by the user in the problem program. These labels are available for use by OS and are always bypassed on the input for the Disk Operating System.

Only one American National Standards Institute, Inc. standard volume label can be used for ASCII tape files. This may be followed by a maximum of nine user standard volume labels (identified by UVL1,...UVL9 in the first four positions of the label). These additional labels are bypassed on input and are not created by DOS on output.

The volume label set is followed by a standard file header label. This label (80 characters) identifies the logical file record on the tape and offers file protection. It contains the label identifier HDR1 in the first four positions, and other identifying information such as file identifier, file serial number, creation date, etc. An input tape may contain standard header labels HDR1-HDR8. IOCS checks label HDR1 and bypasses HDR2-HDR8 (these labels are used by OS). An ASCII input tape may contain American National Standards Institute, Inc. standard header labels HDR1 through HDR9. IOCS checks label HDR1 and bypasses HDR2-HDR9.

For EBCDIC files, the standard file header labels may be followed by a maximum of eight user-written standard labels, if desired. If so, the file header labels must be identified by UHL1, UHL2, etc. Labels UHL1-UHL8 may be processed if the DTF entry LABADDR is specified. A tapemark follows the last file header label.

A standard file trailer label is located at the end of a logical file (EOF), or at the end of a volume (EOV) if the logical file continues on another volume. The trailer label has the same format as the header label. It is identified by EOF1 or EOV1 (instead of HDR1) and contains a physical record count (block count). Like

the file header label, the standard file trailer label may be followed by user standard trailer labels. These must be identified by UTL1, UTL2, etc (for EBCDIC files). A tapemark must follow the label set.

American National Standards Institute, Inc. user standard header and trailer labels are identified by UHLa and UTLa respectively. a represents an ASCII character in the range 2/0 through 5/14 excluding 2/7 (quote).

All user-written standard labels must be 80 characters long and must contain the standard identification in the first four positions. The remaining 76 positions may contain whatever information the user wants. Additional information about tape labels is given in the Data Management Concepts publication.

Note 1: On 7-track tape, standard labels are written on the same density as the data on the tape. All information on a tape reel must be written in single density. These standard labels are written with even parity in the translation mode.

Note 2: The last file on any volume is followed by two consecutive tapemarks.

Appendix B: Control Character Codes

CTLCHR=ASA

A control character must appear in each logical record if the ASA option is chosen. If the control character for the printer is not valid, a message is given and the job is canceled. If the control character for the card punch is not V or W, the card is selected into pocket 1. The codes are:

Code	Interpretation
(blank)	Space one line before printing
0	Space two lines before printing
-	Space three lines before printing
+	Suppress space before printing
1	Skip to channel 1 before printing
2	Skip to channel 2 before printing
3	Skip to channel 3 before printing
4	Skip to channel 4 before printing
5	Skip to channel 5 before printing
6	Skip to channel 6 before printing
7	Skip to channel 7 before printing
8	Skip to channel 8 before printing
9	Skip to channel 9 before printing
A	Skip to channel 10 before printing
B	Skip to channel 11 before printing
C	Skip to channel 12 before printing
V	Select stacker 1
W	Select stacker 2

CTLCHR=YES

The control character is the command-code portion of the System/360 channel command word used in printing a line or spacing the forms. If the character is not one of the following characters, unpredictable events will occur.

Hexadecimal Code	Punch Combination	Function
Stacker Selection on 1442		
81	12,0,1	Select into stacker 1
C1	12,1	Select into stacker 2
Pocket Selection on 2540		
01	12,9,1	Select into pocket 1
41	12,0,9,1	Select into pocket 2
81	12,0,1	Select into pocket 3
Stacker Selection on 2520		
01	12,9,1	Select into stacker 1
41	12,0,9,1	Select into stacker 2
Printer Control		
01	12,9,1	Write (no automatic space)
09	12,9,8,1	Write and space 1 line after printing
11	11,9,1	Write and space 2 lines after printing
19	11,9,8,1	Write and space 3 lines after printing
89	12,0,9	Write and skip to channel 1 after printing

Hexa-decimal Code	Punch Combination	Function
91	12,11,1	Write and skip to channel 2 after printing
99	12,11,9	Write and skip to channel 3 after printing
A1	11,0,1	Write and skip to channel 4 after printing
A9	11,0,9	Write and skip to channel 5 after printing
B1	12,11,0,1	Write and skip to channel 6 after printing
B9	12,11,0,9	Write and skip to channel 7 after printing
C1	12,1	Write and skip to channel 8 after printing
C9	12,9	Write and skip to channel 9 after printing
D1	11,1	Write and skip to channel 10 after printing
D9	11,9	Write and skip to channel 11 after printing
E1	11,0,9,1	Write and skip to channel 12 after printing

Hexa-decimal Code	Punch Combination	Function
0B	12,9,8,3	Space 1 line immediately
13	11,9,3	Space 2 lines immediately
1B	11,9,8,3	Space 3 lines immediately
8B	12,0,8,3	Skip to channel 1 immediately
93	12,11,3	Skip to channel 2 immediately
9B	12,11,8,3	Skip to channel 3 immediately
A3	11,0,3	Skip to channel 4 immediately
AB	11,0,8,3	Skip to channel 5 immediately
B3	12,11,0,3	Skip to channel 6 immediately
BB	12,11,0,8,3	Skip to channel 7 immediately
C3	12,3	Skip to channel 8 immediately
CB	12,0,9,8,3	Skip to channel 9 immediately
D3	11,3	Skip to channel 10 immediately
DB	12,11,9,8,3	Skip to channel 11 immediately
E3	0,3	Skip to channel 12 immediately
03	12,9,3	No operation

Appendix C: Assembling the Problem Program, DTFs, and Logic Modules

All the programs described in this appendix perform the same function, namely, a card-to-disk operation with the following equipment and options:

1. Card reader: IBM 2540 (SYS004).
2. Disk: IBM 2311 with user labels.
3. Record size: 80 bytes.
4. Block size: 408 bytes including 8-byte count field (blocking factor of 5).
5. One I/O area and workarea for the card reader.
6. Two I/O areas for the disk.

The following methods may be used to furnish the DTFs and IOCS logic modules to the card-to-disk program.

1. DTFs, IOCS logic modules, and problem program assembled together.
2. Logic modules assembled separately.
3. DTFs and logic modules assembled separately, label exit, EOF exit, and I/O areas assembled with DTFs.
4. Same as in 3 except that I/O areas are moved back into main program.
5. Same as in 4 except that label exit and EOF exit are also moved back into main program.

An example of each of these five methods of assembling the main program, modules, DTFs, and related functions follows. In the figures that accompany the examples, each dashed arrow represents a symbolic linkage, with an external reference at the base of the arrow, and a label or section definition designating the same symbol at the head of the arrow.

At the points where an arrow is marked with a circle, it is the programmer's responsibility to define an ENTRY or EXTRN symbol, as applicable.

Each dotted arrow represents a direct linkage. Components are represented by the small rectangles. Assemblies are represented by the larger bordered areas.

The examples are followed by a Comparison of the Five Methods.

EXAMPLE 1: ASSEMBLING THE PROBLEM PROGRAM, DTFs, AND LOGIC MODULES TOGETHER

Figure 65 shows the assembly of the DTFs, logic modules, and problem program. The assembly source deck is:

```
                                                          Col. 72

CDTODISK  START   0
          BALR    12,0                          Initialize base register.
          USING   *,12                          Establish addressability.
          LA      13,SAVEAREA                   Use reg 13 as pointer to save area.
          OPEN    CARDS,DISK                    Open both files.
NEXT      GET     CARDS,(2)                     Read one card and move it
          PUT     DISK                          to the disk output buffer.
          B       NEXT                          Return for next card.
SAVEAREA  DS      9D                            Save area is 72-byte, doubleword
                                                aligned.

EOFCD     CLOSE   CARDS,DISK                    At card-reader EOF, close
          EOJ                                   both files and exit to job
                                                control.

MYLABELS  .                                     User's label-processing
          .                                     routine.
          LBRET   2                             Return to main program.

CARDS     DTFCD                            X
                  DEVADDR=SYS004,          X
                  EOFADDR=EOFCD,           X
                  IOAREA1=A1,              X
                  WORKA=YES

DISK      DTFSD                            X
                  BLKSIZE=408,             X
                  IOAREA1=A2,              X
                  IOAREA2=A3,              X
                  IOREG=(2),               X
                  LABADDR=MYLABELS,        X
                  RECFORM=FIXBLK,          X
                  RECSIZE=80,              X
                  TYPEFLE=OUTPUT

A1        DS      80C                           Card-input buffer
A2        DS      408C                          First disk buffer
A3        DS      408C                          Second disk buffer

(1)       CDMOD                            X
                  DEVICE=2540,             X
                  TYPEFLE=INPUT,           X
                  WORKA=YES

          SDMODFO

          END     CDTODISK                      Program-start address
```

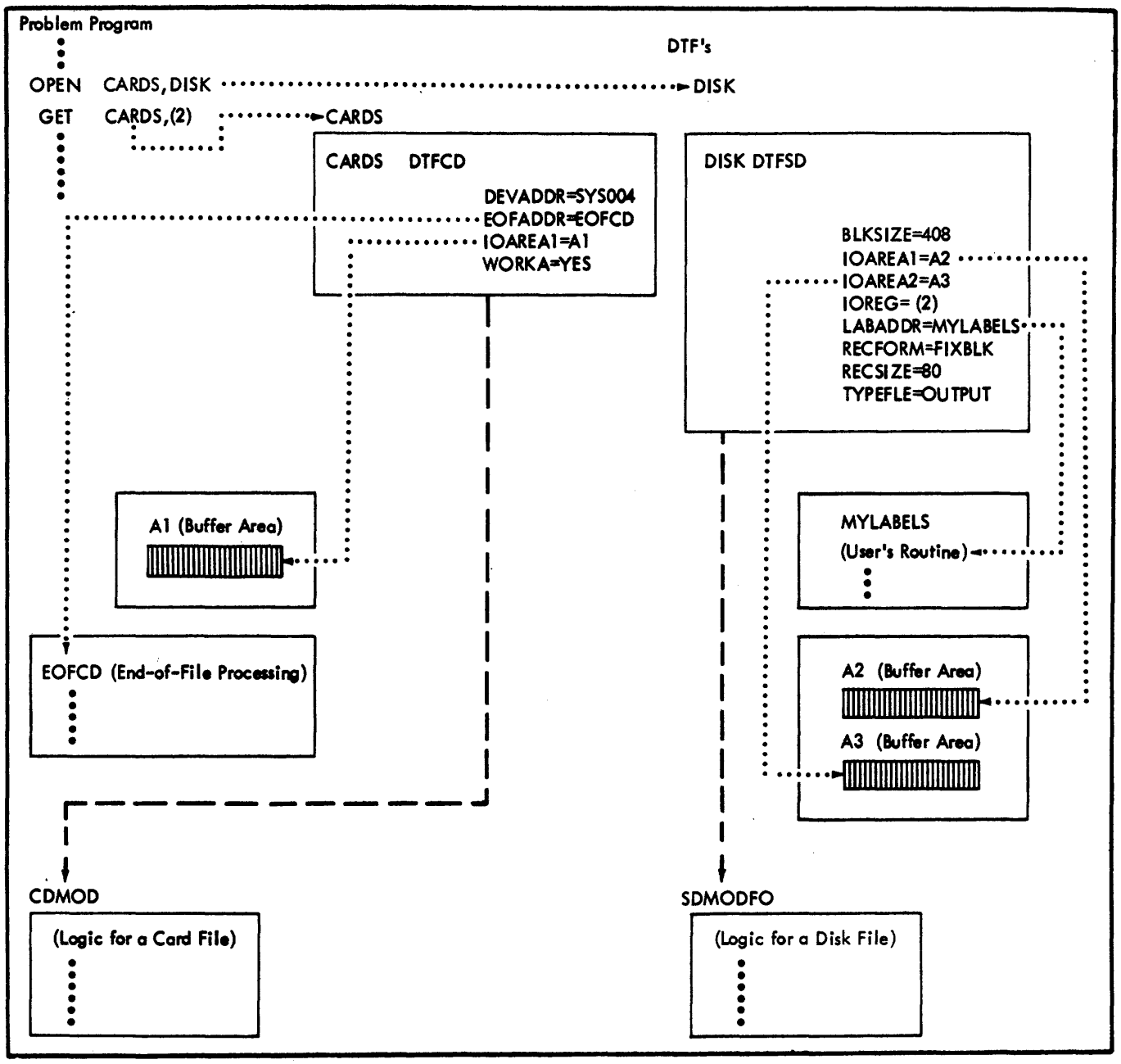


Figure 65. Assembling the Problem Program DTFs and Modules Together (Example 1)

EXAMPLE 2: ASSEMBLING THE LOGIC MODULES SEPARATELY

The main-program source deck is identical to that in Example 1 until (1); at this point, the user simply furnishes the END card. Figure 66 shows the separation of the I/O logic modules.

