GC33-5373-2
File No. S370-30

Systems

DOS/VS Supervisor and I/O Macros

Release 29

IBM

Summary of Amendments

This edition reflects the availability of virtual storage enhancements and support of the following new devices:

System/370 Model 115
3203 and 5203 Printers
3340 Disk Storage
3540 Diskette I/O Unit
3780 Data Communication Terminal
5425 Multifunction Card Unit

In addition, technical changes and editorial corrections have been made throughout the book.

Changes in content are indicated by a vertical bar to the left of the change.

Third Edition (November 1973)

This is a major revision of, and obsoletes, GC33-5373-1.

This edition applies to Version 5, Release 29, of the IBM Disk Operating System, DOS/VS, and all the subsequent versions and releases until otherwise indicated in new editions or Technical Newsletters.

Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/360 and System/370 Bibliography,* GA22-6822, 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 may be addressed to IBM Laboratory, Programming Publication Dept., P.O.Box 210, 703 Boeblingen, Germany.

IS THIS THE RIGHT BOOK FOR YOU?

This book is intended as a reference for the programmer using DOS/VS macro instructions (macros). Both the DOS/VS Input/Output Control System (IOCS) macros and the DOS/VS supervisor macros are described. After a brief introduction to the use of macros, and a chapter on label processing, descriptions are given of the logical IOCS (LIOCS) macros for the access methods SAM, DAM, ISAM, and VSAM. Then follow descriptions of physical IOCS (PIOCS) macros, supervisor macros, multitasking macros, and program linkage macros.
Those familiar with DOS Versions 3 or 4 (up to and including Release 27) may note that this book is based on *DOS Supervisor & I/O Macros*, GC24-5037, as modified by TNL GN33-8689 and by *DOS Version 4*, GC33-5007. This manual includes information on the macro support for the following DOS/VS features that were not part of Release 27:

- VSAM macros
- virtual storage macros
- changes to the program loading macros
- modified and new interval timer macros
- modified and new dump macros
- macro usage for I/O devices specific to Models 115 and 125 such as 5425 Multi-Function Card Unit and the 5203 Printer
- macro usage for the IBM 3881 Optical Mark Reader and the IBM 3886 Optical Character Reader.
- macro usage for the IBM 3540 Diskette I/O Unit

As this book is intended for reference only, you should, before consulting it, be familiar with three others which introduce macro concepts and give important prerequisite information on macro usage:

Introduction to DOS/VS, GC33-5770

DOS/VS Data Management Guide, GC33-5372

DOS/VS System Management Guide, GC33-5371

In addition, you should be familiar with the device manuals for those devices which you intend to use.

Systems publications related to this one are listed below.

IBM System/370 Principles of Operation, GA22-7000

OS/VS and DOS/VS Assembler Language, GC33-4010

Guide to DOS/VS Assembler, GC33-4024

DOS/VS System Control Statements, GC33-5376

DOS/VS DASD Labels, GC33-5375

DOS/VS Tape Labels, GC33-5374

DOS/VS System Generation, GC33-5377

DOS/VS Serviceability Aids and Debugging Procedures, GC33-5380

Table of Contents

PART 1

INTRODUCTION

Macro Types and their Usage

Label Processing

MACRO TYPES AND THEIR USAGE

A macro is a single assembler language instruction which generates a sequence of assembler language instructions. The macros you code in your program are called the source program macros. The assembler uses what is called the macro definition to generate the sequence of instructions requested by the source program macro. Use of macros simplifies the coding of programs and reduces the possibility of programming errors.

Macro Definitions

A macro definition is a set of statements which defines the name of, format of, and conditions for generating a sequence of assembler language instructions from a single macro instruction. Macro definitions are stored in the Macro Sublibrary of the source statement library.

Source-Program Macros

Source-program macros are those you specify in your program; they indicate to the assembler which macro definition is to be called from the library. With a source-program macro you specify operands and parameters which the assembler uses, together with the called macro definition, to determine what assembler instructions to generate. There are two different types of source-program macros: supervisor macros and IOCS macros.

Supervisor Macros

These macros enable you to make use of functions performed by the supervisor. The RUNMODE macro, for example, determines whether your program is to run in virtual or real mode.

IOCS Macros

IOCS macros are divided into basic categories: imperative IOCS macros and declarative IOCS macros.

Imperative IOCS Macros

These macros identify what I/O operation you want to perform. The GET macro, for example, indicates that you want to obtain a record.

Declarative IOCS Macros

Declarative IOCS macros for all access methods except VSAM are of two related types--DTFxx macros and logic module generation (xxMOD) macros. Declarative IOCS macros for VSAM are the ACB, EXLST, and RPL macros. For further details on DTF declarative macros see the section on *DTF Declarative Macros*. The logic module generation macros and VSAM declarative macros are briefly discussed below and described in detail later on.

Logic Module Generation Macros. Logic module generation macros give information about the type of logic module to be generated. The module is the object code routine which will handle the conditions you specify in the module generation macro. For example, the CDMOD macro could generate a module to handle card input on a 2540 (as shown in the example in *Appendix B*).

VSAM Macros. The Virtual Storage Access Method (VSAM) has a set of declarative macros different from the DTFxx and logic module generation macros described above. VSAM declarative macros are ACB, EXLST, and RPL.

The ACB macro produces an access method control block which connects your program to the file. The access method control block contains information about the kind of processing to be done, and is usually specified only once in a program.

The EXLST macro produces an exit list containing addresses of routines you supply to handle special situations--such as an end-of-data routine, or a routine to manage I/O buffers. This macro is also usually specified only once in your program.

The RPL macro produces a request parameter list containing such information as a buffer address that is needed for execution of the VSAM imperative macros.

VSAM has no logic module generation macros. Standard VSAM modules are placed in the core image library during system generation, and loaded into your partition when a VSAM file is opened. The possibility of coding your own modules with a logic module generation macro, and of assembling or link-editing modules with your program, does not exist with VSAM.

Macro Processing

A direct relationship exists between the two basic parts of the macro system--between the source-program macros and the macro definitions--as described above.

During assembly, the source-program macro specifies which macro definition is to be called from the library. Figure 1-1 depicts schematically the source program before and after inclusion of the macro expansion. This is accomplished by a selection and substitution process using the general information in the macro definition and the specific information in the macro itself. The insertion consists of 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 of both 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 tested macro definitions. The macro instructions needed to use these definitions are described in this manual. You can also write your own macro definitions and include them in the source statement library of your system. For additional information on this subject, see *OS/VS and DOS/VS Assembler Language*, GC33-4010.

The IBM-supplied macros, which are explained in this book, are organized as follows:

Part 2. Sequential Access Method (SAM) LIOCS macros.

Part 3. Direct Access Method (DAM) LIOCS macros.

Part 4. Indexed Sequential Access Method (ISAM) LIOCS macros.

Part 5. Virtual Storage Access Method (VSAM) LIOCS macros.

Part 6. PIOCS macros.

Part 7. Supervisor macros.

Multitasking macros.

Program Linkage macros.

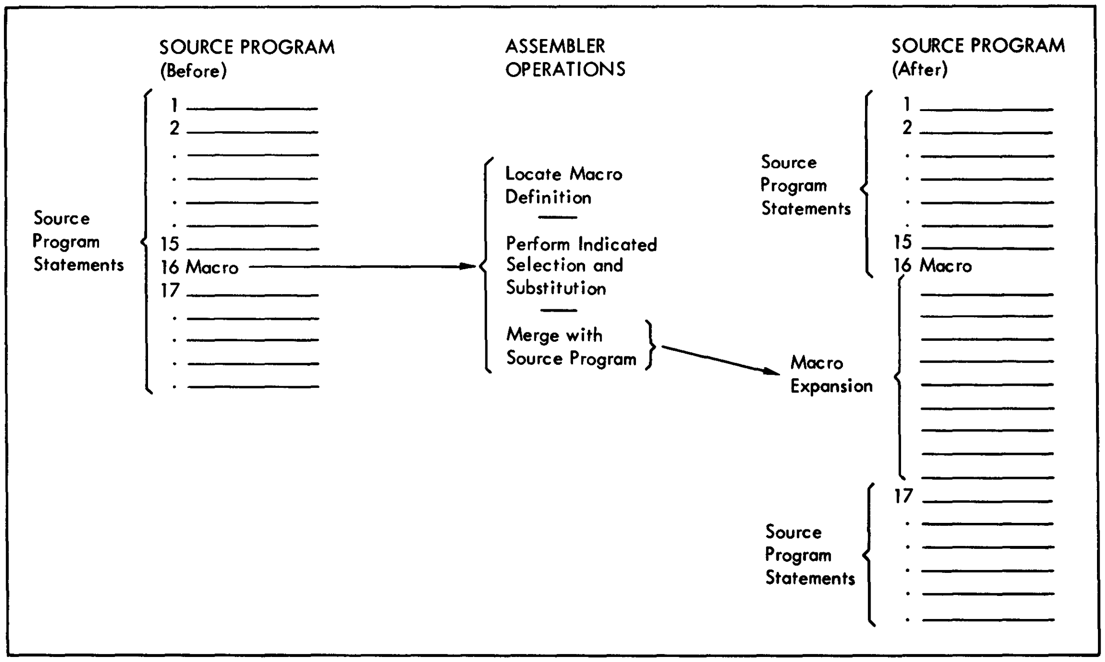

SOURCE PROGRAM (Before)
1
2
15
16 Macro
17
Source Program Statements
ASSEMBLER OPERATIONS
Locate Macro Definition
Perform Indicated Selection and Substitution
Merge with Source Program
SOURCE PROGRAM (After)
Source Program Statements
Macro Expansion
Source Program Statements

Figure 1-1 **Schematic of macro processing**

DTF Declarative Macros

With all access methods except VSAM, whenever you use logical IOCS imperative macros (such as GET, PUT, READ, or WRITE) in your program to control the input/output of records in a file, that file must be defined by a declarative macro called a DTF (Define The File). The DTF macro describes the characteristics of the file, indicates the type of processing for the file, and specifies the virtual storage areas and routines to be used in processing the file.
For example, if a GET is issued, the DTF macro supplies such information as:

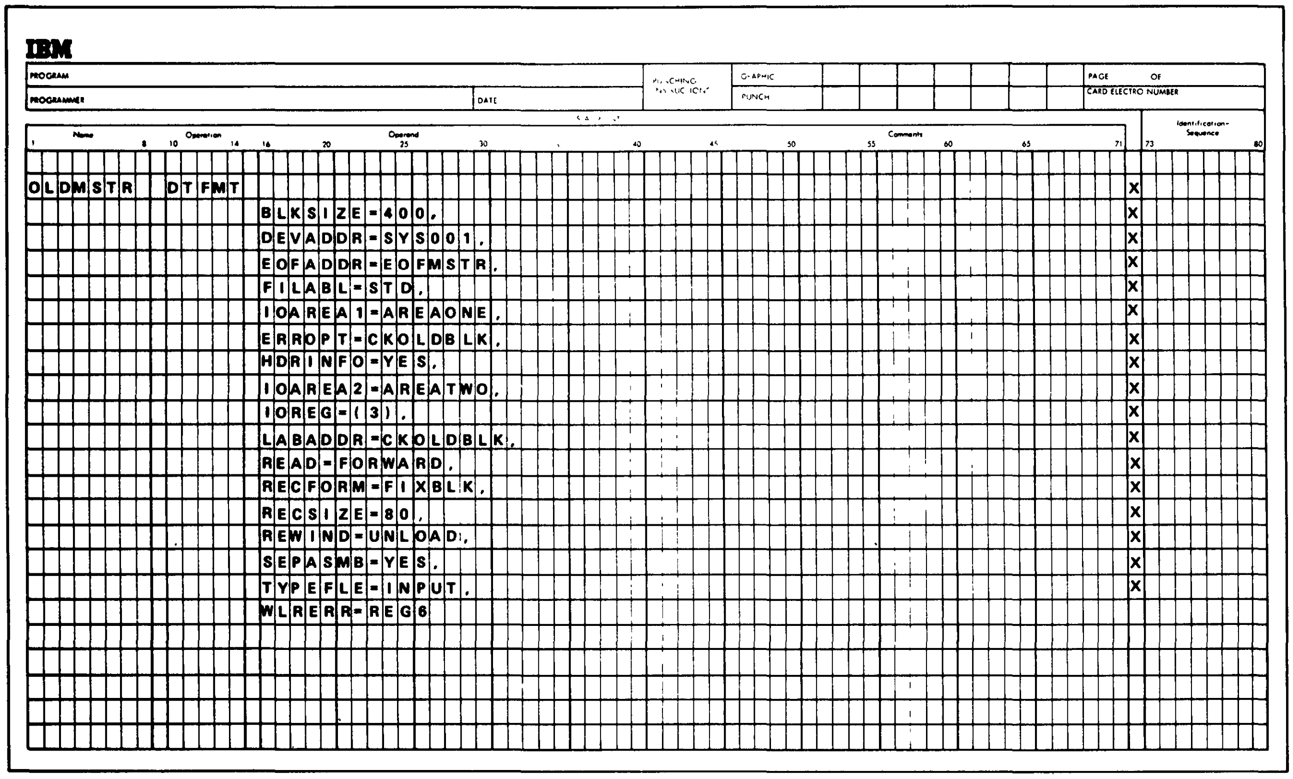
- Record type and length.
- Input device from which the record is to be retrieved.
- Address of the area in storage where the record is to be located for processing by your program.