The two logic modules are assembled as follows:

Card logic module

```
                                                    Col. 72
         CDMOD                                        X
               DEVICE=2540,                           X
               SEPASMB=YES,                           X
               TYPEFLE=INPUT,                         X
               WORKA=YES
         END
```

Disk logic module

```
         SDMODFO                                      X
               SEPASMB=YES
         END
```

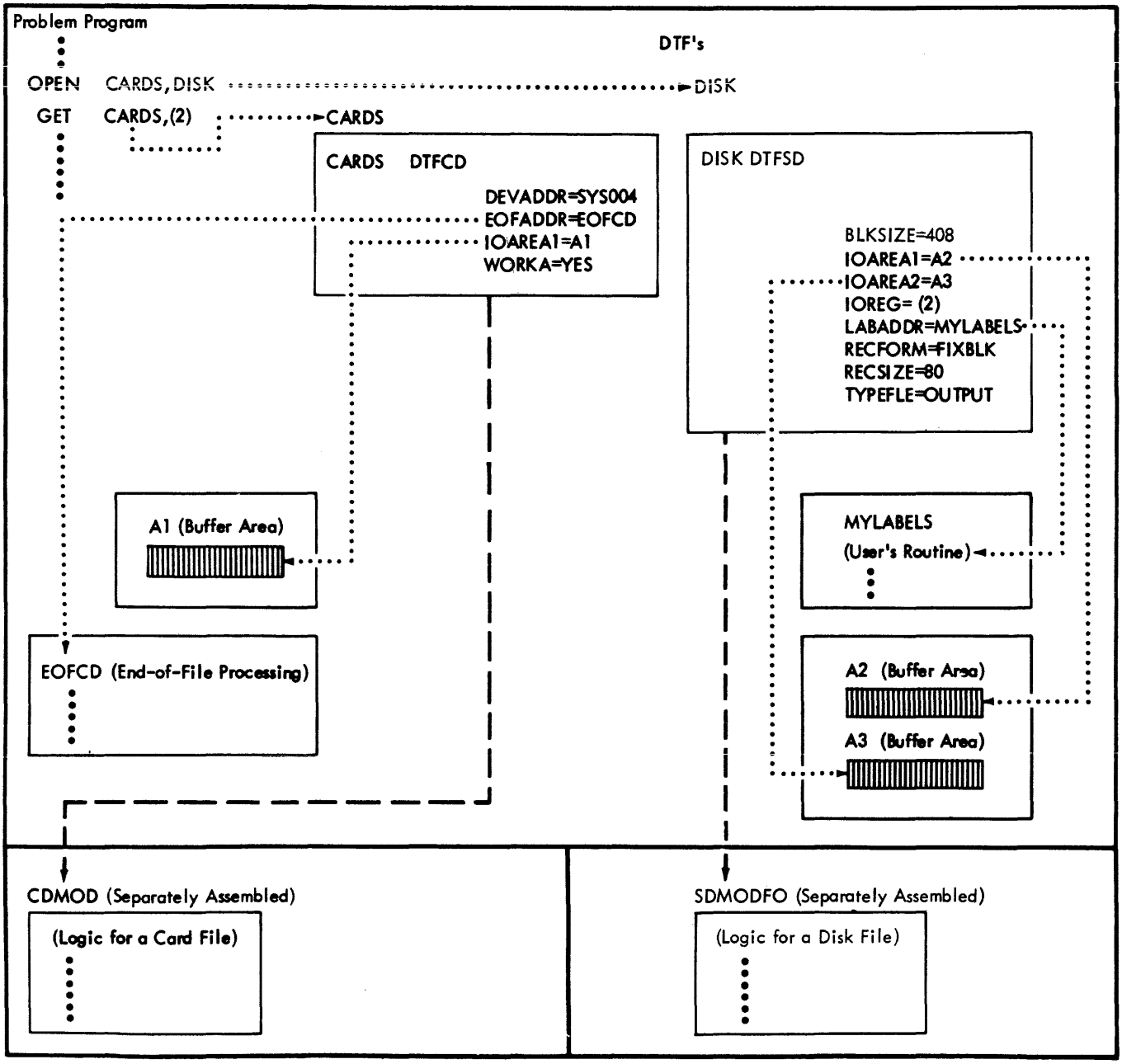


Figure 66. Logic Modules Assembled Separately (Example 2)

After assembly, each logic module is preceded by the appropriate CATALR card. The modules may be added to the system relocatable library during a maintenance run. Thereafter, logic modules are automatically included in the user program by the linkage editor while it prepares the preceding main program for execution.

EXAMPLE 3: ASSEMBLING THE DTFs AND LOGIC MODULES SEPARATELY

The main program is assembled:

```
      CDTODISK   START   0
                 BALR    12,0
                 USING   *,12
                 LA      13,SAVEAREA
                 OPEN    CARDS,DISK
      NEXT       GET     CARDS,(2)
                 PUT     DISK
                 B       NEXT
      SAVEAREA   DS      9D

(2)              EXTRN   CARDS,DISK
                 END     CDTODISK
```

The logic modules are assembled as in Example 2. Figure 67 shows the separation of the DTFs and logic modules.

The card-file macro instruction and related functions are assembled:

```
                                                    Col. 72

      CARDS    DTFCD                              X
                     DEVADDR=SYS004,              X
                     SEPASMB=YES,                 X
                     EOFADDR=EOFCD,               X
                     IOAREA1=A1,                  X
                     WORKA=YES
               USING  *,14


      EOFCD    CLOSE  CARDS,DISK
               EOJ


               EXTRN DISK


(3)   A1       DS     80C
               END
```

The disk-file macro instruction and related functions are assembled:

```
                                                    Col. 72

      DISK     DTFSD                              X
                     BLKSIZE=408,                 X
                     SEPASMB=YES,                 X
                     .
                     .
                     .
                     TYPEFLE=OUTPUT
      MYLABELS BALR  10,0
               USING *,10
               .
               .
               LBRET 2

(4) A2         DS    408C
(5) A3         DS    408C
               END
```

In the card-file and the disk-file assemblies, a USING statement was added because certain user routines are segregated from the main program and moved into the DTF assembly.

When user routines, such as error, label processing, or EOF routines, are segregated from the main program, it is necessary to establish addressability for these routines. The user can provide this addressability by assigning and initializing a base register. In the special case of the EOF routine, the addressability is established by logical IOCS in register 14. For error exits and label processing routines, however, this addressability is not supplied by logical IOCS. Therefore, if the user segregates his error routines, it is his responsibility to establish addressability for them.

Figure 69 contains the printer output to show how the coding of Example 3 would look when assembled.

In Figure 68, the standard name for the logic modules was generated: statement 13 of the DTFCD--V(IJCFZIWO), and statement 12 of the DTFSD--V(IJGFOZZZ). These module names appear in the External Symbol Dictionary of each of the respective logic module assemblies.

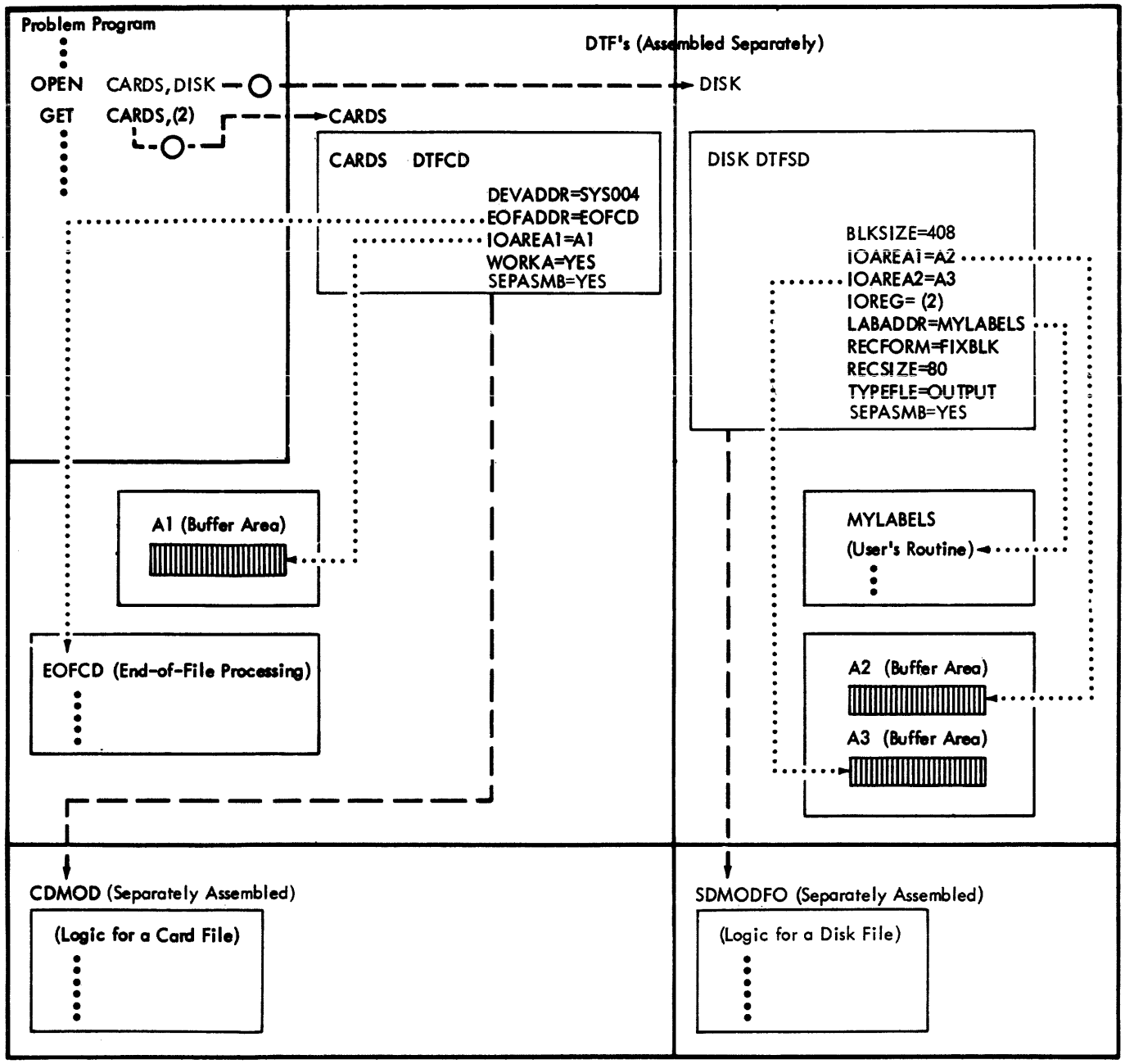


Figure 67. Logic Modules and DTFs Assembled Separately (Example 3)

```
MAIN PROGRAM                                                                    PAGE    1
                                  EXTERNAL SYMBOL DICTIONARY
 SYMBOL   TYPE ID  ADDR   LENGTH LD ID

CDTODISK  SD  01 000000 000090     Section definition.   Control section defined by START statement.
CARDS     ER  02                   External reference.  }
DISK      ER  03                   External reference.  } Defined by EXTRN statement.
```

```
                                  EXAMPLE 3                                     PAGE    1

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                          DD200CT66 10/26/66

000000                               1 CDTODISK START 0
000000 05C0                          2          BALR  12,0                INITIALIZE BASE REGISTER
000002                               3          USING *,12                ESTABLISH ADDRESSABILITY
000002 41D0 C036            00038    4          LA    13,SAVEAREA         USE REGISTER 13 AS POINTER TO SAVE
                                     5          OPEN  CARDS,DISK          OPEN BOTH FILES
                                     6+* SYSTEM CONTROL AND BASIC IOCS 360N-CL-453 CHANGE LEVEL 2-0
000006 0700                          7+         CNOP  0,4
000008                               8+         DC    0F'0'
000008 4110 C07E            00080    9+         LA    1,=C'$$BOPEN '
00000C 4500 C016            00018   10+IJJ00001 BAL   0,*+4+4*(3-1)
000010 00000000                     11+         DC    A(CARDS)
000014 00000000                     12+         DC    A(DISK)
000018 0A02                         13+         SVC   2
                                    14 NEXT     GET   CARDS,(2)           READ ONE CARD, MOVE TO WORK AREA
                                    15+* CHANGE LEVEL 2-0
00001A 5810 C086            00088   16+NEXT     L     1,=A(CARDS) GET DTF TABLE ADDRESS
00001E 1802                         17+         LR    0,2 GET WORK AREA ADDRESS
000020 58F1 0010            00010   18+         L     15,16(1) GET LOGIC MODULE ADDRESS
000024 45EF 0008            00008   19+         BAL   14,8(15) BRANCH TO GET ROUTINE
                                    20          PUT   DISK                WRITE ON DISK
                                    21+* CHANGE LEVEL 2-0
000028 5810 C08A            0008C   22+         L     1,=A(DISK) GET DTF TABLE ADDRESS
00002C 58F1 0010            00010   23+         L     15,16(1) GET LOGIC MODULE ADDRESS
000030 45EF 000C            0000C   24+         BAL   14,12(15) BRANCH TO PUT ROUTINE
000034 47F0 C018            0001A   25          B     NEXT                GO FOR NEXT CARD
000038                              26 SAVEAREA DS    9D                  72-BYTE SAVE AREA
                                    27          EXTRN CARDS,DISK
000000                              28          END   CDTODISK
000080 5B5BC2D6D7C5D540             29                =C'$$BOPEN
000088 00000000                     30                =A(CARDS)
00008C 00000000                     31                =A(DISK)
```