Device-oriented DTF macros are available for defining files processed by LIOCS (Logical Input/Output Control System). An additional DTF macro is available for magnetic tape or DASD files processed by PIOCS (Physical Input/Output Control System). Figure 1-2 contains an example of a DTF source statement. For LIOCS operations, the DTF macro used depends on the type of processing that will be performed and upon the type of device used. For detailed information on LIOCS and PIOCS, please refer to the *DOS/VS Data Management Guide*, GC33-5372.

The following is a list of DTFs available for the various types of processing.

Processing with SAM

This applies to input/output with serial or diskette devices, or with direct access devices when records are processed sequentially. (ISAM and VSAM may also be used with direct access devices when records are to be processed sequentially). The macros used for SAM processing are listed by device name in alphabetical order in Figure 1-3. For details refer to Part 2 of this manual.



```
OLDMSTR  DTFMT                                                         X
               BLKSIZE=400,                                            X
               DEVADDR=SYS001,                                         X
               EOFADDR=EOFMSTR,                                        X
               FILABL=STD,                                             X
               IOAREA1=AREAONE,                                        X
               ERROPT=CKOLDBLK,                                        X
               HDRINFO=YES,                                            X
               IOAREA2=AREATWO,                                        X
               IOREG=(3),                                              X
               LABADDR=CKOLDBLK,                                       X
               READ=FORWARD,                                           X
               RECFORM=FIXBLK,                                         X
               RECSIZE=80,                                             X
               REWIND=UNLOAD,                                          X
               SEPASMB=YES,                                            X
               TYPEFLE=INPUT,                                          X
               WLRERR=REG6
```

Figure 1-2 Sample DTFMT macro

File to be processed on	Macro
Card device	DTFCD
Console printer-keyboard	DTFCN
DASD sequential	DTFSD
Device independent	DTFDI
3540 Diskette I/O Unit	DTFDU
Display operator console	DTFCN
Magnetic reader (MICR)	DTFMR
Magnetic tape	DTFMT
Optical reader (excluding 3886)	DTFOR
3886 Optical character reader	DTFDR
Optical reader/sorter	DTFMR
Paper tape reader	DTFPT
Paper tape punch	DTFPT
Printer	DTFPR
Sequential DASD	DTFSD
Serial type device (for compatibility only)	DTFSR

Figure 1-3 **SAM declarative macros**

Processing with DAM

Whenever a file on a direct access device is to be processed by DAM, DTFDA must be used. For details refer to Part 3 of this manual.

Processing with ISAM

Whenever a file on a direct access device is to be organized or processed by ISAM, DTFIS must be used. For details refer to Part 4 of this manual.

Processing with PIOCS

When PIOCS macros (EXCP, WAIT, etc.) are used for a file, the DTFPH macro is required only if standard labels are to be checked or written on a file on a direct access device or magnetic tape, or if the file on a direct access device is file-protected. For details refer to Part 6 of this manual.

Referencing the DTF Table

A DTFxx macro generates a DTF table that contains indicators and constants describing the file. You can reference this table by using the symbol filename+constant, or filenamex where x is a letter. When referencing the DTF table, you must ensure addressability through the use of an A-type constant, or through reference to a base register. Should you need to reference a DTF table in your program, you can obtain detailed information on the layout of DTF tables in the *LIOCS Program Logical Manuals*, SY33-8559, SY33-8560 and SY33-8561.

Symbolic Unit Addresses in the DTFxx Macro

In most of the DTF macros you can specify a symbolic unit name in the DEVADDR operand. This symbolic unit name is also used in the ASSGN job control statement to assign an actual I/O device address to the file. For files on diskettes or direct access devices, the symbolic unit name is supplied in the DEVADDR operand and/or with the EXTENT job control statement (if both are provided the EXTENT specification overrides the DEVADDR specification).

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

Figure 1-4 shows the relationship between the source program, the DTF table, and the job control I/O assignment.

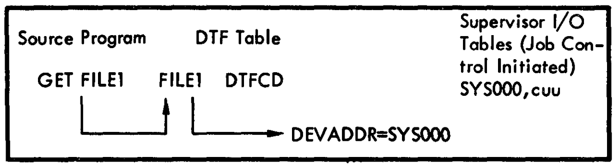


Figure 1-4 **Relationship between source program, DTF table, and job control I/O assignment**

The fixed set of symbolic names that can be used with a DTF macro for a program in any partition is the same and is represented by SYSxxx. Programs in different partitions can reference the same logical unit providing different devices, or DASD extents, are assigned.

These symbolic units are divided into system logical units and programmer logical units.

System Logical Units

SYSRDR	Card reader, magnetic tape unit, disk extent, or diskette extent primarily for job control statements.
SYSIPT	Card reader, magnetic tape unit, disk extent, or diskette extent as the primary input unit for programs.
SYSPCH	Card punch, magnetic tape unit, disk extent, or diskette extent as the primary unit for punched output.
SYSLST	Printer, magnetic tape unit, disk extent, or diskette extent as the primary unit for printed output.
SYSLOG	Console printer-keyboard or display operator console for operator messages and for logging job control statements. Can also be assigned to a printer.
SYSLNK	Disk extent as input to the linkage editor.
SYSVIS	Disk extent for the page data set.
SYSCAT	Disk extent for the VSAM catalog.
SYSCLB	Disk extent for a private core image library.
SYSRLB	Disk extent for a private relocatable library.
SYSUSE	Disk extent used by the system for internal purposes.
SYSSLB	Disk extent for a private source statement library.
SYSREC	Disk extent for error log records and for the hard copy file of the display operator console.
SYSIN	**Can** be used if you want to assign SYSRDR and SYSIPT to the same card reader or magnetic tape unit. **Must** be used if you want to assign SYSRDR and SYSIPT to the same disk extent.
SYSOUT	**Must** be used if you want to assign SYSPCH and SYSLST to the same magnetic tape unit. **Cannot** be used to assign SYSPCH and SYSLST to disk because these two units must refer to separate disk extents. SYSIN and SYSOUT are valid only to job control and cannot be referenced in a user program. Examples for the use of SYSIN and SYSOUT are given in the section *System Files on Tape or Disk* in the *DOS/VS Systems Management Guide,* GC33-5371.

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 the system: SYSmax=255-NPART x 14 where NPART is the number of partitions. Each of these programmer logical units can be assigned to any partition without a prescribed sequence, except when using DAM (see *Note*, below). For a given partition, the maximum number of programmer logical units is equal to SYSmax minus the sum of all programmer logical units assigned to other partitions.

Note: For DAM the EXTENT job control statements must be supplied in ascending order, and the symbolic units for multivolume files must be assigned in consecutive order.

Each declarative macro requiring a symbolic unit to be specified has a list of symbolic units that are valid for that macro. In that list, SYSnnn represents programmer logical units, while SYSxxx indicates either a system or a programmer logical unit.

For files processed by either SAM or DAM, only one symbolic unit may be assigned to all extents of a file on one volume.

In physical IOCS, the symbolic unit name is specified in the CCB and in the DTFPH macros. Instead or additionally it is specified with the EXTENT job control statement. (If more than one of these is used to provide the specification, an EXTENT specification overrides a DTFPH specification, and a CCB specification overrides an EXTENT and/or a DTFPH specification.)

Figure 1-5 shows the relationship between the source program and the job control I/O assignment.

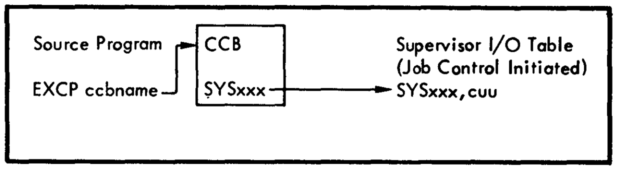


Figure 1-5 **Relatioship between source program and job control I/O assignment**

Logic Module Generation Macros

Each DTF except DTFCN DTFPH, and DTFSR must link to an IOCS logic module. A logic module is generated by a logic module generation (xxMOD) macro. The modules provide the necessary instructions to perform the input/output functions required by your program. For example, the module reads or writes data, tests for unusual input/output conditions, blocks or deblocks records if necessary, or places records in a work area. Most imperative macros enter a logic module to perform the necessary function.

Providing Logic Modules:

There are two ways of providing logic modules for your DTFs:

1. Do not code the logic module generation macro needed by your DTF(s). In this case, the standard logic modules needed for your installation should have been assembled and cataloged (in the relocatable library) at system generation time. You can then autolink needed modules from the relocatable library at link-edit time.

2. Code the logic module generation macro needed by your DTF(s), assembling it either in-line with your program or supplying it at link-edit time.

Keeping Modules Small

Some of the module functions are provided on a selective basis, according to the parameters specified in the xxMOD macro. If you code the xxMOD macro yourself, you have the option of selecting or omitting some of these functions according to the requirements of your program. If, as described above, you do not code the xxMOD macro yourself, IOCS will automatically select or omit the appropriate functions. In either case the omission of unneeded functions saves storage space for a particular module.

Note: If you issue 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.

Subsetting/Supersetting

Some modules may be subset modules to a superset module. A superset module is one which performs all the functions of its subset or component modules, avoiding duplication and thereby saving storage space. The functions required by several similar DTFs (that is, several DTFCDs, or several DTFPRs, etc.) are thus available via a single xxMOD macro, even if the DTFs have slightly different parameters. An example is shown in Figure 1-6.

Superset Module Functions	**Subset Module Functions**	**Subset Module Functions**
Optional use of CNTRL macro	CNTRL macro cannot be used	Optional use of CNTRL macro
Workarea and I/O area processing	Workarea and I/O area processing	I/O area processing only
Support of printer overflow	No printer overflow support	Support of printer overflow
Support of user-specified error actions	Support of user-specified error actions	Support of user-specified error actions

Figure 1-6 **Subset and superset module example**

If you do not code the logic modules yourself, IOCS will automatically perform all subsetting/supersetting which is possible.

If you code the logic modules yourself, subsetting/supersetting can be achieved by coding a single xxMOD macro which contains all of the functions needed by all of the DTFs which will use that macro. In this case you may either:

- Not name the module and let IOCS name it for you--that is, specify no name for the xxMOD macro and also no MODNAME operands in the DTFs; or

- Name the module--specifying a name for the xxMOD macro and also specifying the same name in the MODNAME operands of all of the DTFs which will use that module.

Subsetting/supersetting cannot be performed if you supply an xxMOD macro for each DTF of a given device type. In this case:

- If you did not name the modules, the assembler program will detect a double declaration error condition, or

- If you did name the modules, they will be generated without any subsetting/supersetting.

Interrelationship of the Macros

Figure 1-7 shows the relationship between the program, the DTF, and the logic module. Imperative macros initiate the action to be performed 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.

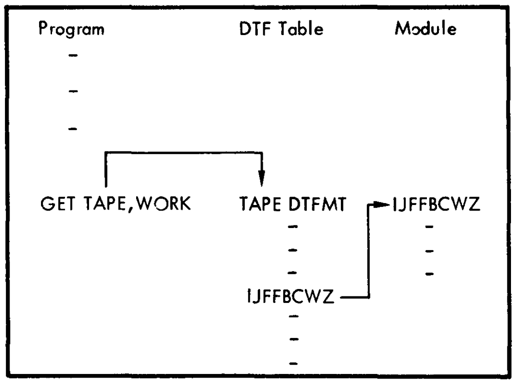


Figure 1-7 **Relationship between program, DTF and logic module**

Linkage between the program, DTF, and logic module is accomplished by the assembler and the linkage editor.

Module Names

As mentioned under *Logic Module Generation Macros*, you can have IOCS provide a name for the required logic module, or you can specify that name. Both methods are discussed below.

IOCS Supplies the Name
In order to make use of this facility omit the MODNAME operand from the DTF macro; the IOCS macro will then generate a standard module name as determined by the functions required by the DTF.

Likewise, if you code your own module, the name should be omitted from the name field, and IOCS will generate a standard module name matching that referenced in the DTF.

Standard module names used by IOCS are given under *Standard CDMOD Names*, *Standard DIMOD Names*, etc., following the discussion of the appropriate xxMOD macro.

IOCS Subset/Superset Names IOCS performs subsetting/supersetting of modules with standard module names by collecting the services required by the DTFs and generating a single module with different entry points corresponding to the standard module names. If you are interested in seeing how IOCS forms subset/superset names, charts showing the name-building conventions are given throughout the book for the various logic modules-- under *Subset/Superset CDMOD Names Subset/Superset DIMOD Names*, etc., following the discussion of the appropriate module. The following is a model for these charts:

```
          *  +  +  +  *
I  J  x   F  B  C  W  Y
          U  Z  Y  Z  Z
          V  +  +
          W  E  N
             Z  S
                Z
```

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

The letters indicate functions which can be performed by the logic module (these are fixed for a given module and are explained in the sections *Standard CDMOD Names*, *Standard DIMOD Names*, etc.). If a module name were composed of letters from the top row exclusively, it could only be a superset name; and names including letters from the second or lower rows would then be subset names to the top-row superset name. For example, the module IJxWESZZ is a subset module to superset module IJxWENZZ. IJxWEZZZ is another subset module to superset module IJx-WENZZ. Similarly, IJxWEZZZ is also a subset module to superset module IJxWESZZ.