Figure 68. Separate Assemblies, (Example 3) (Part 1 of 4)

```
DTFCD ASSEMBLY
                                    EXTERNAL SYMBOL DICTIONARY                                    PAGE    1
 SYMBOL  TYPE ID  ADDR  LENGTH LD ID

CARDSC    SD  01 000000 0000A0          Section definition.          } Generated by specifying SEPASMB=YES in DTFCD macro instruction.
CARDS     LD     000000           01    Label definition (entry point). }
IJCFZIWO  ER  02                        External reference.  Corresponds to V-type address constant generated in DTFCD.
DISK      ER  03                        External reference.  Defined by EXTRN statement.
```

```
                                          EXAMPLE 3                                               PAGE    1

 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                     DD20OCT66 10/26/66

                                    1 CARDS     DTFCD DEVADDR=SYS004,                                   X
                                                      SEPASMB=YES,                                      X
                                                      EOFADDR=EOFCD,                                    X
                                                      IOAREA1=A1,                                       X
                                                      WORKA=YES
                                    2+* SYSTEM CONTROL AND BASIC IOCS 360N-CL-453 CHANGE LEVEL 2-0
                                    3+          PUNCH '          CATALR CARDS'
000000                              4+CARDSC    CSECT
                                    5+          ENTRY CARDS
000000                              6+          DC    0D'0'
000000 000080000000                 7+CARDS     DC    X'000080000000' RES. COUNT,COM. BYTES,STATUS BTS
000006 01                           8+          DC    AL1(1) LOGICAL UNIT CLASS
000007 04                           9+          DC    AL1(4) LOGICAL UNIT
000008 00000020                    10+          DC    A(IJCX0001) CCW ADDRESS
00000C 00000000                    11+          DC    4X'00' CCB-ST BYTE,CSW CCW ADDR.
000010 00                          12+          DC    AL1(0)
000011 000000                      13+          DC    VL3(IJCFZIWO) ADDRESS OF LOGIC MODULE
000014 02                          14+          DC    X'02' DTF TYPE (READER)
000015 01                          15+          DC    AL1(1) SWITCHES
000016 02                          16+          DC    AL1(2) NORMAL COMM.CODE
000017 02                          17+          DC    AL1(2) CNTROL COMM.CODE
000018 00000048                    18+          DC    A(A1) ADDR. OF IOAREA1
00001C 00000034                    19+          DC    A(EOFCD) EOF ADDRESS
000020 0200004820000050            20+IJCX0001 CCW   2,A1,X'20',80
000028 4700 0000           00000   21+          NOP   0 LOAD USER POINTER REG.
00002C D24F D000 E000 00000 00000  22+          MVC   0(80,13),0(14) MOVE IOAREA TO WORKA
000032                             23+IJJZ0001 EQU   *
000032                             24           USING *,14               ESTABLISH ADDRESSABILITY
                                   25 EOFCD     CLOSE CARDS,DISK         END OF FILE ADDRESS FOR CARD READER
                                   26+* CHANGE LEVEL 2-0
000032 0700                        27+          CNOP  0,4
000034                             28+EOFCD     DC    0F'0'
000034 4110 E066           00098   29+          LA    1,=C'$$BCLOSE'
000038 4500 E012           00044   30+IJJC0002 BAL   0,*+4+4*(3-1)
00003C 00000000                    31+          DC    A(CARDS)
000040 00000000                    32+          DC    A(DISK)
000044 0A02                        33+          SVC   2
                                   34           EOJ
                                   35+* CHANGE LEVEL 2-0
000046 0A0E                        36+          SVC   14
                                   37           EXTRN DISK
000048                             38 A1        DS    80C                CARD I/O AREA
                                   39           END
000098 5B5BC2C3D3D6E2C5            40                 =C'$$BCLOSE'
```

Figure 68. Separate Assemblies, (Example 3) (Part 2 of 4)

```
DTFSD ASSEMBLY                          EXTERNAL SYMBOL DICTIONARY                    PAGE    1
 SYMBOL   TYPE ID  ADDR   LENGTH LD ID

DISKC     SD  01 000000 0003D4       Section definition.                  }Generated by specifying SEPASMB=YES in DTFSD macro instruction.
DISK      LD     000000         01   Label definition (entry point).      }
IJGFOZZZ  ER  02                     External reference. Corresponds to V-type address constant generated in DTFSD.
```

```
                                        EXAMPLE 3                                     PAGE    1

LOC   OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                          DD20OCT66 10/26/66

                                     1 DISK     DTFSD BLKSIZE=408,                                X
                                                      SEPASMB=YES,                                 X
                                                      IOAREA1=A2,                                  X
                                                      IOAREA2=A3,                                  X
                                                      IOREG=(2),                                   X
                                                      LABADDR=MYLABELS,                            X
                                                      RECFORM=FIXBLK,                              X
                                                      RECSIZE=80,                                  X
                                                      TYPEFLE=OUTPUT
                                     2+* CONSECUTIVE DISK PROCESSING IOCS 360N-10-455 CHANGE LEVEL 2-0
                                     3+         PUNCH '         CATALR DISK'
000000                               4+DISKC    CSECT
                                     5+         ENTRY DISK
000000                               6+         DC    0D'0'
000000 000080000000                  7+DISK     DC    X'000080000000' CCB
000006 FF                            8+         DC    AL1(255) LOGICAL UNIT CLASS
000007 FF                            9+         DC    AL1(255) LOGICAL UNIT NUMBER
000008 00000068                     10+         DC    A(IJGC0001) CCB-CCW ADDRESS
00000C 00000000                     11+         DC    4X'00' CCB-ST BYTE,CSW CCW ADDRESS
000010 00                           12+         DC    AL1(0)
000011 000000                       13+         DC    VL3(IJGFOZZZ) LOGIC MODULE ADDRESS
000014 20                           14+         DC    X'20' DTF TYPE
000015 49                           15+         DC    AL1(73) OPEN/CLOSE INDICATORS
000016 C4C9E2D240404040             16+         DC    CL8'DISK' FILENAME
00001E 000000000000                 17+         DC    6X'00' BCCHHR ADDR OF F1 LABEL IN VTOC
000024 0000                         18+         DC    2X'00' VOL SEQ NUMBER
000026 08                           19+         DC    X'08' OPEN COMMUNICATIONS BYTE
000027 00                           20+         DC    X'00' XTENT SEQ NO OF CURRENT EXTENT
000028 00                           21+         DC    X'00' XTENT SEQ NO LAST XTENT OPENED
000029 0000A0                       22+         DC    AL3(MYLABELS) USER'S LABEL ADDRESS
00002C 000000A4                     23+         DC    A(A2) ADDRESS OF IOAREA
000030 80000000                     24+         DC    X'80000000' CCHH ADDR OF USER LABEL TRACK
000034 0000                         25+         DC    2X'00' LOWER HEAD LIMIT
000036 00000000                     26+         DC    4X'00' XTENT UPPER LIMIT
00003A 0000                         27+DISKS    DC    2X'00' SEEK ADDRESS-BB
00003C 0000FF00                     28+         DC    X'0000FF00' SEARCH ADDRESS-CCHH
000040 00                           29+         DC    X'00' RECORD NUMBER
000041 00                           30+         DC    X'00' KEY LENGTH
000042 0190                         31+         DC    H'400' DATA LENGTH
000044 00000000                     32+         DC    4X'00' CCHH CONTROL FIELD
000048 06                           33+         DC    AL1(6) R    CONTROL FIELD
000049 00                           34+         DC    X'00' SWITCHES
00004A 018F                         35+         DC    H'399' SIZE OF BLOCK-1
00004C 0000000000                   36+         DC    5X'00' CCHHR BUCKET
000051 00                           37+         DC    X'00'
000052 0E29                         38+         DC    H'3625' TRACK CAPACITY CONSTANT
000054 5821 0058          00058     39+         L     2,88(1) LOAD USER'S IOREG
000058 000000AC                     40+         DC    A(A2+8) DEBLOCKER-INITIAL POINTER
00005C 00000050                     41+         DC    F'80' DEBLOCKER-RECORD SIZE
000060 0000023B                     42+         DC    A(A2+8+400-1) DEBLOCKER-LIMIT
000064 0A                           43+         DC    AL1(10) LOGICAL INDICATORS
000065 000000                       44+         DC    AL3(0) USER'S ERROR ROUTINE
000068 0700003A40000006             45+IJGC0001 CCW   7,*-46,64,6 SEEK
000070 3100003C40000005             46+         CCW   X'31',*-52,64,5 SEARCH ID EQUAL
000078 0800007000000000             47+         CCW   8,*-8,0,0 TIC
```

Figure 68. Separate Assemblies, (Example 3) (Part 3 of 4)

DTFSD (Continued)

```
                                        EXAMPLE 3                                                    PAGE    2

 LOC   OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                        DD20OCT66 10/26/66

000080 1D00023C00000198              48+          CCW   X'1D',A3,0,400+8 WRITE COUNT KEY AND DATA
000088 3100003C40000005              49+          CCW   X'31',DISKS+2,64,5 SEARCH ID EQUAL
000090 0800008800000000              50+          CCW   8,*-8,0,0 TIC
000098 1E00009830000001              51+          CCW   30,*,48,1 VERIFY
0000A0                               52+IJJZ0001 EQU   *
0000A0 05A0                          53 MYLABELS BALR  10,0              INITIALIZE BASE REGISTER
0000A2                               54          USING *,10              ESTABLISH ADDRESSABILITY
                                     55 *        .                       USER'S LABEL PROCESSING ROUTINE
                                     56 *        .
                                     57          LBRET 2                 RETURN TO IOCS
                                     58+* CHANGE LEVEL 2-0
0000A2 0A09                          59+         SVC   9 BRANCH BACK TO IOCS
0000A4                               60 A2       DS    408C              FIRST DISK I/O AREA
00023C                               61 A3       DS    408C              SECOND DISK I/O AREA
                                     62          END
```

CDMOD ASSEMBLY

EXTERNAL SYMBOL DICTIONARY

SYMBOL	TYPE	ID	ADDR	LENGTH	LD ID	
IJCFZIW0	SD	01	000000	000060		Section definition. CSECT name generated by CDMOD macro instruction.

EXAMPLE 3

```
 LOC   OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                      2          PRINT NOGEN
                                      3          CDMOD                                                X
                                                       DEVICE=2540,                                    X
                                                       SEPASMB=YES,                                    X
                                                       TYPEFLE=INPUT,                                  X
                                                       WORKA=YES
                                     73          END
```

SDMODFO ASSEMBLY

EXTERNAL SYMBOL DICTIONARY

SYMBOL	TYPE	ID	ADDR	LENGTH	LD ID	
IJGFOZZZ	SD	01	000000	0001D4		Section definition. CSECT name generated by SDMODFO macro instruction.

EXAMPLE 3

```
 LOC   OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                      2          PRINT NOGEN
                                      3          SDMODFO                                              X
                                                       SEPASMB=YES
                                    169          END
```

Figure 68. Separate Assemblies, (Example 3) (Part 4 of 4)

The DTF assembly generates a table that contains no executable code. Each of the DTF tables is preceded by the appropriate CATALR card. These two object decks can be cataloged into the relocatable library together with the logic modules:

```
// JOB CATRELOC
// EXEC MAINT
   (DTFCD Assembly)
   (DTFSD Assembly)
   (CDMOD Assembly)
   (SDMODFO Assembly)
/*
```

Alternately, the object decks from these assemblies (DTF tables and logic modules) can be furnished to the linkage editor along with the main program object deck. The sequence follows:

```
// JOB CATALOG
// OPTION CATAL
   INCLUDE
   PHASE name,*
   (Object deck, main program)
   (Object deck, DTFCD assembly)
   (Object deck, DTFSD assembly)
   (Object deck, CDMOD assembly)
   (Object deck, SDMODFO assembly)
/*
// EXEC LNKEDT
/&
```

Note: It is not necessary to remove the CATALR card because the linkage editor bypasses it.

EXAMPLE 4: DTFs AND LOGIC MODULES ASSEMBLED SEPARATELY, I/O AREAS WITH MAIN PROGRAM

The main program is identical to Example 3 except the following four cards are inserted after the card marked (2):

```
A1      DS      80C
A2      DS      408C
A3      DS      408C
        ENTRY   A1,A2,A3
```

The separate assembly of logic modules is identical to Example 3.

In the card-file assembly of Example 3, replace the card marked (3) with the following card:

```
        EXTRN A1
```

Similarly, in the disk-file assembly of the previous example, replace the cards marked (4) and (5) with the following card:

```
        EXTRN A2,A3
```

Figure 69 shows the separation of the logic modules, DTFs and I/O areas.