An asterisk (*) over a column indicates that no subsetting or supersetting is permitted, while a plus (+) sign in a column indicates that both are permitted. 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. If a column is divided by plus and/or asterisk signs into sets, then the vertical arrangement of letters within each set of a column is in alphabetical order.

You Supply the Name
Specify the name of the module in the MODNAME operand of the DTF macro. A module with this name must then be present in your program, or be supplied to your program when it is link-edited. Subsetting/supersetting will occur if one module contains all of the functions needed by all of the DTFs which will use the module (all must reference it by the same name).

Nothing is gained by giving your modules standard IOCS names (see *IOCS Supplies the Name*, above), for IOCS will supply the same name for you if you let it name the modules. Should you decide to name your modules, use names which are meaningful to you in the context of your program.

Link-Editing Logical IOCS Programs

You have the option of assembling your DTFs, and any logic modules which you code yourself, either with your main program or separately for later link-editing with the main program. These possibilities are discussed below and are illustrated in *Appendix B*.

Program, DTF, and Logic Module Assembled Together

If you assemble DTFs and logic modules with the main program, the linkage editor searches the input stream and resolves the symbolic linkages between tables and modules. This is accomplished by external-reference information (V-type address constants generated in DTF tables) and the control section definition information (CSECT definitions in logic modules). Further information on linkage editing can be found in the section *Linkage Editor* of the *DOS/VS System Control Statements*, GC33-5376.

Program, DTF, and Logic Module Assembled Separately

Specify the operand SEPASMB=YES in the DTF macro or xxMOD macro which is to be separately assembled. For DTFs which are separately assembled, there are some symbolic linkages which you must define yourself in the form of EXTRN-ENTRY symbols. See *Appendix B* for a full description of which symbolic linkages you must define yourself.

Supplying the SEPASMB=YES operand in a DTF macro 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. Specifying the SEPASMB=YES operand in an xxMOD macro causes a CATALR card with the module name to be punched ahead of the object deck and defines the module name as an ENTRY point in the assembly. In either case, a START card must not be used in a separate assembly.

Using the Relocatable Library

As stated earlier, considerable coding effort is saved if logic modules are cataloged in the relocatable library. The same applies to DTFs. Using DTFs cataloged in the relocatable library requires that you take care in naming the DTFs--that is, that you develop a set of standard names and then use them both for your DTFs and in all references your program makes to the DTFs. However, should you decide to name modules yourself, instead of letting IOCS do it, then you make sure that you refer to precisely those modules in your DTFs by using their exact names (see *Module Names* above).

If, at system generation time, a standard set of logic modules needed by your installation has been generated, autolinking the appropriate modules to your DTFs presents no problem. This is particularly true if both the modules and DTF references to them use standard module names.

Using logic modules which you named yourself--as opposed to those named by IOCS--cataloged in the relocatable library requires care. You should verify that the desired modules have been cataloged in the library by consulting a DSERV listing of the library. The linkage editor can perform an autolink only if there is an exact match of module names specified in the DTFs and the names of the modules themselves.

Self-Relocating Programs and IOCS

The Relocating Load feature, an option in supervisor generation, makes it unnecessary for you to write your own self-relocating programs. If, however, your supervisor does not have this feature and you want to make IOCS imperative macros (except VSAM macros) and supervisor macros self-relocating you must do the following:

1. Use the OPENR and CLOSER macro.

2. Use register notation within all your imperative macros (see *Register Notation* later in this chapter).

Appendix D gives detailed instructions on writing self-relocating programs.

Macro Format

Macros, like assembler statements, have a name field, operation field, and operand field as shown below. Comments can also be included as in assembler statements.

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

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

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, and no embedded blanks are allowed. If an operand is to be omitted in the macro 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 (any nonblank character) if the operands fill the operand field and overflow onto another 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 expressed as a keyword value. An operand written in keyword format has this form for example:

```
LABADDR=MYLABELS
```

where LABADDR is the keyword, MYLABELS is the parameter, and LABADDR=MYLABELS is the complete operand. The keyword operands in the macro 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 except for the last operand of the macro. Or, they may be punched in separate cards as in Figure 1-2.

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.

For additional information on coding macro statements, see *OS/VS and DOS/VS Assembler Language*, GC33-4010.

Cards for Declarative Macros

The operands of the DTFxx and the module generation macros can be punched in a set of cards in the assembler format. Figure 1-2 shows an example of the cards used for a DTFMT macro. The DTF macros may be assembled in any order.

The first 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 using IJ as the first two letters when defining symbols as they may conflict with IOCS symbols beginning with IJ. Avoid symbols that are identical to a filename plus a single character suffix because IOCS generates symbols by concatenating the filename with an additional character--for the filename RECIN, for example, IOCS generates the symbols RECINS, RECINL, etc.

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 *Module Names*, above.

- The mnemonic operation code of the macro in the operation field.
- Keyword operands in the operand field, if desired.
- A continuation punch in column 72, if a continuation card is 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--or, for the last detail card, if a space is left after the last operand.

Notational Conventions

The following conventions are used in this book to illustrate the format of macros:

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 which you must supply. 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. Stacked options contained within brackets represent alternatives, one of which can be chosen for example:

$\begin{bmatrix}\text{name}\\ \text{label}\\ \text{address}\end{bmatrix}$	A name-field symbol in this assembly, or an operand of an EXTRN statement, or * (the Location counter).

5. An ellipsis (a series of three periods) indicates that a variable number of items may be included.
6. Stacked 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)

7.	filename	Symbol appearing in the name field of a DTF macro.
8.	$\begin{Bmatrix}\text{n}\\ \text{(r)}\end{Bmatrix}$	Self-defining value, such as 3, X'04', (15), B'010'.
9.	length	Absolute expression, as defined in *OS/VS and DOS/VS Assembler Language,* GC33-4010.
10.	$\begin{Bmatrix}\underline{\text{A}}\\ \text{B}\\ \text{C}\end{Bmatrix}$	Underlined elements represent an assumed value in the event a parameter is omitted.
11.	$\begin{Bmatrix}\text{name}\\ \text{(r)}\end{Bmatrix}$	Ordinary register notation. Any register except 0 or 1.
12.	$\begin{Bmatrix}\text{name}\\ \text{(0)}\\ \text{name}\\ \text{(1)}\end{Bmatrix}$	Special register notation (ordinary register notation can be used).

Register Notation

Certain operands can be specified in either of two ways:

1. You may specify the operand directly and produce code which cannot be executed in the SVA because it is not reentrant.
2. You may load the address of the value into a register before issuing the macro. This way the macro is reentrant and will be executed in the SVA. 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, you must specify the register in the macro. (The registers that can be used for this purpose are discussed under *Register Usage*, below.) This method is known as ordinary register notation.

When the macro is assembled, instructions are generated to pass the information contained in the specified register 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.

You can save both storage and 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 use 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, you can use ordinary register notation and the macro expansion will contain an extra LR instruction.

The format description for each macro shows whether special register notation can be used, and for which operands. The following example indicates that the filename operand can be written as (1) and the workname operand as (0):

$$\text{GET} \begin{Bmatrix} \text{filename} \\ (1) \end{Bmatrix} \left[, \begin{Bmatrix} \text{workname} \\ (0) \end{Bmatrix} \right]$$

If either of these special register notations is used, your program must load the designated parameter register before executing 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 your program only under certain conditions.

The following paragraphs describe the general uses of these registers by IOCS, but the description is not meant to be allinclusive. For more information on subroutine linkage through registers, refer to the *Linkage Registers* section of the *Program Linkage Macros* chapter. In addition, special applications, such as a MICR 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. However, if you use these registers for computations not completed before IOCS requires them, you must save their contents and reload them later when required.

Register 13:
Control program subroutines, including logical IOCS, use this register as a pointer to a 72-byte, doublewordaligned save area. When using the CALL, SAVE, or RETURN macros, you 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 program) from DTF routines, called programs, and your subroutines. Register 15 contains the entry point into these routines, and is used as a base register by the OPEN, OPENR, CLOSE, CLOSER, and certain DTF macros.

IOCS does not save the contents of these registers before using them. If you use these registers, you must either save their contents yourself (and reload them later) or finish with them before IOCS uses them.

Registers for Your Use

Registers 2-12 are available for general usage. There are, however, a few restrictions.

The assembler instruction for translate and test (TRT) makes special use of register 2. It is your 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 your program. After the TRT instruction has been executed, you can then restore the contents of register 2.

If an ISMOD logic module precedes a USING statement or follows your program, the use of registers 2-12 remains unrestricted even at assembly time. However, if the ISMOD logic module lies within the problem program, you should issue the same USING statement (which was issued before the logic module) directly following the logic module. 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

This section provides the information you need on order to process labels with the IOCS macros. More detailed information about labeling conventions and label processing considerations will be found in the *DOS/VS Data Management Guide,* GC33-5372, *DOS/VS DASD Labels,* GC33-5375, and *DOS/VS Tape Labels,* GC33-5374.

DASD Standard Labels

Labels are required when processing files on direct access devices. Accordingly you must supply both a DASD label (DLBL) job control statement for each logical file to be processed, and one or more extent (EXTENT) job control statements to allocate one or more areas on a direct access device. Also, when processing standard labels, a LBLTYP job control statement is required to define virtual storage needed at link-edit time for label processing for files defined by DTFDA or DTFIS macros or by a DTFPH macro with the MOUNTED=ALL operand (more information will be found in *DOS/VS System Control Statements,* GC33-5376).

OPEN and OPENR Macro Processing

The OPEN and OPENR macros use the information supplied in the DLBL and EXTENT job control statements as well as information from the DTF for the file.

For input, the extent(s) for a file must either coincide with, or be within, the existing extent(s) as defined in the Volume Table of Contents (VTOC). This is necessary 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 without your explicit request, except when an internal system file (IJSYS) overlays an identical system file. However, if OPEN (or OPENR) determines that the output file will overlay an existing file that has expired, the macro OPEN (or OPENR) deletes the expired label(s) from the VTOC. This in effect removes the file from the volume. In a multi-volume file, the file may be removed from all the volumes that it occupies or from only some of the volumes.

If OPEN (or OPENR) determines that an existing file to be overlaid by the output file has not expired, the old file cannot be destroyed automatically. In this case, the following actions are possible. For SAM, DAM, physical IOCS, or work file processing:

1. Delete the unexpired file, or
2. Terminate the job, or
3. Bypass the extent. That extent and any remaining extents for that file are bypassed and the job is terminated.

For ISAM processing:

1. Delete the unexpired file, or
2. Terminate the job.

Reopening a File
If further processing of a file which your program has closed 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 (or OPENR). When a file is processed by physical IOCS with the DTFPH operand MOUNTED=SINGLE, IOCS opens the next extent specified by your EXTENT job control statement. When a file is processed by DAM (defined by the DTFDA macro), by ISAM (defined by the DTFIS macro), or by physical IOCS with the DTFPH operand MOUNTED=ALL specified, all label processing is repeated and all extents are again made available.

For more information on label processing see the discussion of the OPEN (or OPENR) macro under the appropriate access method.

End-of-Volume Processing

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 your 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, IOCS processes the standard labels on the next volume and branches to your LABADDR routine to process user standard header labels. After the header labels are processed, IOCS continues to process the data.

End-of-File Processing

Output Files
When all records for a logical output file have been written, the CLOSE (or CLOSER) macro must be issued to perform normal end-of-file processing. IOCS then branches to your LABADDR routine (if provided) to write user trailer labels, and the file is closed. If the end of the last extent specified for the file is reached before the CLOSE (or CLOSER) macro is issued, IOCS assumes an error condition.

Input Files
IOCS determines an end-of-file condition for a logical input file either by the ending address of the last extent specified for the file in the EXTENT job control statement, or by an end-of-file record read from the file. For SAM 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. You can then test this byte and take action necessary to close your file. However, when processing in random order you must determine the end-of-file by checking filenameC (DTFIS) or ERRBYTE (DTFDA).

User Standard Labels

If user standard labels are desired, you must supply a LABADDR routine, unless processing with physical IOCS. SAM and DAM process both user header and trailer standard labels. ISAM does not process user standard labels. User labels cannot be created for a file whose first extent is a split cylinder extent. DAM writes a user trailer label only on the first volume of a multi-volume file.

When the LABADDR routine is entered, IOCS loads an alphabetic O, V, or F into 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. Your 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; this is the address you use if checking labels, or from which you move the label to your program's label area if you are modifying labels.

Within the LABADDR routine, you cannot issue a macro that calls a transient routine (such as OPEN (or OPENR), CLOSE (or CLOSER), DUMP, CANCEL, and CHKPT). For multi-volume files, the LABADDR routine should save registers 14 and 15 upon entry, and restore them before issuing the LBRET macro to return to IOCS.

Writing User Standard Labels on Disk
When you specify LABADDR, OPEN (or OPENR) reserves the first track of the first data extent as a user label area. At least one user header and trailer label must be written if the access method is to process it. For DAM, 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). You can use the other 76 bytes as you wish. The maximum number of user standard header or trailer labels is eight for files on all DASDs except the 2321, and five for files on the 2321. IOCS stores the label information (UHLx or UTLx) that it generates in bytes 1-4 of the IOCS label area. You can test this information, in addition to registers 0 and 1, to determine the type and number of the label. (The label formats will be found in *DOS/VS DASD Labels*, GC33-5375.)

In your area of virtaul storage, build either an 80-byte label, leaving the first four bytes free, or simply a 76-byte label. For the 80-byte label, load the address of the label area into register 0; for the 76-byte label, load the label area address minus four into register 0. Then issue the LBRET macro. When the label is moved into the IOCS area, IOCS adds four to the address in register 0, thus only moving the 76 bytes of user information into the IOCS label area.

When the label is ready to be written, the LBRET macro returns control to IOCS. If LBRET 2 is used, OPEN (or OPENR) writes the label and returns control to your label routine unless the maximum number of labels has been written. If LBRET 1 is used, the label set is considered complete and no more labels can be created.

When IOCS receives control, the IOCS routines move the label from the address you loaded into 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 your label routine unless LBRET 1 was used. If the maximum number of labels has been created, IOCS automatically terminates building of the label set.

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

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

If you wish to end label checking before all the labels have been read, the LBRET 1 macro may be issued.

Diskette Labels

Labels are required when processing files on diskette I/O units. Accordingly you must supply a DASD label (DLBL) job control statement for each logical file to be processed, and one or more extent (EXTENT) job control statements (more information will be found in *DOS/VS System Control Statements*, GC33-5376.

OPEN and OPENR Macro Processing

The OPEN or OPENR macro uses the information supplied in the DLBL and EXTENT job control statements, information from the DTF for the file, and information from the file label on the diskette.

For input, extent limits are taken directly from the file label in the VTOC on the diskette; extent limits provided in the extent statement(s) are ignored. Similarly for output files, the extent limits for the file are determined by OPEN (or OPENR) from available space on the diskette extent limits provided by the user are ignored. If the name of the output file to be created is the same as that of an unexpired or write-protected file already present on the volume, OPEN (or OPENR) will cause the job to be canceled. You will not be allowed to request that a duplicate file (unexpired or write-protected) be deleted. If the duplicate file has expired and is not write-protected or if a duplicate file is not being created, OPEN (or OPENR) will allocate space for the file, starting at the cylinder following the end of the last unexpired or write-protected file on the diskette. If expired and non-write-protected files are overlapped by this allocation, their labels are deleted from the VTOC.

End-of-Volume Processing

During processing, IOCS recognizes an end-of-volume condition when end-of-extent is reached on a volume and more extents are available. When this occurs, IOCS processes the standard labels on the next volume and continues to process the data.

End-of-File Processing

Output Files
When all records for an output logical file have been written, the CLOSE (or CLOSER) macro must be issued to perform normal end-of-file procedures. If the end of the last extent specified for the file is reached before the CLOSE (or CLOSER) macro is issued, IOCS assumes an error condition.

Input Files
IOCS determines an end-of-file for an input logical file by the end-of-data address. This address is specified in the file label in the VTOC of the last diskette of the file (first diskette if this is not a

multi-volume file). IOCS branches to the EOFADDR routine upon an end-of-file condition.

Tape Labels

Tape Output Files

For output on magnetic tape, OPEN (or OPENR), CLOSE (or CLOSER), or an end-of-volume condition rewinds the tape as specified in the DTFSR or DTFMT REWIND operand. No rewind can be defined in the DTFPH macro, and tape positioning depends on the labels to be processed and is your responsibility.

If you write any user standard labels, a LABADDR routine must be supplied. (For ASCII tape files, the LABADDR routine may only be used to process user standard labels.) Your LABADDR routine, specified in the DTF, cannot issue a macro that calls a transient routine. For example, OPEN (or OPENR), CLOSE (or CLOSER), DUMP, CANCEL, and CHKPT cannot be issued. Also when processing multi-volume files, your 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 of a file are processed, CLOSE (or CLOSER) 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 you have not specified any positioning, 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 a CLOSE (or CLOSER) is issued, logical IOCS prepares for closing the file by ensuring that all records are written on the tape. If you issue another PUT, indicating that more records are to be written on this output file, EOV procedures are initiated. If you issue a CLOSE (or CLOSER), 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 this short block. You 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 EOV labels.

When IOCS detects the EOV condition, it switches to an alternate unit as designated in an ASSGN job control 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* in the *Imperative Macros* section of the *Sequential Access Method Macros* chapter).

Writing Standard Labels on Tape

When standard labels are written (DTFMT or DTFSR FILABL=STD or DTFPH TYPFLE=OUTPUT), you must supply the TLBL job control statement for standard label information. Also, when standard labels are processed, a LBLTYP job control statement is required to define virtual storage needed at link-edit time for label processing (more information will be found in *DOS/VS System Control Statements*, GC33-5376).

When an OPEN (or OPENR) macro 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 you supply in the tape label statement. The volume label is not rewritten, altered, or updated.

A comparison is made between the 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 (are) rewritten according to the specified density.

If an output file begins in the middle of a reel, it is your responsibility to properly position the tape immediately past the tapemark for the preceding file before issuing the OPEN (or OPENR) macro. The MTC command can be used to do this. If the tape is improperly positioned, IOCS issues an appropriate message to the operator.

If user standard labels are written, the LABADDR operand must be specified in the DTF (see *Tape Output Files*, above). 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.

Your 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; this is the address you use if checking labels, or from which you move the label to your program's label area if you are modifying labels.

Note: For ASCII files, you process your 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 you. 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 (apostrophe). The remaining 76 bytes can be used as desired. It is your 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.

You should build your labels in your area of virtual storage, and load the address of the label into register 0 before issuing the LBRET macro.

When the label is ready to be written, you issue the LBRET macro, which returns control to IOCS. If LBRET 2 is used, IOCS writes the label and returns control to your 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 (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, your program must accumulate the block count, if desired, and supply it to IOCS for inclusion in 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 among 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, your 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*, above).

Writing Nonstandard Labels on Tape

To write nonstandard labels, you must specify FILABL=NSTD and LABADDR=name. When the file is opened, the tape must be positioned to the first label that you wish to process. The MTC job control statement can be used to skip the necessary number of tapemarks or records to position the file. You must also write your own channel program and use physical IOCS macros to transfer the labels from virtual storage onto tape. For an example on reading, writing, and checking with unstandard labels see *Appendix C*.

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 bytes 6 and 7 of the CCB for these values (the format of the CCB is given in the *Physical IOCS Macros* chapter). 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.

Your LABADDR routine can then test this character to determine the type of labels to be written.

In your LABADDR routine, physical IOCS macros must be used to transfer labels from virtual storage onto tape. For each label record, a CCB and CCW must be established, and the EXCP macro must be issued (see the *Physical IOCS Macros* chapter). Other logical IOCS macros can be used for any processing other than the transfer of the labels from virtual storage to tape. Additional LABADDR routine restrictions are discussed above.

After all labels are written, you return control to IOCS by use of the LBRET 2 macro. IOCS processing after LBRET is executed, has been discussed above.

Note: Nonstandard labels are not permitted with ASCII.

Writing Unlabeled Files on Tape

If you use unlabeled files, you should specify FILABL=NO and omit TPMARK=NO in the DTF to improve the efficiency of your program. Your file must be positioned properly with the MTC job control statement, if necessary, and writing begins immediately. Other processing information can be found under *Tape Input Files*, below.

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

Tape Input Files

For a magnetic tape input file, the macros OPEN (or OPENR), CLOSE (or CLOSER), or an end-of-volume condition cause the tape to be rewound as specified by the DTFSR of DTFMT REWIND parameter. No rewind can be defined in the DTFPH macro. Tape positioning depends on the labels to be processed and is your responsibility.

If any labels other than standard labels are to be checked, a LABADDR routine must be supplied. Your LABADDR routine, specified in the DTF, cannot issue a macro that calls a transient routine. This is the same as for the tape output files.

When an end-of-file condition occurs, IOCS branches to your EOFADDR routine specified in the DTF. Generally, you issue a CLOSE (or CLOSER) in this routine to initiate a rewind operation for the tape (as specified by the DTF REWIND operand), and deactivate the file. If CLOSE (or CLOSER) is issued before the end of data is reached, the rewind option is executed and the file is deactivated without any subsequent label checking.

When logical IOCS reads 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 yourself, 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 on Tape* or *Checking Nonstandard Labels on Tape*, below). If labels are not processed, your 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 an ASSGN job control 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), you must issue an OPEN (or OPENR) macro for the new volume. Then, IOCS checks the header label(s) and processing continues.

In some cases, you may desire to force an end-of-volume condition at a point other than at the normal tapemark. You 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 for such cases (see *FEOV Macro* in the *Imperative Macros* section of the *Sequential Access Method Macros* chapter).

Reading a Tape Backwards

When reading backwards (READ=BACK), a labeled tape must be positioned so that the first record read, when OPEN (or OPENR) is executed, is the tapemark physically following the trailer labels. An unlabeled file must be positioned so that the first record read, when OPEN (or OPENR) is executed, is the tapemark physically following the first logical data record to be read (the last record written when the file was created). Although ASCII unlabeled tapes contain no leading tapemark, special error recovery procedures allow these tapes to be read backwards.

Label checking of standard and nonstandard labels is similar. That is, IOCS still processes standard labels, and your 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 backwards reading is confined to one volume, an end-of-file condition always exists when the header label is encountered. At end-of-file for standard lables, IOCS checks only the block count (which was stored from the trailer label) and then branches to your EOFADDR routine. At EOF for nonstandard labels, IOCS branches to your LABADDR routine where the header label may be checked. To check labels, you must evoke physical IOCS macros to read the label(s). Your LABADDR routine, specified in the DTF, cannot issue a macro that calls a transient routine. For example, OPEN (or OPENR), CLOSE (or CLOSER), DUMP, CANCEL, and CHKPT cannot be issued. Also, when processing multivolume files, your 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 ckecking is the same as that for standard labels.

Checking Standard Labels on Tape

When standard labels are to be checked (DTFMT or DTFSR FILABL=STD or DTFPH TYPFLE=INPUT), you must supply the TLBL job control statement for standard label information. Also, when processing standard labels, a LBLTYP job control statement is required to define virtual storage needed at link-edit time for label processing (more information will be found in *DOS/VS System Control Statements*, GC33-5376).

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 are present for EBCDIC files, or UHLa or UTLa for ASCII files), IOCS enters the LABADDR routine and enters an 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.

Your routine can test this character to determine what labels should be checked. IOCS also loads the address if an 80-byte IOCS label area in register 1; this is the address you use if checking labels, or from which you move the label to your program's label area if you are modifying labels.

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 you wish to end label checking before all labels are read, you can issue a LBRET 1 macro. After all trailer labels are checked, IOCS initiates EOV or EOF procedures (see *Tape Input Files*, above).

Checking Nonstandard Labels on Tape

Any tape labels not conforming to the standard label specifications are considered nonstandard. It is your responsibility to check such labels if they are present. The MTC job control statement 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. The following possible conditions can thus 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. 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, you 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 (in the low-order bytes of register 1) the hexadecimal representation of the symbolic unit currently used. These values are as shown in bytes 6 and 7 of the CCB. IOCS also loads an alphabetic O into the low-order byte of register 0 when the file is opened.

When your routine gains control, the tape is not moved by OPEN (or OPENR). Physical IOCS macros must be used to transfer labels from tape to virtual storage. Therefore, you must establish a CCB and a CCW. The macro EXCP is used to initiate the transfer. After all labels are checked, you return control to OPEN (or OPENR) by use of the LBRET 2 macro.

When IOCS reads a tapemark, it checks to determine if you have supplied a LABADDR routine. 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, you must use physical IOCS macros to read your label(s). Furthermore, you must determine the EOF and/or EOV condition and indicate to IOCS which condition exists by loading either EF (end-of-file) or EV (end-of-volume) into 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.

Unlabeled Input Files on Tape

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 MTC job control statement. When the tapemark following the last data record is read, IOCS branches to the end-of-file address.

PART 2

SEQUENTIAL ACCESS METHOD

Declarative Macros

DTFxx Macro	**Associated Macros**	**Device Type**
DTFCD	CDMOD	Card
DTFCN	-	Console
DTFDI	DIMOD	Device Independent
DTFDR	DRMOD DFR DLINT	3886 Optical Character Reader
DTFDU	DUMODFx	Diskette
DTFMR	MRMOD	Magnetic Reader
DTFMT	MTMOD	Magnetic Tape
DTFOR	ORMOD	Optical Reader
DTFPR	PRMOD	Printer
DTFPT	PTMOD	Paper Tape
DTFSD	SDMODxx	Sequential DASD
DTFSR	-	Sequential Device

Imperative Macros

CHECK	ERET	OPENR	RDLINE
CHNG	FEOV	POINTR	READ
CLOSE	FEOVR	POINTS	RELSE
CLOSER	GET	POINTW	RESCN
CNTRL	LBRET	PRTOV	SETDEV
DISEN	NOTE	PUT	TRUNC
DSPLY	OPEN	PUTR	WAITF
			WRITE

Figure 2-1 summarizes the declarative and imperative macros which may be used for SAM processing on a given I/O device.