EXAMPLE 5: ASSEMBLING DTFs AND LOGIC MODULES SEPARATELY: I/O AREAS, LABEL EXIT, AND END-OF-FILE EXIT WITH MAIN PROGRAM

In addition to the changes in Example 4, the label exit and the end-of-file exit may be assembled separately. Figure 70 shows these separate assemblies. The main program is assembled:

```
CDTODISK   START   0
           BALR    12,0
           USING   *,12
           LA      13,SAVEAREA
           OPEN    CARDS,DISK
NEXT       GET     CARDS,(2)
           PUT     DISK
           B       NEXT
SAVEAREA   DS      9D

EOFCD      CLOSE   CARDS,DISK
           EOJ

MYLABELS   .
           .
           .
           LBRET   2

           EXTRN   CARDS,DISK
A1         DS      80C
A2         DS      408C
A3         DS      408C
           ENTRY   A1,A2,A3,EOFCD,MYLABELS
           END     CDTODISK
```

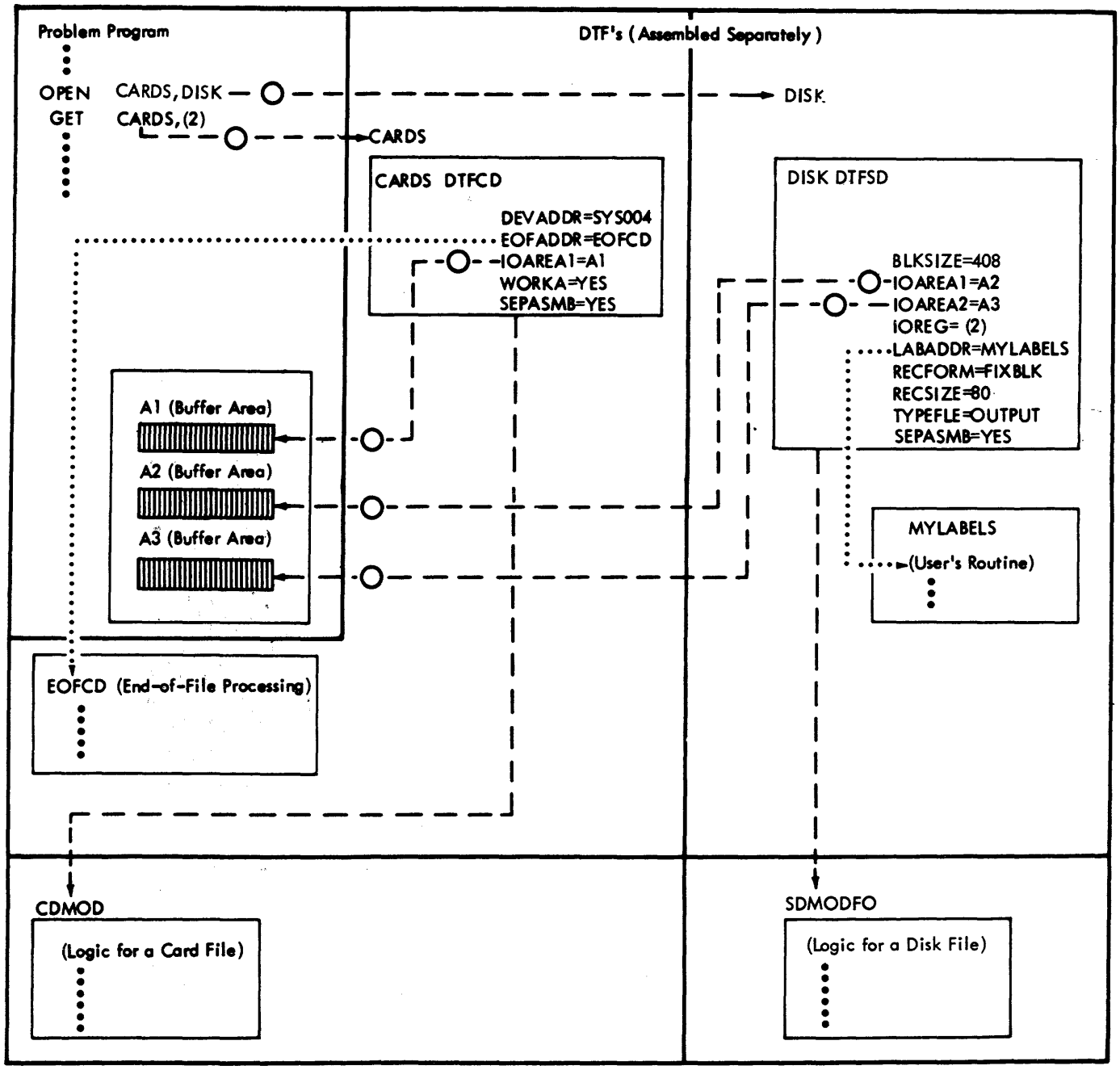


Figure 69. Logic Modules and DTFs Assembled Separately, I/O Areas with Main Program (Example 4)

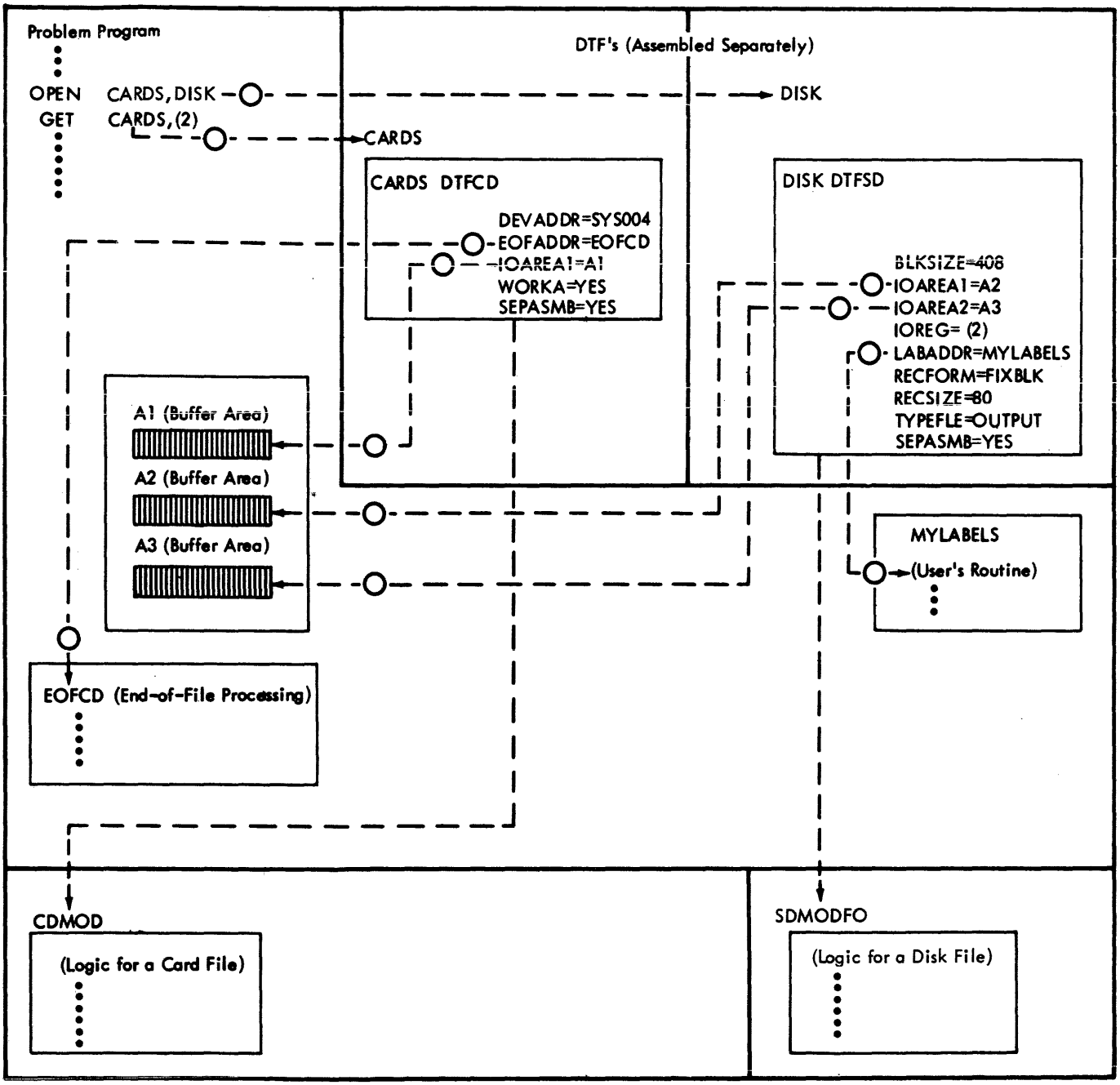


Figure 70. DTFs, and Logic Modules Assembled Separately; I/O Areas Label Exit, EOF Exit with Main Program (Example 5)

The file definition instructions are separately assembled:

```
                                                    Col. 72
CARDS    DTFCD   DEVADDR=SYS004,          X
                 WORKA=YES,               X
                 EOFADDR=EOFCD,           X
                 SEPASMB=YES,             X
                 IOAREA1=A1
         EXTRN   EOFCD,A1
         END

DISK     DTFSD   BLKSIZE=408,             X
                 TYPEFLE=OUTPUT,          X
                 SEPASMB=YES,             X
                 .
                 .
                 .
                 .
                 IOAREA1=A2,              X
                 IOAREA2=A3
         EXTRN   A2,A3,MYLABELS
         END
```

The separate assembly of logic modules is identical to Example 3 and Example 4.

Comparison of tne Five Methods

Example 1 requires the most assembly time and the least linkage-edit time. Because the linkage editor is substantially faster than the assembler, frequent reassembly of this program requires more total time for program preparation than Examples 2 through 5.

Example 2 segregates the IOCS logic modules from the remainder of the program. Because these modules are generalized, they can serve several different applications. Thus, they are normally retained in the system relocatable library for ease of access and maintenance.

When a system pack is generated or when it requires maintenance, the IOCS logic modules that are required for all applications should be identified and generated onto it. Each such module requires a separate assembly and a separate catalog operation, as shown in Examples 2 through 5. Many assemblies, however, can be batched together as can many catalog operations.

Object programs produced by COBOL, PL/I, and RPG require one or more IOCS logic modules in each executable program. These modules are usually assembled (as in Example 2) during generation of a system pack and are permanently cataloged into the system relocatable library.

Example 3 shows how a standardized IOCS package can be separated almost totally from a main program. Only the imperative IOCS macro instructions remain: OPEN, CLOSE, GET, and PUT. All file parameters, label processing, other IOCS exits, and buffer areas are preassembled. If there are few IOCS changes in an application compared to other changes, this method reduces to a minimum the total development/maintenance time. This approach also serves to standardize file descriptions so that they can be shared among several different applications. This reduces the chance of one program creating a file that is improperly accessed by subsequent programs. In Example 3, the user need only be concerned with the record format and the general register pointing to the record. He can virtually ignore the BLKSIZE, LABADDR, etc parameters in his application program, although he must ultimately consider their effect on main storage, job-control cards, etc.

In Example 4, a slight variant of Example 3, the I/O buffer areas are moved into the main program rather than being assembled with the DTFs. In Example 5, the label processing and exit functions are also moved into the main program. Examples 4 and 5 show how buffers and IOCS facilities can be moved between main program and separately assembled modules. If user label processing is standard throughout an installation, label exits should be assembled together with the DTFs. If each application requires special label processing, label exits should be assembled into the main program.