Figure 2-1 I/O macros for SAM processing

	DECLARATIVE MACROS												IMPERATIVE MACROS																												
													INITIALIZATION	PROCESSING																										COMPLETION	
	DTFCD	DTFCN	DTFDI[1]	DTFDU	DTFMR	DTFMT	DTFOR	DTFPR	DTFPT	DTFSD	DTFSR[2]	DTFDR	OPEN/OPENR	LBRET[3]	CHECK	CHNG[2]	CNTRL	DISEN	DSPLY	ERET	GET	LITE	NOTE	POINTR	POINTS	POINTW	PRTOV	PUT	PUTR	RDLNE	READ	RELSE	RESCN	TRUNC	WAIT F	WRITE	SETDEV	FEOV	FEOVD	LBRET[3]	CLOSE/CLOSER
Operator Console		X											X								X							X	X												
1287/1288 Optical Reader							X				X		X[4]				X		X[11]		X[10]									X[10]	X[11]		X[11]		X[11]						X[4]
1403/1443/3203/ 3211/5203 Printer			X					X			X		X[4]				X										X[15]	X													X[4]
1255/1259/ 1419 Magnetic Character Reader					X								X[4]		X			X			X	X[14]									X				X						X[4]
1442/2501/2520/2540/ 2596/3504/3505/3525[19]	X		X[23]								X[21]		X[4]				X[15]				X[13]							X[22]													X[4]
1442/2520/2540/3525/ Punch	X		X					X[20]			X[21]		X[4]				X[15]										X[20]	X[12]													X[4]
2560 MFCM/ 5425 MFCU	X		X					X					X[4]				X[15]	X			X							X													X[4]
2311 Disk Unit			X						X	X	X		X[4]	X	X[6]		X[15]			X[16]	X		X[6]	X[6]	X[6]			X[5]			X[6]	X[7]		X[8]		X[6]			X	X	X[4]
2314/2319/3330/3333/ 3340 Disk Unit			X							X			X[4]	X	X[6]		X[15]			X[16]	X		X[6]	X[6]	X[6]	X		X[5]			X[6]	X[7]		X[8]		X[6]			X	X	X[4]
2321 Data Cell										X			X[4]	X			X[15]			X[16]	X							X[5]											X	X	X[4]
2400-Series/3420 Magnetic Tape Unit			X			X					X		X[4]	X	X[6]	X[9]	X[15]			X[16]	X		X[6]	X[6]	X[6]	X[6]		X			X[6]	X[7]		X[8]		X[6]		X		X	X[4]
2671/1017 Paper Tape Reader									X		X[18]		X[2]								X																				X[2]
1018 Paper Tape Punch									X				.X[17]															X													X[17]
1270/1275 Optical Reader/ Sorter					X								X[4]		X			X			X	X[14]									X				X						X[4]
3881 Optical Mark Reader	X												X				X				X																				X
3886 Optical Reader												X	X				X														X				X		X				X
3540 Diskette I/0 Unit			X	X									X[4]							X[16]	X							X													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 simultanous-read-while-write tape control units are installed.
10. Journal tape processing only.
11. 1287/1288 document processing only.
12. PUT punches on input card with additional information if TYPEFLE=CMBND is specified for the 1442, 2520, or 2540, or if FUNC=RP or RPW is specified for the 3525. PUT prints on the card for the 3525 with the print feature.
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 specified.
17. Required if two I/O areas.
18. Valid for 2671 only.
19. 3525 Card Punch with read feature.
20. 3525 Card Punch with print feature.
21. Not supported for 2501, 3505, or 3525, respectively.
22. Not supported for 2501 or 3505. PUT is supported for any device that has a punch.
23. Not supported by 2596.

DECLARATIVE MACROS

As stated earlier, there are two related types of declarative macros: DTFxx macros and logic module generation macros. In this section each type of processing is divided by type of storage medium: card, magnetic tape, DASD, etc. The DTFxx macro used with the file is discussed first, and then (where applicable) the corresponding logic module generation macro.

As discussed earlier, you need not specify names for your modules. IOCS will do this for you, making use of subsetting/supersetting wherever it is possible (see *Module Names* in *The Macro System* chapter).

The sections on module-naming conventions following the discussions of the logic module generation macros are therefore provided only for those who are interested in seeing how IOCS forms the names for the modules.

DTFCD Macro

This macro defines a file for a card reader. However, it should not be used to read SYSIPT data if the program might be invoked by a catalogued procedure. In this case, the DTFDI macro should be used.

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 2-3 lists the keyword operands contained in the operand field.

ASOCFLE=filename
This operand is used together with the FUNC operand to define associated files for the 2560, 3525, or 5425. (For a description of associated files see the *DOS/VS Data Management Guide*, GC33-5372.) ASOCFLE specifies the filename of associated read, punch, or print files, and enables macro sequence checking by the logic module of each associated file. One filename is required per DTF for associated files.

Figure 2-2 defines the filename specified by the ASOCFLE operand for each of the associated DTFs.

	In ASOCFILE operand of ...		
FUNC=	read DTFCD, specify filename of	punch DFTCD, specify filename of	print DTFPR, specify filename of
RP	punch DTFCD	read DTFCD	
RW	print DTFPR		read DTFCD
PW		print DTFPR	punch DTFCD
RPW	punch DTFCD	print DTFPR	read DTFCD

Figure 2-2 ASOCFLE operand usage

For example, if FUNC=PW is specified, specify the filename of the print DTFPR in the ASOCFLE operand of the punch DTFCD, and specify the filename of the punch DTFCD in the print DTFPR. Or if FUNC=RPW is specified, specify the filename of the punch DTFCD in the ASOCFLE operand of the read DTFCD; specify the filename of the print DTFPR in the punch DTFCD; and specify the filename of the read DTFCD in the print DTFPR.

P-35

Applies to: Input	Output	Combined			
x	x	x	M	DEVADDR = SYSxxx	Symbolic unit for reader-punch used for this file
x	x	x	M	IOAREA1 = xxxxxxxx	Name of first I/O area, or seperate input area if TYPEFLE=CMBND and IOAREA2 are specified.
x	x		O	ASOCFLE = xxxxxxx	Name for FUNC=RP, RW, RPW, PW
x	x	x	O	BLKSIZE = nnn	Length of one I/O area, in bytes. If omitted, 160 is assumed for a column binary on the 2560,3504,3505, or 3525; 96 is assumed for the 2596 or 5425, otherwise 80 is assumed.
x	x	x	O	CONTROL = YES	CNTRL macro used for this file. Omit CTLCHR for this file. Does not apply to 2501.
	x		O	CRDERR = RETRY	Retry if punching error is detected. Applies to 2520 and 2540 only.
	x		O	CTLCHR = xxx	(YES or ASA). Data records have control character. YES for S/370 character set; ASA for American National Standards Institute character set. Omit if TYPEFLE=CMBND. Omit CONTROL for this file.
x	x	x	O	DEVICE = nnnn	(1442, 2501, 2520, 2540, 2560P, 2560S, 2596, 3504, 3505, 3525, 5425P, or 5425S). If omitted, 2540 is assumed.
x		x	O	EOFADDR = xxxxxxxx	Name of your end-of-file routine.
x	x		O	ERROPT = xxxxxx	IGNORE, SKIP, or name. Applies to 2560, 3504, 3505, 3525 and 5425 only.
x	x		O	FUNC = xxx	R, P, W, I, RP, RW, RPW, PW. Applies to 2560, 3525, and 5425 only.
x	x	x	O	IOAREA2 = xxxxxxx	Name of second I/O area, or separate output area if TYPEFLE=CMBND. Not allowed if FUNC=RP, RW, RPW, or PW. Not allowed for output file if ERROPT=IGNORE.
x	x		O	IOREG = (nn)	Register number, if two I/O areas used and GET or PUT does not specify a work area. Omit WORKA.
x	x		O	MODE = xx	(E or C) for 2560. (E, C, O, R, EO, ER, CO, CR) for 3504 and 3505. (E, C, R, ER, CR) for 3535. If omitted, E is assumed.

M=Mandatory; O=Optional

Figure 2-3 **DTFCD macro (part 1 of 2)**

Applies to

Input	Output	Combined			
x	x	x	O	MODNAME = xxxxxxxx	Name of CDMOD logic module for this DTF. If omitted, IOCS generates standard name.
		x	O	OUBLKSZ = nn	Length of IOAREA2 if TYPEFLE=CMBND. If OUBLKSZ omitted, length specified by BLKSZ is assumed for IOAREA2.
x	x	x	O	RDONLY = YES	Generates a read-only module. Requires a module save area for each task using the module.
x	x	x	O	RECFORM = xxxxxx	(FIXUNB, UNDEF, or VARUNB). If omitted, FIXUNB is assumed. Input or combined files always FIXUNB.
	x		O	RECSIZE = (nn)	Register number if RECFORM=UNDEF. General registers 2-12, written in parentheses.
x	x	x	O	SEPASMB = YES	DTFCD is to be assembled separately.
x	x		O	SSELECT = n	(1 or 2) for 1442, 2520, 2596, 3504, or 3525. (1, 2, or 3) for 2540. (1, 2, 3, 4, or 5) for 2560. (1, 2, 3, or 4) for 5425. Stacker-select character.
x	x	x	O	TYPEFLE = xxxxxx	(INPUT, OUTPUT, or CMBND) If omitted INPUT assumed. CMBND may be specified for 1442N1, 2520B1, or 2540 punch-feed-read only.
x	x	x	O	WORKA = YES	GET or PUT specifies work area. Omit IOREG. Not allowed for output file if ERROPT=IGNORE.

M=Mandatory; O=Optional

Figure 2-3 DTFCD macro (part 2 of 2)

BLKSIZE=n
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 the operand FUNC=I is specified for the 2560 or 3525, the length specified for BLKSIZE must be 80 data bytes if CTLCHR=YES or ASA is not specified, or 81 if CTLCHR=YES or ASA is specified.

For the 3881, the BLKSIZE operand must be sufficient to contain:

- 6 bytes of record description information
- Mark read data
- Binary coded decimal (BCD) mark read data if the BCD feature is being used.
- 7 bytes of serial number and batch number data if the serial number feature is being used.

The BLKSIZE operand for the 3881 cannot exceed 900. If a BLKSIZE greater than 900 is specified, the BLKSIZE defaults to 900.

If the BLKSIZE operand is omitted, the length is assumed to be 80, with the following exceptions:

- 160 is assumed for column binary mode on the 2560, 3505, or 3525.
- 96 is assumed for the 2596 or 5425.
- 900 is assumed for the 3881.

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 for an input file with two I/O areas (when the IOAREA2 operand is specified).

This operand must not be specified for an input file used in association with a punch file (when the operand FUNC=RP or RPW is specified) on the 2560, 3525, or 5425; in this case, however, this operand can be specified in the DTFCD for the associated punch file.

CRDERR=RETRY
This operand applies to card output on the 2520 or 2540. It specifies the operation to be performed if an error is detected. From this specification, IOCS generates a retry routine and a save area for the card punch record.

If a punching error occurs, it is usually ignored and operation continues. The error card is stacked in stacker P1 (punch), while correct cards are stacked in the stacker you select. 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.

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 System/370 character set. *Appendix A* 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 ASSGN job control statement to assign the actual I/O device address to the file.

SYSIPT, SYSPCH, or SYSRDR must not be specified:

- for the 2596
- for the 3881
- for 1442, 2520, or 2540 combined files (TYPEFLE=CMBND)
- for 2560, 3525, or 5425 associated files (FUNC=RP, RW, RPW, or PW)
- if the operand FUNC=I is specified
- if the MODE operand is specified with the C, O, or R parameters.

DEVICE={2540 | 1442 | 2501 | 2520 | 2560P | 2560S | 2596 | 3504 | 3505 | 3525 | 5425P | 5425S | 3881}
This operand specifies the I/O device associated with a file. The "P" and "S" included with the "2560" and "5425" parameters specify primary or secondary input hoppers.

EOFADDR=name
This entry must be included for input and combined files and specifies the symbolic name of your end-of-file routine. IOCS automatically branches to this routine on an end-of-file condition. In your routine you can perform any operations required for the end

of the file (you generally issue a CLOSE instruction for the file).

IOCS detects end-of-file conditions in the card reader by recognizing the characters /* punched in card columns 1 and 2. If the system logical units SYSIPT and SYSRDR are assigned to a 5425, IOCS requires that the /* card, indicating end-of-file, be followed by a blank card. An error condition results if cards are allowed to run out without a /* trailer card (and without a / & card if end-of-job).

ERROPT={IGNORE | SKIP | name}
This operand specifies the error exit option used for an input or output file on a 2560, 3504, 3505, 3525, or 5425. Either IGNORE, SKIP, or the symbolic name of an error routine can be specified for input files. Only IGNORE can be specified for output files. This operand must be omitted when using 2560 or 5425 associated output files. The functions of these parameters are described below.

IGNORE indicates that the error is to be ignored. The address of the record in error is put in register 1 and made available for processing. For output files, byte 3, bit 3 of the CCB is also set on (see Figure 6-1); you can check this bit and take the appropriate action to recover from the error. Only one I/O area and no work area is permitted for output files. When IGNORE is specified for an input file associated with a punch file (FUNC=RP or RPW) and an error occurs, a PUT for the card in error must nevertheless be given for the punch file.

SKIP indicates that the record in error is not to be made available for processing. The next card is read and processing continues.

If name is specified, IOCS branches to your routine when an error occurs, where you may perform whatever actions you desire. Register 1 contains the address of the record in error, and register 14 contains the return address. GET macros must not be issued in the error routine for cards in the same device (or in the same card path for the 2560 or 5425). If the file is an associated file, PUT macros must not be issued in the error routine for cards in the same device (for the 2560 or 5425 this applies to cards in either card path). If any other IOCS macros are issued in the routine, register 14 must be saved. If the operand RDONLY=YES is specified, register 13 must also be saved. At the end of your routine return to IOCS by branching to the address in register 14. If the input file is associated with an output file (FUNC=RP, RPW,or RW), no punching or printing must be done for the card in error. IOCS continues processing by reading the next card.

Note: When ERROPT is specified for an input file and an error occurs, there is a danger that the /* end-of-file card may be lost. This is because IOCS, after taking the action for the card in error specified by the ERROPT operand, returns to normal processing by reading the next card which is assumed to be a data card. If this card is in fact an end-of-file card, the end-of-file condition cannot be recognized.

FUNC={R | P | I | RP | RW | RPW | PW}
This operand specifies the type of file to be processed by the 2560, 3525, or 5425. R indicates read, P indicates punch, and W indicates print.

When FUNC=I is specified, the file will be both punched and interpreted; no associated file is necessary to achieve this. The information printed will be the same as the information punched, in contrast to FUNC=PW, where any relation between the information printed and the information punched is determined by your program. When FUNC=I is specified the file can have only one I/O area.

RP, RW, RPW, and PW are used, together with the ASOCFLE operand, to specify associated files; when one of these parameters is specified for one file, it must also be specified for the associated file(s). Associated files can each have only one I/O area.

IOAREA1=name
This operand specifies the name of the input or output area used for this file.

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 the name of a second I/O area. If the file is a combined file and the operand is specified, the designated area is an output area.

If this operand is specified for the 3881, the IOREG operand must also be specified.

This operand must not be specified if FUNC=I, FUNC=RP, RPW, RW, or PW, or for output files if ERROPT=IGNORE.

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 record. For output files, IOCS puts the address where the user can build a record. This operand cannot be used for combined files.

This operand must be specified for the 3881 if the IOAREA2 operand is specified.

MODE={E | C | O | R | EO | ER | CO | CR}
This operand specifies the mode used to process an input or output file for a 2560, 3504, 3505, or 3525. E indicates normal EBCDIC mode; C indicates column binary mode; O indicates optical mark read (OMR) mode; R indicates read column eliminate mode. E is also assumed if only O or R is specified.

For the 2560, only E and C are valid entries.

Valid entries for the 3504 and 3505 are E, C, O, R, EO, ER, CO, and CR. Valid entries for the 3525 are E, C, R, ER, and CR. If O or R is specified (with or without E or C), a format descriptor card defining the card columns to be read, or eliminated, must be provided. See OMR considerations in the *DOS/VS Data Management Guide*, GC33-5372, for instructions on how to write this card as well as on how to code and process OMR data.

Only E is valid for SYSIPT, SYSPCH, or SYSRDR. O and R (with or without E or C) cannot be specified for output files. E is assumed if the MODE operand is omitted.

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, MODNAME must specify the same name as the CDMOD macro.

If this operand is omitted, standard names are generated for calling the logic module. If two DTF macros 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, but only for a combined file. Enter the maximum number 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 task should have its own uniquely defined save area. Each time an imperative macro (except OPEN or OPENR) is issued, register 13 must contain the address of the save area associated with that task. 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* in the *Multitasking Macros* chapter.

If an ERROPT routine issues I/O macros using the same read-only module that caused control to pass to the error routine, your 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 routine. Before returning to the module that entered the ERROPT routine, register 13 must contain the save area address originally specified for the task.

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: fixed length, variable length, or undefined. If the record format is FIXUNB, this operand may be omitted. If TYPEFLE=INPUT, TYPEFLE=CMBND, FUNC=I, or DEVICE=3881, 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. You must load the length of each record into the specified register before you issue the PUT macro for the record.

SEPASMB=YES
Include this operand only if the DTFCD 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 problem program and no CATALR card is punched.

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) stackers. For the 5425 cards are placed in stacker 1 when the cards came from hopper 1 and into stacker 5(4) when they came from hopper 2. This operand must not be specified for combined files or for the 3881. This operand must not be specified for 2560, 3525, or 5425 read files associated with punch files (FUNC=RP or RPW); in this case the SSELECT=n operand may be specified for the associated output file. See the *CNTRL Macro* in the *Processing Macros* section later in this chapter for further information.

Note: When this operand is used with a device other than a 1442 or 2596, 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 a 1442 or 2520 or for a 2540 with the punch-feed-read feature. TYPEFLE=CMBND is applicable if both GETs and PUTs are issued for the same card file.

Only TYPEFLE=INPUT can be specified for the 3881. If TYPEFLE=OUTPUT or TYPEFLE=CMBND is specified, the DTF defaults to DEVICE=2540 and a non-executable CDMOD logic module is produced. The MNOTE "Improper device. 2540 assumed." is then printed at assembly time. If TYPEFLE=INPUT is omitted, INPUT is assumed.

WORKA=YES
If I/O records are processed in work areas instead of in the I/O areas, specify this operand. You must set up the work area in storage. The address of the work area, or a general-purpose register which contains the address, must be specified in each GET and PUT macro.

If ERROPT=IGNORE is specified for an output file or if DEVICE=3881, WORKA=YES must not be specified.

CDMOD Macro

Listed here are the operands you can specify 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 is used with the module and its associated DTFs. The module also processes files for which the CNTRL macro is not used.

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

This operand must not be specified for an input file used in association with a punch file (when the operand FUNC=RP or RPW is specified) on the 2560, 3525, or 5425; in this case, however, this operand can be specified in the DTFCD and CDMOD for the associated punch 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 same 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. Any DTF to be used with this module must have the same operand. If CTLCHR is included, CONTROL must not be specified. This operand does not apply to a combined file or to an input file.

DEVICE={2540 | 1442 | 2501 | 2520 | 2560P | 2560S | 2596 | 3504 | 3505 | 3525 | 5425P | 5425S | 3881}
Include this operand to specify the I/O device used by the module. The "P" and "S" included with the "2560" and "5425" parameters specify primary or secondary input hoppers; regardless of which is specified, however, the module generated will handle DTFs specifying either hopper.

Any DTF to be used with this module must have the same operand (except as just noted concerning the "P" and "S" specification for the 2560 or 5425).

FUNC={R | P | I | RP | RW | RPW | PW}
This operand specifies the type of file to be processed by the 2560, 3525, or 5425. Any DTF used with the module must have the same operand. R indicates read, P indicates punch, and W indicates print.

When FUNC=I is specified, the file will be both punched and interpreted; no associated file is necessary to achieve this.

RP, RW, RPW, and PW specify associated files; when one of these parameters is specified for one file, it must also be specified for the associated file(s). Associated files can have only one I/O area each.

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. This operand is not valid for associated files.

RDONLY=YES
This operand causes a read-only module to be generated. Whenever this operand is specified, any DTF used with the module must have the same operand.

RECFORM={<u>FIXUNB</u> | VARUNB | UNDEF}
This operand specifies the record format: fixed-length, variable-length, or undefined. Any DTF used with the module must have the same operand. If TYPEFLE=INPUT, TYPEFLE=CMBND, or FUNC=I, this operand must be FIXUNB. For the 3881, only RECFORM=FIXUNB is valid. If this operand is omitted for the 3881, RECFORM=FIXUNB is assumed.

SEPASMB=YES
Include this operand only if the module is assembled separately. This causes a CATALR card with the module name (standard or user-specified) to be punched ahead of the object deck and defines the module name as an ENTRY point in the assembly. If the operand is omitted, the program assumes that the DTF is being assembled with the problem program and no CATALR card is punched.

TYPEFLE={<u>INPUT</u> | OUTPUT | CMBND}
This operand generates a module for either an input, output, or combined file. Any DTF used with the module must have the same operand. For the 3881, only TYPEFLE=INPUT is valid. If TYPEFLE=INPUT is omitted, INPUT is assumed.

WORKA=YES
This operand must be included if records are to be processed in work areas instead of in I/O areas. Any DTF used with the module must have the same operand. This operand is not valid for the 3881.

Standard CDMOD Names
Each name begins with a 3-character prefix (IJC) and continues with 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, CMBND, or FUNC=I 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, 3881
	=	1	DEVICE=1442, 2596
	=	2	DEVICE=2520
	=	3	DEVICE=2501
	=	4	DEVICE=2540 and CRDER
	=	5	DEVICE=2520 and CRDERR
	=	6	DEVICE=3505 or 3504
	=	7	DEVICE=3525 and FUNC=R/P or omitted
	=	8	DEVICE=2560 and FUNC=R/P or omitted
	=	9	DEVICE=5425 and FUNC=R/P or omitted
	=	A	DEVICE=3525 and FUNC=RP
	=	B	DEVICE=3525 and FUNC=RW
	=	C	DEVICE=3525 and FUNC=PW
	=	D	DEVICE=3525 and FUNC=I
	=	E	DEVICE=3525 and FUNC=RPW
	=	F	DEVICE=2560 and FUNC=RP
	=	G	DEVICE=2560 and FUNC=RW
	=	H	DEVICE=2560 and FUNC=PW
	=	I	DEVICE=2560 and FUNC=I
	=	J	DEVICE=2560 and FUNC=RPW
	=	K	DEVICE=5425 and FUNC=RP
	=	L	DEVICE=5425 and FUNC=RW
	=	M	DEVICE=5425 and FUNC=PW
	=	N	DEVICE=5425 and FUNC=I
	=	O	DEVICE=5425 and FUNC=RPW

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 *IOCS Subset/Superset Names* in *The Macro System* chapter.

```
      * * * * *
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
              6
              7
              8
              9
              A
              B
              C
              .
              .
              .
              M
              N
              O
```

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

DTFCN Macro

DTFCN defines an input or output file that is processed on a 3210 or 3215 console printer-keyboard, or a display operator console. DTFCN provides GET/PUT logic as well as PUTR logic for a file. Enter the symbolic name of the file in the name field and DTFCN in the operation field. The detail entries follow the DTFCN header card in any order. Figure 2-4 lists the keyword operands contained in the operand field.

BLKSIZE=n
This operand specifies the length of the I/O area; if the PUTR macro is used (TYPEFLE=CMBND is specified), this operand specifies the length of the output part of the I/O area. For the undefined record format, BLKSIZE must be as large as the largest record to be processed. The length must not exceed 256 characters.

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

DEVADDR={SYSLOG | SYSnnn}
This operand specifies the symbolic unit associated with the file. In a multiprogramming environment, DEVADDR=SYSLOG must be specified to obtain Background (BG), Foreground 1 (F1), Foreground 2 (F2), Foreground 3(F3), or Foreground 4(F4) prefixes for message identification. DEVADDR=SYSLOG must be specified if TYPEFLE=CMBND is specified.

INPSIZE=n
This operand specifies the length of the input part of the I/O area for PUTR macro usage.

IOAREA1=name
This operand specifies the name of the I/O area used by the file. For PUTR macro usage, the first part of the I/O area is used for output, and the second part is used for input. The lengths of these parts are specified by the BLKSIZE and INPSIZE operands respectively. 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.

M	DEVADDR = SYSxxx	Symbolic unit for the console used for this file.
M	IOAREA1 = xxxxxxxx	Name of I/O area.
O	BLKSIZE = nnn	Length in bytes of I/O area (for PUTR macro usage, length of output part of I/O area). If RECFORM=UNDEF, max. is 256. If omitted, 80 is assumed.
O	INPSIZE = nnn	Length in bytes for input part of I/O area for PUTR macro usage.
O	MODNAME = xxxxxxx	Logic module name for this DTF. If omitted, IOCS generates a standard name. The logic module is generated as part of the DTF.
O	RECFORM = xxxxxx	(FIXUNB or UNDEF). If omitted, FIXUNB is assumed.
O	RECSIZE = (nn)	Register number if RECFORM=UNDEF. General registers 2-12, written in parentheses.
O	TYPEFLE = xxxxxx	(INPUT, OUTPUT, or CMBND). INPUT processes both input and output. CMBND must be specified for PUTR macro usage. If omitted, INPUT is assumed.
O	WORKA = YES	GET or PUT specifies work area.

M = Mandatory; O = Optional

Figure 2-4 **DTFCN macro**

MODNAME=name
This operand specifies 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 link-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: fixed length or undefined. FIXUNB must be specified if TYPEFLE=CMBND is specified. FIXUNB is assumed if the RECFORM operand is omitted.

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, you must load the length of each record into the designated register before you issue a PUT macro. If specified for input files, IOCS provides the length of the record transferred to storage.

TYPEFLE={INPUT | OUTPUT | CMBND}
This operand specifies a file as input, output, or combined. If INPUT is specified, code is generated for both input and output files. If OUTPUT is specified, code is provided for output files only.

CMBND must be specified if you use the PUTR macro. CMBND specifies that coding be generated for both input and output files; in addition, coding is generated to allow usage of the PUTR macro to ensure that messages requiring operator action are not deleted from the console. When CMBND is specified, DEVADDR=SYSLOG must also be specified.

WORKA=YES
This operand indicates that a work area is used with the file. A GET or PUT macro moves the record to or from the work area. A PUTR macro moves the record from and to the work area.

DTFDI Macro

The DTFDI macro provides device independence for system logical units. If several DTFDI macros 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 storage than multiple LIOCS macros. It allows you to change device assignments without reassembling the logic module.

The DTFDI macro should always be used to read SYSIPT data if the program migth be invoked by a catalogued procedure.