Appendix D: Reading, Writing, and Checking with Nonstandard Labels

```
                                        EXTERNAL SYMBOL DICTIONARY                                PAGE    1
 SYMBOL   TYPE ID   ADDR   LENGTH LD ID

          PC   01 003000 000048C
IJCFZIZ0  ER   02
IJFFZZZZ  ER   03
IJFFBZZZ  ER   04
IJDFZZZZ  ER   05
IJ2L0006  SD   06 003490 0000C64
```

```
       TEST  CREATING AND PROCESSING NON-STANDARD LABELS                                          PAGE    1

 LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  DOS CL3-2 07/07/69

                                     2          PRINT ON,NOGEN,NODATA                                    NSTD0004
003000                               3          START 12288                                              NSTD0005
                                     4 *                                                                 NSTD0006
                                     5 READER   DTFCD DEVICE=2540,DEVADDR=SYSIPT,BLKSIZE=80,TYPEFLE=INPUT,     *NSTD0007
                                                EOFADDR=ENDCARD,IOAREA1=IOAREA                           NSTD0008
                                    26 *                                                                 NSTD0009
                                    27 TAPEOUT DTFMT DEVADDR=SYS004,IOAREA1=IOAREA,BLKSIZE=80,TYPEFLE=OUTPUT,*NSTD0010
                                                LABADDR=LABELOUT,READ=FORWARD,FILABL=NSTD                NSTD0011
                                    58 *                                                                 NSTD0012
                                    59 TAPEIN  DTFMT DEVADDR=SYS004,IOAREA1=IOAREA,BLKSIZE=80,TYPEFLE=INPUT, *NSTD0013
                                                EOFADDR=ENDTAPE,READ=FORWARD,FILABL=NSTD,REWIND=NORWD,   *NSTD0014
                                                LABADDR=LABELIN                                          NSTD0015
                                    93 *                                                                 NSTD0016
                                    94 TAPEIN2 DTFMT DEVADDR=SYS004,IOAREA1=IOAREA,BLKSIZE=80,TYPEFLE=INPUT, *NSTD0017
                                                EOFADDR=ENDTAPE2,READ=BACK,FILABL=NSTD                   NSTD0018
                                   129 *                                                                 NSTD0019
                                   130 PRINT  DTFPR DEVICE=1403,DEVADDR=SYSLST,IOAREA1=IOAREA,BLKSIZE=80    NSTD0020
                                   151 *                                                                 NSTD0021
                                   152 CONSOLE DTFCN BLKSIZE=80,DEVADDR=SYSLOG,IOAREA1=CAREA,RECFORM=FIXUNB, *NSTD0022
                                                WORKA=YES                                                NSTD0023
                                   221 *                                                                 NSTD0024
                                   222 *                                                                 NSTD0025
0031AC 0520                        223 START  BALR  2,0                    SET UP A BASE REGISTER       NSTD0026
0031AE                             224        USING *,2                                                  NSTD0027
                                   225 * ** ROUTINE TO WRITE TAPE                                        NSTD0028
                                   226        OPEN  TAPEOUT                TO WRITE NSTD RECORDS        NSTD0029
                                   234 GETCARD GET   READER                 READ A CARD FROM CARD READER NSTD0030
                                   239        PUT   TAPEOUT                WRITE CARD IMAGE ON TAPE     NSTD0031
0031D6 47F0 2010          0318E    244        B     GETCARD                BRANCH AND GET ANOTHER CARD  NSTD0032
                                   245 ENDCARD CLOSE TAPEOUT                TO WRITE NSTD TRAILER LABEL  NSTD0033
                                   253 * ** ROUTINE TO READ TAPE FORWARD                                 NSTD0034
                                   254        OPEN  PRINT,TAPEIN           TO PROCESS NSTD LABEL        NSTD0035
                                   263 GETTAPE GET   TAPEIN                 GET A CARD IMAGE FROM TAPE   NSTD0036
                                   268        PUT   PRINT                  PRINT CARD IMAGE ON PRINTER  NSTD0037
003216 47F0 2050          031FE    273        B     GETTAPE        BRANCH AND GET ANOTHER TAPE RECORD NSTD0038
                                   274 ENDTAPE CLOSE TAPEIN                 PROCESS NSTD LABELS          NSTD0039
                                   282 * ** ROUTINE TO READ TAPE BACKWARDS                               NSTD0040
                                   283        OPEN  TAPEIN2                BYPASS NSTD LABELS           NSTD0041
                                   291 GETTAPE2 GET  TAPEIN2                READ A TAPE RECORD           NSTD0042
                                   296        PUT   PRINT                  PRINT RECORD                 NSTD0043
003252 47F0 208C          0323A    301        B     GETTAPE2       BRANCH AND GET ANOTHER TAPE RECORD NSTD0044
                                   302 ENDTAPE2 CLOSE PRINT,TAPEIN2         BYPASS NSTD RECORDS          NSTD0045
                                   311        CNTRL TAPEIN2,REW            REWIND TAPE TO LOAD POINT    NSTD0046
                                   317        EOJ                          NORMAL END OF JOB            NSTD0047
                                   320 * ** LABEL CREATION ROUTINE                                       NSTD0048
00327A 4900 22A6          03454    321 LABELOUT CH   0,ALPHA0              OPEN OR CLOSE                NSTD0049
00327E 4770 20F0          0329E    322        BNE   TRAILOUT               BRANCH IF CLOSE              NSTD0050
003282 D227 221C 21CC 033CA 0337A  323        MVC   IOAREA(40),HEADER      MOVE HEADER TO I/O AREA      NSTD0051
                                   324 RITELAB EXCP  OUTCCB                WRITE LABEL                  NSTD0052
                                   328        WAIT  OUTCCB                 WAIT FOR COMPLETION          NSTD0053
                                   334        LBRET 2                      RETURN CONTROL TO IOCS       NSTD0054
```

Figure 71. Reading, Writing, and Checking with Nonstandard Labels (Part 1 of 2)

```
        TEST  CREATING AND PROCESSING NON-STANDARD LABELS                                                    PAGE    2

  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                                                DOS CL3-2 07/07/69

00329E D227 221C 21F4 033CA 033A2   337 TRAILOUT MVC   IOAREA(40),TRAILER      MOVE TRAILER LABEL TO I/O AREA     NSTD0055
0032A4 47F0 20DA              03288   338          B     RITELAB               BRANCH TO WRITE THE LABEL          NSTD0056
                                      339 * ** LABEL PROCESSING ROUTINE                                       NSTD0057
0032A8 4900 22A6              03454   340 LABELIN  CH    0,ALPHAO              OPEN OR CLOSE                      NSTD0058
0032AC 4780 212C              032DA   341          BE    HEADIN                OPEN TIME                          NSTD0059
                                      342 TRAILIN  EXCP  INCCB                READ A TRAILER LABEL               NSTD0060
                                      346          WAIT  INCCB                WAIT FOR I/O COMPLETION            NSTD0061
0032C4 9101 2270       0341E          352          TM    INCCB+4,X'01'         TEST FOR A TAPE MARK               NSTD0062
0032C8 4710 2164              03312   353          BO    EXITEOF               BRANCH IF YES                      NSTD0063
0032CC D527 221C 21F4 033CA 033A2   354          CLC   IOAREA(40),TRAILER    COMPARE TRAILER LABEL              NSTD0064
0032D2 4780 2102              032B0   355          BE    TRAILIN               BRANCH TO GET ANOTHER RECORD       NSTD0065
0032D6 47F0 2152              03300   356          B     ERRLAB          BRANCH IF LABELS DO NOT COMPARE          NSTD0066
                                      357 HEADIN   EXCP  INCCB                READ A HEADER LABEL                NSTD0067
                                      361          WAIT  INCCB                WAIT FOR COMPLETION                NSTD0068
0032EE 9101 2270       0341E          367          TM    INCCB+4,X'01'         TEST FOR A TAPE MARK               NSTD0069
0032F2 4710 2168              03316   368          BO    EXIT                  BRANCH IF YES                      NSTD0070
0032F6 D527 221C 21CC 033CA 0337A   369          CLC   IOAREA(40),HEADER     DOES HEADER LABEL COMPARE          NSTD0071
0032FC 4780 212C              032DA   370          BE    HEADIN                IF YES, BRANCH AND READ TAPE       NSTD0072
                                      371 ERRLAB   PUT   CONSOLE,LABELERR       PUT LABEL ERROR MESSAGE            NSTD0073
                                      377          EOJ                         TERMINATE JOB                      NSTD0074
003312 5800 22A2              03450   380 EXITEOF  L     0,EOFIND              INDICATE EOF TO IOCS               NSTD0075
                                      381 EXIT     LBRET 2                     RETURN CONTROL TO IOCS             NSTD0076
                                      384 * CONSTANTS                                                         NSTD0077
003318 4040404040404040               385 CAREA    DC    CL50' '               CONSOLE I/O AREA                   NSTD0078
00334A E4E2C5D940D3C1C2               386 LABELERR DC    C'USER LABELS DO NOT COMPARE. ABNORMAL END OF JOB.  NSTD0079
00337A E4E2C5D940C8C5C1               387 HEADER   DC    CL40'USER HEADER LABEL'                             NSTD0080
0033A2 E4E2C5D940E3D9C1               388 TRAILER  DC    CL40'USER TRAILER LABEL'                            NSTD0081
0033CA 4040404040404040               389 IOAREA   DC    CL80' '               INPUT/OUTPUT AREA                  NSTD0082
                                      390 INCCB    CCB   SYS004,INCCW          READ TAPE CCB                      NSTD0083
                                      401 OUTCCB   CCB   SYS004,OUTCCW         WRITE TAPE CCB                     NSTD0084

00343A 000000000000
003440 020033CA0000CC28               412 INCCW    CCW   X'02',IOAREA,X'00',40 READ TAPE CCW                      NSTD0085
003448 010033CA0000C028               413 OUTCCW   CCW   X'01',IOAREA,X'00',40 WRITE TAPE CCW                     NSTD0086
003450 0000C5C6                       414 EOFIND   DC    X'0000C5C6'                                         NSTD0088
003454 00D6                           415 ALPHAO   DC    X'00D6'                                             NSTD0089
0031AC                                416          END   START                                               NSTD0090
003458 5B5BC2D6D7C5D540               417                =C'$$BOPEN '
003460 5B5BC2C3D3D6E2C5               418                =C'$$BCLOSE'
003468 00003000                       419                =A(READER)
00346C 00003038                       420                =A(TAPEOUT)
003470 00003090                       421                =A(TAPEIN)
003474 00003150                       422                =A(PRINT)
003478 000030F0                       423                =A(TAPEIN2)
00347C 0000342A                       424                =A(OUTCCB)
003480 0000341A                       425                =A(INCCB)
003484 00003180                       426                =A(CONSOLE)
003488 0000334A                       427                =A(LABELERR)
```

Notes: 1. IOCS wrote the first tapemark because the TAPEMARK=NO parameter was omitted.
2. IOCS always writes the tapemark following the data.
3. IOCS wrote the two tapemarks after the user trailer label.

Notes: 1. IOCS reads the first tapemark or bypasses it if user labels are not checked.
2. Upon encountering the second tapemark IOCS branches to the user label address.
3. After the user reads the third tapemark he should issue a LBRET 1 and IOCS will branch to the end-of-file address.

Figure 71. Reading, Writing, and Checking with Nonstandard labels (Part 2 of 2)

Appendix E: MICR Document Buffer Format

Buffer Status Indicators		
Byte	Bit	Comment
0,	0	The document is ready for processing (the user need never test this bit).
	1	Unrecoverable stacker select error, but all document data is present. The user may continue to issue GETs and READs.
	2	Unrecoverable I/O error. An operator I/O error message is issued. The file is inoperative and must be closed.
	3	Unit Exception. User requested disengage and all follow-up documents are processed. The LITE macro may now be issued, and the next GET or READ engages the device for continued reading.
	4	Intervention required or disengage failure. This buffer contains no data. The next GET or READ continues normal processing. This indicator allows the user program to give the operator information necessary to select pockets for documents not properly selected and to determine unread documents.
	5	The program issued a READ, no document is ready for processing, byte 0, bits 0-2 are OFF, cr the file is closed (byte 0, bit 6 is ON). The CHECK macro interrogates this bit. Note: The user must test bits 1-4 and take appropriate action. Any data from a buffer should not be processed if bits 2, 3, or 4 are ON.
	6	The program has issued a GET or READ and the file is closed. Bit 5 is also ON.
	7	Reserved with zero.

Figure 72. MICR Document Buffer Format (Part 1 of 4)

Buffer Status Indicator (Continued)		
Byte	Bit	Comment
1,	0	The user stacker selection routine turns this bit ON to indicate that batch numbering update (1419 only) is to be performed in conjunction with the stacker selection for this document. The document is imprinted with the updated batch number unless a late stacker selection occurs (byte 3, bit 2).
	1-7	Reserved with zero. Note: If bits 6 or 7 (byte 2) are ON, bit 0 is ignored by the external interrupt routine. With the 1419 (dual address) only, batch numbering update cannot be performed with the stacker selection of auto-selected documents.
2*,	0	For 1419 or 1275 (dual address) only. An auto-select condition occurred after the termination of a READ command but before a stacker-select command. The document is auto-selected into the reject pocket.
	1-3	Reserved with zero.
	4	Data check occurred while reading. Byte 3 should be interrogated by the user to determine errant fields.
	5	Overrun occurred while reading. Byte 3 should be interrogated to determine the error fields. Overruns cause short length data fields. When the 1419 or 1275 is enabled for fixed-length data fields, bit 4 is set.
	6&7	The specific meanings of bits 6 and 7 depend on the device type, the model, and the Engineering Change level of the MICR reader, but if either bit is ON, the document(s) concerned is auto-selected into the reject pocket. 1. 1412 or 1270: Bit 6 ON indicates that a late read condition occurred. Bit 7 ON indicates that a document spacing error occurred. (Unique to the 1412 or 1270, both the current document and the previous document are auto-selected into the reject pocket when this bit is ON. This previous document reject cannot be detected by IOCS, and byte 5 of its document buffer does not reflect that the reject pocket was selected.)
*Byte 2 (bits 4, 5, 6, and 7) and byte 3 contain MICR sense information. **Only for the 1259 Model 34 or 1419 Model 32. Bits 0 and 1 are not used for other models.		

Figure 72. MICR Document Buffer Format (Part 2 of 4)

Buffer Status Indicators (Continued)		
Byte	Bit	Comment
		2. 1275 and 1419 (single address) without Engineering Change #125358: Bit 6 indicates either a late read condition or a document spacing error occurred. Bit 7 indicates a document spacing error for the current document. 3. 1255, 1259, 1275, and 1419 (single or dual address) with Engineering Change #125358: Bit 6 indicates that an auto-select condition occurred while reading a document. The bit is set at the termination of the READ command before entry into the stacker select routine. Bit 7 is always zero.
3*,	0	Field 6 valid.**
	1	Field 7 valid.**
	2	A late stacker selection (unit check late stacker select on the stacker select command). The document is auto-selected into the reject pocket.
	3	Amount field valid (or field 1 valid).**
	4	Process control field valid (or field 2 valid).**
	5	Account number field valid (or field 3 valid).**
	6	Transit field valid (or field 4 valid).**
	7	Serial number field valid (or field 5 valid).** Note: 1. For the 1412 or 1270, bits 3-7 are set to zero when the fields are read without error. 2. For the 1255, 1259, 1275, and 1419, bits 3-7 set ON when each respective field, including bracket symbols, is read without error. This applies to bits 0, 1, and 3-7 on the 1259 and 1419 Model 32. 3. For the 1255, 1259, 1275, 1412, and 1419, unread fields contain zero bits. Errors are indicated when an overrun or data check condition occurs while reading the data field.

*Byte 2 (bits 4, 5, 6, and 7) and byte 3 contain MICR sense information.
**Only for the 1259 Model 34 or 1419 Model 32. Bits 0 and 1 are not used for other models.

Figure 72. MICR Document Buffer Format (Part 3 of 4)

Buffer Status Indicators (Continued)		
Byte	Bit	Comment
4		User inserted pocket code determination by the user stacker select routine. Whenever byte 0, bits 2, 3, or 4 are ON, this byte is X'00' because no document was read and the user stacker selection routine was not entered. Whenever auto-selection occurs, this user value is ignored. A no-op (X'03') is issued to the device, and a reject pocket value (X'CF') is placed in byte 5. The pocket codes are: (byte 2, bit 6 or 7 ON). Pocket A* - X'AF' Pocket B** - X'BF' Pocket 0 - X'0F' Pocket 1 - X'1F' Pocket 2 - X'2F' Pocket 3 - X'3F' Pocket 4 - X'4F' Pocket 5 - X'5F' Pocket 6 - X'6F' Pocket 7 - X'7F' Pocket 8 - X'8F' Pocket 9 - X'9F' Reject Pocket - X'CF' Except 1270 Models 1 and 3
5		The actual pocket selected for the document. The contents are normally the same as that in byte 4. Note: 1. X'CF' is inserted whenever auto-selection occurs (byte 2, bit 6; byte 2, bit 7; byte 2, bit 0; or byte 3 bit 2). These conditions may result from late READ commands, errant document spacing, or late stacker selection. a. Start I/O for stacker selection is unsuccessful (byte 0, bit 1). b. An I/O error occurs (for example, invalid pocket code) on the 1419 (dual address) secondary control unit when selecting this document.

Additional User Work Area
This additional buffer area can be used as a workarea and/or output area. Its size is determined by the DTFMR ADDAREA=n entry. The only size restriction is that this area, plus the 6-byte status indicators and data portion must not exceed 256 bytes. Note: This area may be omitted.

Document Data Area
The document data area immediately follows the user workarea. The data is right-adjusted in the document data area. The length of this data area is determined by the DTFMR RECSIZE={n / 80} entry.

*1275, 1412, 1419, and 1270 Models 2 and 4 only.
**1275, 1412, and 1419 only.

Figure 72. MICR Document Buffer Format (Part 4 of 4)

Appendix F: Spanned Records

Spanned records are format V records, each of which specifies its own length. Spanned record processing is an extension of variable-length record processing. In this technique, the user need not be concerned with the restrictions the system imposes on the length of physical records. Thus, he can maximize his secondary storage efficiency, while organizing his data files with logical record lengths most suited to his needs. The sequential DASD access method allows a logical record, either blocked or unblocked, to span multiple physical records. This implies that:

1. The user only concerns himself with logical records. The IOCS segments and blocks his logical records for him, while it makes the most efficient use of the track capacities on his DASD devices.

2. The user is allowed greater flexibility in transferring logical records from one type of DASD device to another, and between tape and sequential DASD devices, when he uses the sequential DASD access method.

Figure 73 shows spanned records. The first four bytes of every spanned record, whether blocked or unblocked, constitute the block descriptor word, which describes the information portion of the block that immediately follows it. The first two bytes contain the block length (LL), which is supplied by data management when the data set is written. The last two bytes (RR) are reserved and set to binary zeros. The user is required to reserve (for use by IOCS) the four bytes occupied by the block descriptor word at the beginning of his input and output areas.

The length of each logical record ($\ell\ell$), including two bytes for the length field and two bytes for system use (rr), must be supplied by the problem programmer when the record is written.

Because the length of a logical record may exceed the size of a single physical record on the associated device, IOCS may write a spanned record in sections called segments. Figure 74 shows segmented spanned records.

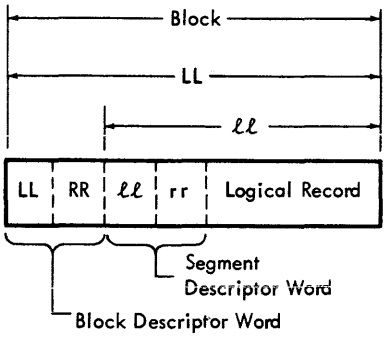
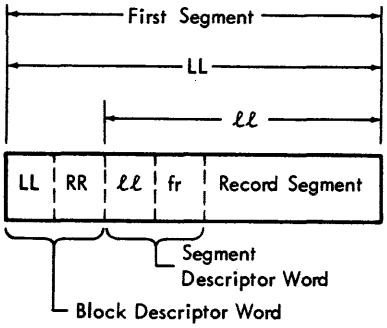


Figure 73. Spanned Records (Unblocked)



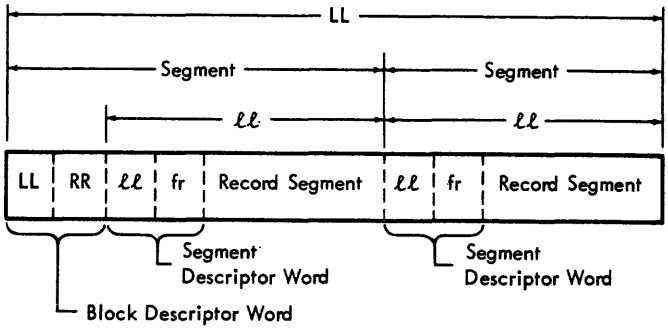


Figure 74. Segmented Spanned Records (Blocked)

When the logical record is written in segments, each segment includes a segment descriptor word. The segment descriptor word is an additional four-byte field that describes the data portion of the segment that immediately follows it.

The segment length, including the four bytes occupied by the segment descriptor word itself, is contained in bits 1-15 of the first two bytes ($\ell\ell$). The value must lie in the range $4 \leq \ell\ell \leq 32,763$.

Bit 0 describes the segment type. If the bit is OFF, it indicates that the segment is a normal one. If the bit is ON, it indicates a null segment containing the eight descriptor bytes only.

The last two bytes of the segment descriptor word are reserved and set to binary zeros, with the exception of bits 6 and 7, which contain a value (f). This value specifies the relative position of the segment with respect to other segments, if any (that is, whether it is a single segment; or first, last, or intermediate segment of a multisegment logical record). When a spanned record is read, the segment lengths specified in each segment descriptor word are added together to provide the problem program with the length (*ll*) of the logical record.

The first segment of a spanned record may begin at any point in the physical record on the associated device.

Appendix G: Self-Relocating Programs

A system supporting multiprogramming has the capability of executing self-relocating programs. A self-relocating program is one that can be executed at any location in main storage. Writing a self-relocating program is an efficient coding technique because self-relocating programs are linkage edited only once for execution in any partition. When linkage editing, use OPTION CATAL and a PHASE card, such as:

```
PHASE  Phasename,+0
```

This causes the linkage editor to assume that the program is loaded at core location zero, and to compute all absolute addresses from the beginning of the phase. The job control EXEC function recognizes a zero phase address and adjusts the origin address to compensate for the current partition boundary save area and label area (if any). Control is then given to the updated entry address of the phase. Programs that are written using self relocating techniques can be cataloged as either self relocating or non-self-relocating phases.

RULES FOR WRITING SELF-RELOCATING PROGRAMS

In general, if a problem program is written to be self-relocating, these rules must be followed:

1. The supervisor must support multiprogramming (that is, MPS=YES or BJF must be specified as a parameter in the SUPVR macro at system generation time).

2. The PHASE card must specify an origin of +0.

3. The program must relocate all address constants used in the program. Whenever possible, use the LA instruction to load an address in a register instead of using an A-type address constant. For example,

Instead of Using:

```
         USING  *,12
         BALR   12,0
         LA     12,0(12)
         BCTR   12,0
         BCTR   12,0
         LA     1,EOF
         ST     1,AEOF
         .
         .
         L      10,AEOF
         .
         .
         .
EOF      EOJ
         .
         .
AEOF     DC     A(EOF)
```

Use:

```
         USING  *,12
         BALR   12,0
         LA     12,0(12)
         BCTR   12,0
         BCTR   12,0
         .
         .
         LA     10,EOF
         .
         .
EOF      EOJ
```

4. If logical IOCS is used, the program must use the OPENR and CLOSER macros to open and close files. The console typewriter should also be opened by using the OPENR macro.

5. If physical IOCS is used, the program must relocate all CCW address fields.

6. Register notation must be used when issuing an imperative macro (I/O, I/O control, and supervisor communication). Register notation utilizes less main storage and permits faster execution. An example of coding the GET macro with a work area in self-relocating format follows:

```
RCARDIN EQU    4
RPRTOUT EQU    5
RWORK   EQU    6
        LA     RCARDIN,CARDIN
        LA     RPRTOUT,PRTOUT
        LA     RWORK,WORK
        OPENR  (RCARDIN),(RPRTOUT)
        .
        .
        .
        GET    (RCARDIN),(RWORK)
```

Note: Since the DTF name can be a maximum of seven characters, an R can be prefixed to this name to identify the file. Thus, RCARDIN in this example can immediately be associated with the corresponding DTF name CARDIN.

7. Use // LBLTYP before // EXEC card.

Note: Items 8, 9, and 10 apply to multimodule programs.

8. The relocation factor should be calculated and stored in a register for future use. For register economy, the base register can hold the relocation factor.

   For example:

```
USING  *,12
BALR   12,0
LA     12,0(12)
BCTR   12,0
BCTR   12,0
```

   Register 12 now contains the relocation factor and the program base.

9. When branching to an external address, use one of the following techniques:

```
L      15,=V(EXTERNAL)
BAL    14,0(12,15)
```

   or

```
L      15,=V(EXTERNAL)
AR     15,12
BALR   14,15
```

   where register 12 is the base register containing the relocation factor.

10. The calling program is responsible for relocating all address constants in the calling list(s). See Figure 75 for an example of the calling program relocating the address constants in a calling list.

```
// JOB A
// OPTION LINK
// EXEC ASSEMBLY
CSECT1   START 0
         USING *,12      Use load point value as the base to
         BALR  12,0      find the load point value.
         LA    12,0(12)
         BCTR  12,0
         BCTR  12,0
         .
         .
         LA    1,A
         LA    2,B
         LA    3,C       Modify the CALL address constant list.
         LA    4,D
         STM   1,4,LIST
         OI    LIST+12,X'80'  Restore end of list bit in last adcon.
         LA    13,SAVEAREA
         L     15,=V(EXTERNAL)
         AR    15,12     Adjust CALL address by relocation
                         factor.
         CALL  (15),(A,B,C,D)
LIST     EQU   *-16      For address constants (4 bytes each).
         EOJ
SAVEAREA DC    9D'0'
         END
/* *
// EXEC ASSEMBLY
CSECT2   START 0
         ENTRY EXTERNAL
EXTERNAL SAVE  (14,12)
         USING *,12
         BALR  12,0      Establish new base.
         .
         .
         RETURN(14,12)
         END
/* *
// EXEC LNKEDT
```

Figure 75. Relocating Address Constants in a Calling List

ADVANTAGES OF SELF-RELOCATING PROGRAMS

Self-relocating programs have the ability to run in any problem program partition without needing linkage editing again. The program can also be loaded anywhere within a partition. The restriction of specific partition allocations need not be followed with a self-relocating program because it relocates itself.

Also, once a program is written in self-relocating format, the program can be run under the Operating System (OS) with a minimum of changes.

PROGRAMMING TECHNIQUES

A program is self-relocating if it is capable of proper execution, regardless of where it is loaded. DTFDI should be used to resolve the problem of device differences between partitions. A self-relocating program must also adjust all its own absolute addresses to point to the proper address. This must be done after the program is loaded, and before the absolute addresses are used.

Within these self-relocating programs, some macros generate self-relocating code. For example, the MPS utility macros are self-relocating (that is, they modify all their own address constants to their proper values before using them). OPENR and CLOSER macros are used in self-relocating programs. OPENR and CLOSER can be used in place of OPEN and CLOSE, and adjust all the address constants in the DTFs opened and closed. OPENR and CLOSER can be used in any program because the OPENR macro computes the amount of relocation. If relocation is 0, the standard open is executed. In addition, all the module generation (xxMOD) macros are self-relocating.

The addresses of all address constants containing relocatable values are listed in the relocation dictionary in the assembly listing. This dictionary includes both those address constants that are modified by self-relocating macros, and those that are not. The address constants not modified by self-relocating macros must be modified by some other technique. After the program has been linkage edited with a phase origin of +0, the contents of each address constant is the displacement from the beginning of the phase to the address pointed to by that address constant.