The restrictions on DTFDI processing are:

- Only fixed unblocked records are supported.
- Only forward reading is allowed.
- In a multivolume diskette file, new volumes are fed automatically.
- The last volume of a multivolume diskette *output* file will be ejected automatically, but the last volume of a multivolume diskette *input* file will not.
- If DTFDI is used with diskettes, special records (deleted or sequentially relocated records) on input files are skipped and not passed to the user.
- Rewind options are not provided.
- Combined file processing is not supported for reader-punches.
- Reading of cards is restricted to the first 80 bytes per card.
- 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 uses the same module with RDONLY=YES, overprinting may occur.
- If DTFDI is used with DASD or diskettes, FOPT SYSFIL must have been 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 2-5.

DEVADDR={SYSIPT | SYSLST | SYSPCH | SYSRDR}
This operand must specify the symbolic unit associated with this system file. Only the system names shown above may be specified. The logical device SYSLST must not be assigned to the 2560 or 5425.

EOFADDR=name
This operand must specify the name of your 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, you can perform any operations necessary for the end-of-file condition (you generally issue the CLOSE or CLOSER macro).

IOCS detects the end-of-file condition by recognizing the characters /* in positions 1 and 2 of the record for cards, a tapemark for tape, and a filemark for disk. If the system logical units SYSIPT and SYSRDR are assigned to a 5425, IOCS requires that the /* card, indicating end-of-file, be followed by a blank card. An error condition results if the records are allowed to run out without a /* card (and without a / & card, if end-of-job). IOCS detects the end-of-file condition on diskette units by recognizing that end-of-data has been reached on the current volume and that there are no more volumes available.

M	DEVADDR = SYSxxx	(SYSIPT, SYSLST, SYSPCH, or SYSRDR). System logical unit.
M	IOAREA1 = xxxxxxxx	Name of first I/O area.
O	EOFADDR = xxxxxxxx	Name of your end-of-file routine.
O	ERROPT = xxxxxxxx	(IGNORE, SKIP, or name of your error routine). Prevents termination on errors.
O	IOAREA2 = xxxxxxxx	If two I/O areas are used, name of second area.
O	IOREG = (nn)	Register number. If omitted and 2 I/O areas are used, register 2 is assumed. General registers 2-12, written in parentheses.
O	MODNAME = xxxxxxxx	DIMOD name for this DTF. If omitted, IOCS generates a standard name.
O	RDONLY = YES	Generates a read-only module. Requires a module save area for each task using the module.
O	RECSIZE = nnn	No. of chars. in record. Assumed values: 121(SYSLST), 81(SYSPCH), 80(otherwise).
O	SEPASMB = YES	DTFDI to be assembled separately.
O	WLRERR = xxxxxxxx	Name of your wrong length record routine.

M=Mandatory; O=Optional

Figure 2-5 **DTFDI macro**

ERROPT={IGNORE | SKIP | name}
This operand does not apply to output files. For output files for most devices, the job is automatically terminated after IOCS has attempted to retry writing the record; for 2560 or 5425 output files, normal error recovery procedures are followed. This operand 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 to be performed for an error block. If an error is detected when reading a magnetic tape, a disk pack, or a diskette volume, 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 three specifications are described below.

IGNORE indicates that the error condition is to be ignored. The address of the error record is made available to you for processing (see *CCB Macro* in the chapter *Physical IOCS*).

SKIP indicates that the error block is not to be made available for processing. The next record is read and processing continues.

name indicates that IOCS is to branch to your routine when an error occurs, where you may perform whatever functions desired 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, you must not issue any GET instructions in your error routine. If you use any other IOCS macros, you must save the contents of register

14. If RDONLY=YES is specified, you must also save the contents of register 13. At the end of the error routine, return to IOCS by branching to the address in register 14. The next record is then made available for processing.

IOAREA1=name
This operand must specify the 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 a 2540, 2560, 3525, or 5425, the first byte of the 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 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 you 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.

If this entry is omitted, standard names are generated for calling the logic module. If two different DTF macros 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 task should have its own uniquely defined save area, and each time an imperative macro (except OPEN, OPENR or LBRET) is issued, register 13 must contain the address of the save area associated with that task. 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* in the *Multitasking Macros* chapter.

If an ERROPT or WLRERR routine issues I/O macros using the same read-only module that caused control to pass to either error routine, the 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 error routine, register 13 must be set to the save area address originally specified for the task.

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/370 control character if the character is not a valid ASA character. The program checks ASA control characters before System/370 control characters. Therefore, if it is a valid ASA control character (even though it may also be a System/370 control character), it is used as an ASA control character. Otherwise, it is used as a System/370 control character.

Control character codes are listed in *Appendix A*, except for the following:

- 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 is assumed:

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

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

SEPASMB=YES
Include this operand only if the DTFDI 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 problem program and no CATALR card is punched.

WLRERR=name
This entry applies only to input files on devices other than diskette units. It specifies the name of your 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 that specified in the RECSIZE operand. If the length of the record is less than that specified in the RECSIZE operand, 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 length of the record to be read is larger than that specified in the RECSIZE operand, 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 value of the RECSIZE operand, and the remainder of the record is truncated.

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

At the end of the routine, you must return to IOCS by branching to the address in register 14. When control returns to your 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 you. If, in addition to a wrong-length record error, an irrecoverable parity error occurs, the job is terminated.

DIMOD Macro

Listed here are the operands you can specify for DIMOD. The header card contains DIMOD in the operation field and may contain a module name in the name field. If the module name is omitted, IOCS generates a standard module name.

IOAREA2=YES
Include this operand if a second I/O area is needed. A module with this operand can be used with DTFDIs specifying either 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. Whenever this operand is specified, any DTF used with the module must have the same operand.

SEPASMB=YES
Include this operand only if the module is assembled separately. This causes a CATALR card with the module name (standard or user-specified) to be punched ahead of the object deck and defines the module name as an ENTRY point in the assembly. If the operand is omitted, the program assumes that the DTF is being assembled with the problem program and no CATALR card is punched.

TYPEFLE={OUTPUT | INPUT}
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.

Standard DIMOD Names
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	
b	= C	
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 *IOCS Subset/Superset Names* in *The Macro System* chapter.

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

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

DTFDR Macro

You must use the DTFDR macro to define each 3886 file in your program. This macro defines the characteristics of the file, the format record to be loaded into the 3886 when the file is opened, and the storage areas and routines used. Enter the symbolic name of the file in the name field and DTFDR in the operation field.
The entries of the DTFDR Macro are discussed here and illustrated in Figure 2-6.

Besides the DTFDR Macro, the following declarative macros are required for a 3886 file:

DRMOD	Generate the logic module to process the file.
DFR	Define attributes common to a group of lines described in one format record.
DLINT	Describes the individual line in the format record.

DEVADDR=SYSnnn
Specifies the symbolic unit to be associated with the logical file. The symbolic unit (SYSnnn) is associated with an actual I/O device through the job control ASSGN statement.

FRNAME=phasename
Specifies the phase name of the format record to be loaded when the file is opened.

FRSIZE=number
Specifies the number of bytes to be reserved in the DTF expansion for format records. The number must equal at least the size of the largest DFR macro expansion and its associated DLINT macro expansions, plus four. This size is printed in the ninth and tenth bytes of the DFR macro expansion. For a description of these macro expansions, see *DOS/VS LIOCS Volume 2, SAM,* SY33-8560.

If you use the SETDEV macro in your program to change format records, you can reduce the library retrieval time by specifying a size large enough to contain all the frequently used format records. The area should then be equal to the sum of the format record sizes, plus four bytes for each format record. When the SETDEV macro is issued, the format record is loaded into this area from the core image library if it is not already present in the area.

EXITIND=name
Specifies the symbolic name of the one-byte area in which the completion code is returned to the COREXIT routine for error handling from an I/O operation.

M	DEVADDR = SYSxxx	Symbolic unit assigned to 3886 optical character reader.
M	FRNAME = xxxxxxxx	Phase name of format record to be loaded upon file opening.
M	FRSIZE = nn	Number of bytes to be reserved in DTF expansion for format records.
M	EXITIND = xxxxxxxx	Name of completion code return area.
M	IOAREA1 = xxxxxxxx	Name of file input area.
M	HEADER = xxxxxxxx	Name of area for header record from 3886.
M	EOFADDR = xxxxxxxx	Address of your end-of-file routine.
M	COREXIT = xxxxxxxx	Name of your error condition routine.
O	DEVICE = 3886	If omitted, 3886 is assumed.
O	RDONLY = YES	If DTF is to be used with read-only module.
O	MODNAME = xxxxxxx	Name of DRMODxx logic module for this DTF. If omitted, IOCS generates standard name.
O	BLKSIZE = nnn	Length of area named by IOREG1. If omitted, the maximum length of 130 is assumed.
O	SEPASMB =YES	If DTFDR is to be assembled separately.
O	SETDEV = YES	If SETDEV macro is issued in your program to load a different format record into the 3886.

M=Mandatory; O=Optional

Figure 2-6 DTFDR macro operands

The meanings of the completion codes are:

Code	**Meaning**
X'F0'	No errors occured. (This code should not be present when the COREXIT routine receives control.)
X'F1'	Line mark station timing mark check error.
X'F2'	Nonrecovery error (operator intervention is required).
X'F3'	Incomplete scan.
X'F4'	Line mark station timing mark check and equipment check.
X'F9'	Permanent error.

Note: If any of these errors occur while the file is being opened, the COREXIT routine does not receive control and the job is canceled.

IOAREA1=name

Specifies the symbolic name of the input area to be used for the file. The area must be as large as the size specified in the BLKSIZE parameter. If BLKSIZE is not specified, the input area must be 130 bytes.

HEADER=name

Specifies the symbolic name of the 20-byte area to receive the header record from the 3886.

EOFADDR=name

Specifies the symbolic address of your end-of-file routine. LIOCS branches to this routine whenever end of file is detected on the 3886.

COREXIT=name
Provides the symbolic name of your error correction routine. LIOCS branches to this routine whenever an error is indicated in the EXITIND byte.

You can attempt to recover from various errors that occur on the 3886 through the COREXIT routine you provide. Your COREXIT routine receives control whenever one of the following conditions occurs:

- Incomplete scan
- Line mark station timing mark check error
- Nonrecovery error
- Permanent error

Note: If any of these errors occur while the file is being opened, the COREXIT routine does not receive control and the job is canceled.

Figure 2-7 describes normal functions for the COREXIT routine for the various error conditions and provides the exits that must be taken from the COREXIT routine.

Error messages are provided to describe errors to the operator during program execution.

Error	Normal COREXIT Function	Exit to
X'F2'	Eliminate the data that has been read from this document and prepare to read the next input document (See Note 1).	Routine in your program to read the next document.
X'F4' or X'F9'	Do whatever processing is necessary before the job is canceled. (See Note 1).	Your end-of-job routine.
X'F1'	Do any processing that may be required. The document may have been read incorrectly; you may want to delete all data records from the document (see Note 2).	Branch to the address in register 14 to return to the instruction following the macro causing the error.
X'F3'	Rescan the line using another format record or using image processing and editing the record in your program (see Note 2).	Branch to the address in register 14 to return to the instruction following the macro causing the error.

Note 1: If in your COREXIT routine, you issue an I/O macro to the 3886 and an error occurs during that operation, control is returned to the beginning of the COREXIT routine. You must take precautions in the COREXIT routine to prevent looping in this situation. If no errors occur, control returns to the instruction following the I/O macro.

Note 2: If, in your COREXIT routine, you issue an I/O macro to the 3886, control always returns to the instruction following the macro. You should then check the completion code to determine the outcome of the operation.

Figure 2-7 **COREXIT routine functions**

DEVICE=3886
Indicates that 3886 is the I/O device for this file. If this parameter is omitted, 3886 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, OPENR, 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* in the *Multitasking Macros* chapter.

If a COREXIT routine issues I/O macros using the same read-only module that caused control to pass to either error routine, your 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 COREXIT routine. Before returning to the module that entered the COREXIT 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.

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 (DRMOD) is assembled with the program, the MODNAME parameter in this DTF must specify the same name as the DRMOD macro.

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

BLKSIZE=nnn
Specifies the length of the area named by the IOAREA1 keyword. The length of the area must be equal to the length of the longest record to be passed from the 3886.

If this operand is omitted, the maximum length of 130 is assumed.

Note: DOS/VS LIOCS does not allow you to block records read from the 3886.

SEPASMB=YES
Specifies the DTF is assembled separately. If this operand is specified, a CATALR card with the filename is punched before the deck and defines the filename as an entry point for the assembly.

SETDEV=YES
Specifies that the SETDEV macro is issued in your program to load a different format record into the 3886.

DRMOD Macro

Listed here are the operands you can specify for DRMOD. The first card contains DRMOD in the operation field and may contain a module name in the name field.

DEVICE=3886
Specifies that the 3886 is the input device. If this parameter is omitted, the 3886 is assumed.

SEPASMB=YES
Must be specified if the I/O module is assembled separately. This entry causes a CATALR card to be punched preceding the module.

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 DTFDR RDONLY operand.

SETDEV=YES
Is specified if the SETDEV macro may be used when processing a file with this I/O module. If SETDEV=YES is specified in the DRMOD macro but not in the DTFDR macro, the SETDEV macro cannot be used when processing that file.

Standard DRMOD Names
Each name consists of eight characters. They are: IJMZxxD0. The fifth and sixth characters are variables as follows:

- If SETDEV=YES is specified, the fifth character is S; otherwise it is Z.

- If RDONLY=YES is specified, the sixth character is R; otherwise it is Z.

Note: Subsetting/supersetting is allowed with the

SETDEV keyword, but not with the RDONLY keyword.