The following techniques place relocated absolute addresses in address constants. These techniques are required only when the LA instruction cannot be used.

Technique 1

Named A-type address constants:

```
      .
      L         4,ADCON
      AR        4,12
      ST        4,ADCON
      .
      .
ADCON DC        A(ADCONAME)
```

Technique 2

A-type address constants in the literal pool:

```
      .
      LA        3,=A(ADCONAME)
      LA        4,ADCONAME
      ST        4,0(3)
      .
      .
      LTORG
                =A(ADCONAME)
```

Technique 3

A-type address constants with a specified length of three bytes, and a nonzero value in the adjacent left byte (as in CCWs):

1. If the CCW list dynamically changes during program execution:

```
        .
        IC    3,TAPECCW
        LA    4,IOAREA
        ST    4,TAPECCW
        STC   3,TAPECCW
        .
TAPECCW CCW   1,IOAREA,X'20',100
        .
        .
IOAREA  DS    CL100
```

or

```
         .
         USING *,12
         BALR  12,0
         LA    12,0(12)
         BCTR  12,0
         BCTR  12,0 Register 12 contains
                    relocation factor.
         .
         L     11,TAPECCW
         ALR   11,12
         ST    11,TAPECCW
         .
TAPECCW  CCW   1,IOAREA,X'20',100
         .
IOAREA   DS    GL100
```

The load point of the phase is not synonymous with the relocation factor as developed in register 3 (Technique 4). If the load point of the phase is taken from register 0 (or calculated by a BALR and subtracting 2) immediately after the phase is loaded, correct results are obtained if the phase is linkage edited with an origin of +0. If a phase is linkage edited with an origin of * or S, incorrect results will follow. This is because the linkage editor and the problem program have both added the load point to all address constants. Figure 76 shows an example of a self relocating program.

2. If the CCW list is static during program execution:

```
         .
         LA    4,IOAREA
         ST    4,TAPECCW
         MVI   TAPECCW,1
         .
         .
TAPECCW  CCW   1,IOAREA,X'20',100
         .
         .
         DS    CL100
```

Technique 4

Named V-type or A-type address constants:

```
         .
         LA    3,ADCONAST   Determine
         S     3,ADCONAST   Relocation
                            factor
         .
         .
         L     4,ADCON
         AR    4,3  Add relocation factor
         ST    4,ADCON
         .
         .
ADCONAST DC    A(*)
ADCON    DC    V(NAME)
```

```
    SOURCE STATEMENTS
             REPRO
             PHASE EXAMPLE,+0                    +0 ORIGIN IMPLIES SELF-RELOCATION
             PRINT NOGEN
PROGRAM      START 0
             BALR  12,0
             USING *,12
*       ROUTINE TO RELOCATE ADDRESS CONSTANTS
             LA    1,PRINTCCW                    RELOCATE CCW ADDRESS
             ST    1,PRINTCCB+8                    IN CCB FOR PRINTER
             LA    1,TAPECCW                     RELOCATE CCW ADDRESS
             ST    1,TAPECCB+8                     IN CCB FOR INPUT TAPE
             IC    2,PRINTCCW                    SAVE PRINT CCW OP CODE
             LA    1,OUTAREA                     RELOCATE OUTPUT AREA ADDRESS
             ST    1,PRINTCCW                      IN PRINTER CCW
             STC   2,PRINTCCW                    RESTORE PRINT CCW OP CODE
             LA    1,INAREA                      RELOCATE INPUT AREA ADDRESS
             ST    1,TAPECCW                       IN TAPE CCW
             MVI   TAPECCW,READ                  SET TAPE CCW OP CODE TO READ
*       MAIN ROUTINE...READ TAPE AND PRINT RECORDS
READTAPE     LA    1,TAPECCB                     GET CCB ADDRESS
             EXCP  (1)                           READ ONE RECORD FROM TAPE
             WAIT  (1)                           WAIT FOR I/O COMPLETION
             LA    10,EOFTAPE                    GET ADDRESS OF TAPE EOF ROUTINE
             BAL   14,CHECK                      GO TO UNIT EXCEPTION SUBROUTINE
             MVC   OUTAREA(10),INAREA            EDIT RECORD
             MVC   OUTAREA+15(70),INAREA+10      IN
             MVC   OUTAREA+90(20),INAREA+80      OUTPUT AREA
             LA    1,PRINTCCB                    GET CCB ADDRESS
             EXCP  (1)                           PRINT EDITED RECORD
             WAIT  (1)                           WAIT FOR I/O COMPLETION
             LA    10,CHA12                      GET ADDRESS OF CHAN 12 ROUTINE
             BAL   14,CHECK                      GO TO UNIT EXCEPTION SUBROUTINE
             B     READTAPE
CHECK        TM    4(1),1                        CHECK FOR UNIT EXEC. IN CCB
             BCR   1,10                           YES-GO TO PROPER ROUTINE
             BR    14                             NO-RETURN TO MAINLINE
CHA12        MVI   PRINTCCW,SKIPTO1              SET SEEK TO CHAN 1 OP CODE
             EXCP  (1)                           SEEK TO CHAN 1 IMMEDIATELY
             WAIT  (1)                           WAIT FOR I/O COMPLETION
             MVI   PRINTCCW,PRINT                SET PRINTER OP CODE TO WRITE
             BR    14                            RETURN TO MAINLINE
EOFTAPE      EOJ                                 END OF JOB
             CNOP  0,4                           ALIGN CCB'S TO FULL WORD
PRINTCCB     CCB   SYS004,PRINTCCW,X'0400'
TAPECCB      CCB   SYS001,TAPECCW

PRINTCCW     CCW   PRINT,OUTAREA,SLI,L'OUTAREA
TAPECCW      CCW   READ,INAREA,SLI,L'INAREA
OUTAREA      DC    CL110' '
INAREA       DC    CL100' '
SLI          EQU   X'20'
READ         EQU   2
PRINT        EQU   9
SKIPTO1      EQU   X'8B'
             END   PROGRAM
```

Figure 76. Self-Relocating Sample Program

Appendix H: American National Standard Code for Information Interchange (ASCII)

In addition to the EBCDIC mode, the IBM Disk Operating System accepts magnetic tape files written in ASCII (American National Standard Code for Information Interchange), a 128-character, 7-bit code. The high-order bit in this 8-bit environment is zero. ASCII is based on the specifications of the American National Standards Institute, Inc.

DOS processes ASCII data files in EBCDIC. At system generation time, if ASCII=YES is specified in the SUPVR macro, two translate tables are included in the supervisor. Using these tables, logical IOCS translates from ASCII to EBCDIC as soon as the data is read into the I/O area. For ASCII output, logical IOCS translates data from EBCDIC to ASCII just before writing the record. The address of the ASCII to EBCDIC translate table is in bytes 44-47 of the extension of the communication region for each partition. The address of the EBCDIC to ASCII table is 256 bytes higher than that of the first table. The address of the extension of the communication region is found in bytes 136-139 of the communication region.

Figure 77 shows the relative bit positions of the ASCII character set. An ASCII character is described by its column/row position in the table. The columns across the top of Figure 77 list the three high-order bits. The rows along the left side of Figure 77 are the four low-order bits. Because the letter P in ASCII is under column 5 and in row 0, it is described in ASCII notation as 5/0. ASCII 5/0 and EBCDIC X'50' represent the same binary configuration (B'0101 0000'). However, P graphically represents this configuration in ASCII and & in EBCDIC. ASCII notation is always expressed in decimal. For example, the ASCII Z is expressed 5/10 (not 5/A).

For those EBCDIC characters that have no direct equivalent in ASCII, the substitute character (SUB) is provided during translation. (See Figure 78 for ASCII to EBCDIC correspondence.)

> Note: If an EBCDIC file is translated into ASCII, and then the user translates back into EBCDIC, this substitute character may not receive the expected value.

b_7 b_6 b_5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits ③ b_4	b_3	b_2	b_1	Row ↓ Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!①	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^②	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

① The graphic ¦ (Logical OR) may also be used instead of ! (Exclamation Point).

② The graphic ¬ (Logical NOT) may also be used instead of ^ (Circumflex).

③ The 7 bit ASCII code expands to 8 bits when in storage by adding a high order 0 bit.

Example: Pound sign (#) is represented by

b_7	b_6	b_5	b_4	b_3	b_2	b_1
0	0	1	0	0	0	1	1

Control Character Representations

NUL	Null	DLE	Data Link Escape (CC)
SOH	Start of Heading (CC)	DC1	Device Control 1
STX	Start of Text (CC)	DC2	Device Control 2
ETX	End of Text (CC)	DC3	Device Control 3
EOT	End of Transmission (CC)	DC4	Device Control 4
ENQ	Enquiry (CC)	NAK	Negative Acknowledge (CC)
ACK	Acknowledge (CC)	SYN	Synchronous Idle (CC)
BEL	Bell	ETB	End of Transmission Block (CC)
BS	Backspace (FE)	CAN	Cancel
HT	Horizontal Tabulation (FE)	EM	End of Medium
LF	Line Feed (FE)	SUB	Substitute
VT	Vertical Tabulation (FE)	ESC	Escape
FF	Form Feed (FE)	FS	File Separator (IS)
CR	Carriage Return (FE)	GS	Group Separator (IS)
SO	Shift Out	RS	Record Separator (IS)
SI	Shift In	US	Unit Separator (IS)
		DEL	Delete

(CC) Communication Control
(FE) Format Effector
(IS) Information Separator

Special Graphic Characters

SP	Space	<	Less Than
!	Exclamation Point	=	Equals
¦	Logical OR	>	Greater Than
"	Quotation Marks	?	Question Mark
#	Number Sign	@	Commercial At
$	Dollar Sign	[	Opening Bracket
%	Percent	\	Reverse Slant
&	Ampersand	]	Closing Bracket
'	Apostrophe	^	Circumflex
(	Opening Parenthesis	¬	Logical NOT
)	Closing Parenthesis	_	Underline
*	Asterisk	`	Grave Accent
+	Plus	{	Opening Brace
,	Comma	¦	Vertical Line (This graphic is stylized to distinguish it from Logical OR)
-	Hyphen (Minus)		
.	Period (Decimal Point)		
/	Slant	}	Closing Brace
:	Colon	~	Tilde
;	Semicolon		

Figure 77. ASCII Character Set

ASCII					EBCDIC				
Character	Col	Row	Bit Position		Col (in Hex)	Row	Bit Position		Comments
NUL	0	0	0000	0000	0	0	0000	0000	
SOH	0	1	0000	0001	0	1	0000	0001	
STX	0	2	0000	0010	0	2	0000	0010	
ETX	0	3	0000	0011	0	3	0000	0011	
EOT	0	4	0000	0100	3	7	0011	0111	
ENQ	0	5	0000	0101	2	D	0010	1101	
ACK	0	6	0000	0110	2	E	0010	1110	
BEL	0	7	0000	0111	2	F	0010	1111	
BS	0	8	0000	1000	1	6	0001	0110	
HT	0	9	0000	1001	0	5	0000	0101	
LF	0	10	0000	1010	2	5	0010	0101	
VT	0	11	0000	1011	0	B	0000	1011	
FF	0	12	0000	1100	0	C	0000	1100	
CR	0	13	0000	1101	0	D	0000	1101	
SO	0	14	0000	1110	0	E	0000	1110	
SI	0	15	0000	1111	0	F	0000	1111	
DLE	1	0	0001	0000	1	0	0001	0000	
DC1	1	1	0001	0001	1	1	0001	0001	
DC2	1	2	0001	0010	1	2	0001	0010	
DC3	1	3	0001	0011	1	3	0001	0011	
DC4	1	4	0001	0100	3	C	0011	1100	
NAK	1	5	0001	0101	3	D	0011	1101	
SYN	1	6	0001	0110	3	2	0011	0010	
ETB	1	7	0001	0111	2	6	0010	0110	
CAN	1	8	0001	1000	1	8	0001	1000	
EM	1	9	0001	1001	1	9	0001	1001	
SUB	1	10	0001	1010	3	F	0011	1111	
ESC	1	11	0001	1011	2	7	0010	0111	
FS	1	12	0001	1100	1	C	0001	1100	
GS	1	13	0001	1101	1	D	0001	1101	
RS	1	14	0001	1110	1	E	0001	1110	
US	1	15	0001	1111	1	F	0001	1111	
SP	2	0	0010	0000	4	0	0100	0000	
!①	2	1	0010	0001	4	F	0100	1111	Logical OR
"	2	2	0010	0010	7	F	0111	1111	
#	2	3	0010	0011	7	B	0111	1011	
$	2	4	0010	0100	5	B	0101	1011	
%	2	5	0010	0101	6	C	0110	1100	
&	2	6	0010	0110	5	0	0101	0000	
'	2	7	0010	0111	7	D	0111	1101	
(	2	8	0010	1000	4	D	0100	1101	
)	2	9	0010	1001	5	D	0101	1101	
*	2	10	0010	1010	5	C	0101	1100	
+	2	11	0010	1011	4	E	0100	1110	
,	2	12	0010	1100	6	B	0110	1011	
-	2	13	0010	1101	6	0	0110	0000	Hyphen, Minus
.	2	14	0010	1110	4	B	0100	1011	
/	2	15	0010	1111	6	1	0110	0001	
0	3	0	0011	0000	F	0	1111	0000	
1	3	1	0011	0001	F	1	1111	0001	
2	3	2	0011	0010	F	2	1111	0010	
3	3	3	0011	0011	F	3	1111	0011	
4	3	4	0011	0100	F	4	1111	0100	
5	3	5	0011	0101	F	5	1111	0101	
6	3	6	0011	0110	F	6	1111	0110	
7	3	7	0011	0111	F	7	1111	0111	
8	3	8	0011	1000	F	8	1111	1000	
9	3	9	0011	1001	F	9	1111	1001	
:	3	10	0011	1010	7	A	0111	1010	
;	3	11	0011	1011	5	E	0101	1110	
<	3	12	0011	1100	4	C	0100	1100	
=	3	13	0011	1101	7	E	0111	1110	
>	3	14	0011	1110	6	E	0110	1110	
?	3	15	0011	1111	6	F	0110	1111	

Figure 78. ASCII to EBCDIC Correspondence (Part 1 of 2)

ASCII					EBCDIC				
Character	Col	Row	Bit Pattern		Col (in Hex)	Row (in Hex)	Bit Pattern		Comments
@	4	0	0100	0000	7	C	0111	1100	
A	4	1	0100	0001	C	1	1100	0001	
B	4	2	0100	0010	C	2	1100	0010	
C	4	3	0100	0011	C	3	1100	0011	
D	4	4	0100	0100	C	4	1100	0100	
E	4	5	0100	0101	C	5	1100	0101	
F	4	6	0100	0110	C	6	1100	0110	
G	4	7	0100	0111	C	7	1100	0111	
H	4	8	0100	1000	C	8	1100	1000	
I	4	9	0100	1001	C	9	1100	1001	
J	4	10	0100	1010	D	1	1101	0001	
K	4	11	0100	1011	D	2	1101	0010	
L	4	12	0100	1100	D	3	1101	0011	
M	4	13	0100	1101	D	4	1101	0100	
N	4	14	0100	1110	D	5	1101	0101	
O	4	15	0100	1111	D	6	1101	0110	
P	5	0	0101	0000	D	7	1101	0111	
Q	5	1	0101	0001	D	8	1101	1000	
R	5	2	0101	0010	D	9	1101	1001	
S	5	3	0101	0011	E	2	1110	0010	
T	5	4	0101	0100	E	3	1110	0011	
U	5	5	0101	0101	E	4	1110	0100	
V	5	6	0101	0110	E	5	1110	0101	
W	5	7	0101	0111	E	6	1110	0110	
X	5	8	0101	1000	E	7	1110	0111	
Y	5	9	0101	1001	E	8	1110	1000	
Z	5	10	0101	1010	E	9	1110	1001	
[	5	11	0101	1011	4	A	0100	1010	
\	5	12	0101	1100	E	0	1110	0000	Reverse Slant
]	5	13	0101	1101	5	A	0101	1010	
¬②	5	14	0101	1110	5	F	0101	1111	Logical NOT
_	5	15	0101	1111	6	D	0110	1101	Underscore
`	6	0	0110	0000	7	9	0111	1001	Grave Accent
a	6	1	0110	0001	8	1	1000	0001	
b	6	2	0110	0010	8	2	1000	0010	
c	6	3	0110	0011	8	3	1000	0011	
d	6	4	0110	0100	8	4	1000	0100	
e	6	5	0110	0101	8	5	1000	0101	
f	6	6	0110	0110	8	6	1000	0110	
g	6	7	0110	0111	8	7	1000	0111	
h	6	8	0110	1000	8	8	1000	1000	
i	6	9	0110	1001	8	9	1000	1001	
j	6	10	0110	1010	9	1	1001	0001	
k	6	11	0110	1011	9	2	1001	0010	
l	6	12	0110	1100	9	3	1001	0011	
m	6	13	0110	1101	9	4	1001	0100	
n	6	14	0110	1110	9	5	1001	0101	
o	6	15	0110	1111	9	6	1001	0110	
p	7	0	0111	0000	9	7	1001	0111	
q	7	1	0111	0001	9	8	1001	1000	
r	7	2	0111	0010	9	9	1001	1001	
s	7	3	0111	0011	A	2	1010	0010	
t	7	4	0111	0100	A	3	1010	0011	
u	7	5	0111	0101	A	4	1010	0100	
v	7	6	0111	0110	A	5	1010	0101	
w	7	7	0111	0111	A	6	1010	0110	
x	7	8	0111	1000	A	7	1010	0111	
y	7	9	0111	1001	A	8	1010	1000	
z	7	10	0111	1010	A	9	1010	1001	
{	7	11	0111	1011	C	0	1100	0000	
①|	7	12	0111	1100	6	A	0110	1010	Vertical Line
}	7	13	0111	1101	D	0	1101	0000	
~	7	14	0111	1110	A	1	1010	0001	Tilde
DEL	7	15	0111	1111	0	7	0000	0111	

① The graphic ! (Exclamation Point) can be used instead of | (Logical OR).

② The graphic ^ (Circumflex) can be used instead of ¬ (Logical NOT).

Figure 78. ASCII to EBCDIC Correspondence (Part 2 of 2)

Glossary

For a more complete list of data processing terms, refer to IBM Data Processing Techniques, A Data Processing Glossary, GC20-1699.

access method: Any of the data management techniques (sequential, indexed sequential, or direct) available to the user for transferring data between main storage and an input/output device.

ASCII (American National Standard Code for Information Interchange): A 128-character, 7-bit code. The high order bit in the System/360 8-bit environment is zero.

Basic Telecommunications Access Method (BTAM): A basic access method that permits a READ/WRITE communication with remote devices.

block:

1. To group records physically for the purpose of conserving storage space or increasing the efficiency of access or processing.

2. A physical record on tape or DASD.

block prefix: An optional, 0-99 byte field preceding an ASCII record. It contains user specified data or, for variable length (format D) records, the physical record length.

buffer:

1. A storage device in which data is assembled temporarily during data transfer. An example is the IBM 2821 Control Unit, a control and buffer storage unit for card readers, card punches, and printers.

2. During I/O operations, a portion of main storage into which data is read or from which data is written.

channel program: One or more Channel Command Words (CCWs) that control(s) a specific sequence of channel operations. Execution of the specific sequence is initiated by a single start I/O instruction.

checkpoint record: A record containing the status of the job and of the system at the time the checkpoint routine writes the record. This record provides the necessary information for restarting a job without returning to the beginning of the job.

Checkpoint/Restart: A means of restarting execution of a program at some point other than the beginning. When a checkpoint macro instruction is issued in a problem program, checkpoint records are created. These records contain the status of the program and the machine. When it is necessary to restart a program at a point other than the beginning, the restart procedure uses the checkpoint records to reinitialize the system.

checkpoint routine: A routine that records information for a checkpoint.

Command Control Block: A sixteen-byte field required for each channel program executed by physical IOCS. This field is used for communication between physical IOCS and the problem program.

communication region: An area of the supervisor set aside for interprogram and intraprogram communication. It contains information useful to both the supervisor and the problem program.

control program: A group of programs that provides functions such as the handling of input/output operations, error detection and recovery, program loading, and communication between the program and the operator. IPL, supervisor, and job control make up the control program in the Disk and Tape Operating Systems.

control section: The smallest separately relocatable unit of a program; that portion of text specified by the programmer to be an entity, all elements of which are to be loaded into contiguous main storage locations.

core storage: See main storage.

data conversion: The process of changing data from one form of representation to another.

data file: A collection of related data records organized in a specific manner. For example, a payroll file (one record for each employee, showing his rate of pay, deductions, etc) or an inventory file (one record for each inventory item, showing the cost, selling price, number in stock, etc).

data set security: A feature that provides protection for disk files. A data secured file cannot be accidentally accessed by a problem program.

device independence: The capability of a program to process the same type of data on different device types (punched card devices/printers, tape, or disk).

Disk Operating System: A disk resident system that provides operating system capabilities for 16K and larger IBM System/360 and System/370 systems.

DTF (define the file) macro instruction: A macro instruction that describes the characteristics of a logical input/output file, indicates the type of processing for the file, and specifies the main storage areas and routines to process the file. To do this, use the appropriate entries in the keyword operands associated with the DTF macro instruction.

DUMP: Displaying the contents of main storage.

extent: The physical locations on Input/Output devices occupied by or reserved for a particular file.

fetch:

1. To bring a program phase into main storage from the core image library for immediate execution.

2. The routine that retrieves requested phases and loads them into main storage.

3. The name of a macro instruction (FETCH) used to transfer control to the System Loader.

4. To transfer control to the system loader.

file: See data file.

fixed length record: A record having the same length as all other records with which it is logically or physically associated.

header label: A file label that precedes the data records on a unit of recording media.

I/O area: An area (portion) of main storage into which data is read or from which data is written. In Operating System publications, the term buffer is often used in place of I/O area. I/O means Input/Output.

IOCS (input/output control system): A group of macro instruction routines provided by IBM for handling the transfer of data between main storage and external storage devices.

load: To fetch, that is, to read a phase into main storage returning control to the calling phase.

load point: The beginning of the recording area on a reel of magnetic tape.

logic module: The logical IOCS routine that provides an interface between a processing program and physical IOCS.

main storage: All addressable storage from which instructions can be executed or from which data can be loaded directly into registers.

main task: The main program within a partition in a multiprogramming environment.

MPS: Multiprogramming system.

multifile volume: A unit of recording media, such as a tape reel or disk pack, that contains more than one data file.

multiprogramming system: A system that controls more than one program simultaneously by interleaving their execution.

multitask operation: Multiprogramming; called multitask operation to express not only concurrent execution of one or more programs in a partition, but also of a single reenterable program used by many tasks.

multivolume file: A data file that, due to its size, requires more than one unit of recording media (such as a tape reel or a disk pack) to contain the entire file.

nonstandard labels: Labels that do not conform to the System/360 standard label specifications. They can be any length, need not have a specified identification, and do not have a fixed format.

operating system: A collection of programs that enables a data processing system to supervise its own operations, automatically calling in programs, routines, languages, and data as needed for continuous throughput of a series of jobs.

phase: The smallest complete unit that can be referenced in the core image library. Each program overlay is a complete phase. If the program has no overlays, the program itself is a complete phase.

physical record: A record identified from the standpoint of the manner or form in which it is stored and retrieved; that is, one that is meaningful with respect to access. (Contrasted with Logical Records.)

private library: A relocatable, core image, or source statement library that is separate and distinct from the system library.

problem program:

1. The user's object program. It can be produced by any of the language translators. It consists of instructions and data necessary to solve the user's problem.

2. A general term for any routine that is executed in the data processing system's problem state; that is, any routine that does not contain privileged operations. (Contrasted with Supervisor.)

processing program: A general term for any program that is both loaded and supervised by the control program. Specifically, a collection of certain IBM supplied programs: the language translators, Linkage Editor, Librarian, Autotest, Sort/Merge and Utilities. All user written programs are processing programs. The term processing programs is in contrast to the term control program.

record: A general term for any unit of data that is distinct from all others when considered in a particular context.

reenterable: The attribute of a set of code that allows the same copy of the set of code to be used concurrently by two or more tasks.

relocatable: A module or control section whose address constants can be modified to compensate for a change in origin.

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 files, and control and processing programs.

restart: See Checkpoint/Restart.

self relocating: A programmed routine that is loaded at any doubleword boundary and can adjust its address values so as to be executed at that location.

self relocating program: A program that is able to run in any area of storage by having an initialization routine to modify all address constants at object time.

subtask: A task in which control is initiated by a main task by means of a macro instruction that attaches it.

supervisor: A component of the control program. It consists of routines to control the functions of program loading, machine interruptions, external interruptions, operator communications and physical IOCS requests and interruptions. The supervisor alone operates in the privileged (supervisor) state. It coexists in main storage with problem programs.

symbolic I/O assignment: A means by which problem programs can refer to an I/O device by a symbolic name. Before a program is executed, job control can be used to assign a specific I/O device to that symbolic name.

telecommunications: A general term expressing data transmission between remote locations.

teleprocessing: A term associated with IBM telecommunication systems expressing data transmission between a computer and remote devices.

track hold: A function for protecting DASD tracks that are currently being processed. When track hold is specified in the DTF, a track that is being modified by a task in one partition cannot be concurrently accessed by a task or subtask in another partition.

undefined record: A record having an unspecified or unknown length.

variable length record: A record having a length independent of the length of other records with which it is logically or physically associated. (Contrasted with fixed length record). It contains fields specifying physical and logical record lengths.

volume: That portion of a single unit of storage media that is accessible to a single read/write mechanism. For example, a reel of magnetic tape on a 2400-series magnetic tape drive or a disk pack on an IBM 2311 Disk Storage Drive.

Index

Indexes to systems reference library manuals are consolidated in the publication DOS Master Index, GC24-5063. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

DOS Supervi nd I/O Macros (S360-30) Printed in U.S.A. GC24-5037-11

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)