DFR Macro

Two macros are provided for defining documents. One, the DFR macro, defines attributes common to a group of line types. The other, the DLINT macro, defines specific attributes of an individual line type. As many as 26 DLINT macros can be associated with one DFR macro as long as the number of line types plus the number of fields is less than or equal to 53.

The DFR and associated DLINT macros are used in one assembly to build a format record module. Only one DFR with its associated DLINT macros may be specified in each assembly, and the DFR must precede all DLINT macros in the assembly. The format record must be link-edited into the core image library so that it can be loaded into the 3886 when the file is to be processed. The format record defines the types of lines to be read, the fields on the lines, the editing functions to be used, and the format of the data record to be passed to your program. For an example of how to build a format record using DFR and DLINT macros, see *Appendix B.1: Assembling a Format Record for the 3886 Optical Character Reader.*

The format record is loaded into the 3886 during program execution. The initial format record is loaded when the file is opened and new format records can be loaded using the SETDEV macro.

The DFR macro defines attributes common to a group of line types described by one format record. DLINT macros describe the individual lines. The DFR macro is specified first and provides the following information for the format record:

- Default font
- Reject character
- Group and character erase symbol usage
- National symbol set option
- Edit characters
- Serial and batch number control
- National numeric hand print (NHP) character set options

For more information on any of these topics, see the discussion for the appropriate parameter.

The format of the DFR macro is shown in Figure 2-8. here are the operands you can specify for DFR. The header card contains DFR in the operation field and may contain a module in the name field. If the module name is omitted, IOCS generates a standard module name.

M	FONT = xxxx	Default font for all codes described by format record.
O	REJECT = x	Replacement character for any reject character in the data record read by the 3886. If omitted, X'3F' is assumed.
O	ERASE = YES	Group and character erase symbols are to be recognized. If omitted, NO is assumed.
O	CHRSET = n	Specifies recognizing character (see Figure2-9). If omitted, 0 is assumed.
O	EDCHAR = (x, ...)	Characters that may be deleted from any field that is read. If omitted, no character deletion occurs.
O	BCH = n	Batch numbering is to be performed by 3886. If used, BCHSER is invalid.
O	BCHSER = n	Both batch and serial numbering are to be performed. If specified, BCH is invalid.
O	NATNHP = YES	European Numeric Hand Printing (ENHP) characters 1 and 7 are used. If omitted, NO is assumed, indicating that Numeric Hand Printing (NHP) character 1 + 7 are used.

M=Mandatory; O=Optional

Figure 2-8 **DFR macro operands**

OCR-A			OCR-B			
Numeric Mode	Alphameric Modes		Numeric Mode	Alphameric Mode		
Highspeed Printers or Typewriters	Mode 1 (Highspeed) Printer)	Mode 2 (Typewriter)	Highspeed Printers or Typewriters		Hexa-decimal Code	Format Record Codes
$	$	$	$	$	5B	00
£	£	£	£	£	5B	01
¥	¥	¥	¥	¥	5B	02
	Ñ	Ñ		Ñ	7B	
$	$	$	$	$	5B	03
	Å	Å		Å	5B	
	Æ	Æ		Æ	7B	
	Ø	Ø		Ø	7C	04
$	$	$		Ü Note	5B	
	Ä	Ä		Ä	7B	
	Ö	Ö		Ö	7C	
	Ü	Ü			F0	05

Note: In OCR-A font the Ü is coded as a zero and should be used only in alphabetic fields.

Figure 2-9 **Character set option list**

FONT=code
Specifies the default font for all fields described by the format record. The default font is used to read a field unless another font is specified for an individual field through the DLINT macro. This is the only required operand in the DFR macro. The valid codes and the fonts they represent are:

Code	**Font**
NUMA	Numeric OCR-A font
ANA1	Alphameric OCR-A font (mode1)
ANA2	Alphameric OCR-A font (mode2)
NUMB	Numeric OCR-B font (mode3)
ANB1	Alphameric OCR-B font
NHP1	Numeric hand printing (normal mode)
NHP2	Numeric hand printing (verify mode)
GOTH	Gothic font
MRKA	Mark OCR-A font
MRKB	Mark OCR-B font

For a description of these fonts, see *IBM 3886 Optical Character Reader Component Description and Operating Procedures*, GA21-9147.

REJECT=character
Indicates the character that is to be substituted in the data record for any reject character read by the device. If this parameter is omitted, X'3F' is assumed. Reject characters are characters that are not recognizable by the device.

Note: This note applies to the keywords REJECT and EDCHAR. Apostrophes enclosing the character are optional for all characters except special characters used in macro operands. For a description of these characters, see *OS/VS and DOS/VS Assembler Language*, GC33-4010.

ERASE={YES | <u>NO</u>}
Specifies whether group and character erase symbols are to be recognized as valid symbols. If this operand is not specified, NO is assumed. For more information on group and character erase symbols, see *IBM 3886 Optical Character Reader Component Description and Operating Procedures*, GA21-9147.

CHRSET={<u>0</u> | 1 | 2 | 3 | 4 | 5}
Specifies which one of the options shown in Figure 2-9 is to be used for recognizing characters. If this operand is not entered, 0 is assumed.

EDCHAR=(x,...)
Specifies up to six characters that may be deleted from any field that is read. The EDCHAR parameter in the EDITn keyword in the DLINT macro controls this function for individual fields. If this operand is omitted, no character deletion is performed. See the note under the REJECT parameter for characters that must be specified in quotes. For example, to specify the characters &, >, and), you would code EDCHAR=('&','>',')').

BCH={1 | 2 | 3}
Indicates that batch numbering is to be performed by the 3886. Specifying 1, 2, or 3 indicates that documents routed to a stacker are to be batch numbered. Specifying 1 indicates stacker A, 2 indicates stacker B, 3 indicates both stackers. If this operand is entered, the BCHSER operand is invalid. If neither BCH nor BCHSER are entered, no batch numbering is performed. This parameter is valid only if the serial numbering feature is installed on the 3886. For more information on batch numbering, see *IBM 3886 Optical Character Reader Component Description and Operating Procedures*, GA21-9147.

BCHSER={1 | 2 | 3}
Indicates that both batch and serial numbering are to be performed by the 3886. Specifying 1, 2, or 3 indicates that documents routed to a stacker are to be batch and serial numbered. Specifying 1 indicates stacker A, 2 indicates stacker B, 3 indicates both stackers. If this operand is entered, the BCH operand is invalid. If this operand is omitted, batch and serial numbering are not performed. This parameter is valid only if the serial numbering feature is installed on the 3886. For more information on batch and serial numbering, see *IBM 3886 Optical Character Reader Component Description and Operating Procedures*, GA21-9147.

NATNHP={YES | <u>NO</u>}
Specifies which of the numeric hand printing character set options are used for the numbers 1 and 7. YES indicates that the European Numeric Hand Printing (ENHP) characters 1 and 7 are used; NO indicates the Numeric Hand Printing (NHP) characters 1 and 7 are used. If this operand is not entered, NO is assumed.

M	LFR = nn	Line format record for this line.
M	LINBEG = nn	Specifies beginning of a line.
O	IMAGE = YES	Data record is to be in image mode. If omitted, NO (standard mode) is assumed.
O	NOSCAN = (n,n)	Indicates an area on the document line that is to be ignored by the 3886.
O	FLDn = (n,n,NCRIT,xxx)	Describes a field in a line. n in the FLD keyword may be from 1 to 14, if specified, a corresponding EDITn keyword must follow each FLDn keyword.
O	EDITn = (xxxxxx,EDCHAR)	Specifies editing functions to be performed on the data by 3886. A corresponding FLDn keyword must precede each EDITn keyword.
O	FREND = YES	Indicates last DLINT macro for the format record. If omitted, NO is assumed meaning that further DLINT macros follow.

M=Mandatory; O=optional

Figure 2-10 **DLINT macro operands**

DLINT Macro

The DLINT macro describes one line type in a format group and the individual fields in the line. As many as 26 DLINT macros can be associated with one DFR macro.

The DLINT macro provides line and field information: Line information applies to the entire line; field information describes each of the fields on the line. Up to 14 fields can be scanned on each line.

The format of the DLINT macro is shown in Figure 2-10. Listed here are the operands you can specify for DLINT. The header card contains DLINT in the operation field and may contain a module field. If the module name is omitted, IOCS generated a standard module name.

Line Information Entries

LFR=number
This operand is required. It specifies, the line format record number for the line. The decimal number specified must be in the range of 0 through 63.

The line format record describes the format of one type of line, the line format record number is used to identify the line format record. This number is specified in the READ macro when you read a line of data from a document.

LINBEG=number
This operand is required. It specifies the beginning of a line. The beginning position is the number of tenths of an inch from the left edge of the document to the left boundary of the first field. The limiting range of this position is 4 to 85.

IMAGE={YES | NO}
This operand specifies whether the data record should be in standard mode (IMAGE=NO), or image mode (IMAGE=YES). If this operand is not specified, IMAGE=NO is assumed.

When the standard format is used (IMAGE=NO), all parameters in the DLINT macro are valid. The data read from the document line is edited as specified in the EDITn keywords, the fields in the data record are created as specified in the FLDn keywords, and the standard mode data record then contains fixed-length fields of edited data.

The sequence of operations used to build the standard mode data record in the 3886 is:

1. Recognition of all the characters in the record takes place.
2. Reject characters are removed and the reject code is substituted.
3. Edit characters (specified by the EDCHAR keyword in the DFR macro) are removed from the data (as specified by the EDITn keywords in the DLINT macro).
4. Blanks are removed from the data as specified by the EDITn keywords in the DLINT macro.
5. The length of the fields is checked against the field length specified, the fields are left- or right-justified and padded or truncated as specified in the DLINT macro, and error indicators are set if errors have been detected.

When the image mode is used (IMAGE=YES), all EDITn keywords and the field length parameter in the FLDn keyword are invalid. The image mode data record then contains 14 two-byte field length entries followed by the data fields. Image mode is provided to support exception application requirements where the standard fixed-field edited format does not suffice. It is also applicable for error handling purposes by rereading the same line.

The sequence of operations used to build the image mode data record in the 3886 is:

1. Recognition of all the characters in the record takes place.
2. Reject characters are removed and the reject code is substituted.
3. Each field read and the field lengths are placed in the data record.

NOSCAN=(field-end,...)
Specifies an area on the document line that is to be ignored by the 3886. **Field-end** is a decimal number indicating the number of tenths of an inch from the left edge of the document to the right end of the NOSCAN field. The field immediately to the left of the NOSCAN field must end with an address delimiter rather than a character delimiter.

Field Information Entries

**FLDn=({address-delimiter | character-delimiter}
[,field-length][,{NCRIT | font-code |
NCRIT,font-code}])**
Describes each of the fields in a line. The n suffix is

a number from 1 through 14 and the parameters are the same for keywords FLD1 through FLD14. The following rules apply when specifying these keywords:

- Fields may be described in any order in the macro.
- Each EDITn parameter must follow its associated FLDn parameter.
- The n suffix need not be 1 for the first field in the line; however, the n suffix must increase for each field from left to right on the document line.

address-delimiter is a decimal number that specifies the number of tenths of an inch from the left edge of the document to the right end of the field being defined. The last field in a line must end with an address delimiter.

character delimiter specifies the character that indicates the end of a field. The character delimiter is not considered part of the data; it is not included in the data record nor used in determining the length of the field.

Apostrophes enclosing the characters are optional for all characters except 0-9, and the special characters used in macro operands. For these characters, the apostrophes are required. For a description of these characters, see *OS/VS and DOS/VS Assembler Language*, GC33-4010.

If a field ends with a character delimiter, the next field must be read using a font from the same font group. The font groups are:

- NPH1, NPH2, GOTH
- ANA1, ANA2, NUMA, MRKA
- NUMB, MRKB
- ANB1

field-length is a decimal number specifying the length of the field in the edited record. The length specified cannot be less than 1 or more than 127. If IMAGE=NO is specified, this parameter is required; if IMAGE=YES is specified, this parameter is invalid. The length specified in this parameter refers to the length of the field after any EDITn options have been performed. The sum of the field lengths for a line cannot be greater than 130.

NCRIT indicates that this is not a critical field. If this parameter is omitted, the field is assumed to be critical.

font-code specifies a font for this field, different from the font specified in the DFR macro. If this parameter is not specified, the font specified in the DFR macro is used for the field. For information about the valid codes, see the DRF macro descriptions.

EDITn=({code | EDCHAR | code,EDCHAR})
Describes the editing functions to be performed on the data by the 3886.

The parameters are the same for keywords EDIT1 through EDIT14. There must be a FLDn keyword corresponding with each EDITn keyword you specify. If an EDITn keyword is specified, a code, EDCHAR, or both must be specified. When image mode is used, the EDITn keywords are invalid.

When the editing functions are completed and the field is greater than the specified length, the field is truncated from the right and the wrong length field indicator is set on in the header record. If only blanks are truncated, the wrong length field indicator is not set.

code specifies the blanks to be removed and the fill characters to be added to the field, if any. The valid codes and their meanings are:

Code	**Meaning**
HLBLOF	All high- and low-order blanks are removed, the data is left justified, and the field is padded with blanks on the right (see Note).
ALBLOF	All blanks are removed from the data, the data is left-justified, and the field is padded with blanks on the right.
NOBLOF	No blanks are removed, the data is left-justified, and the field is padded on the right with blanks.
HLBHIF	All high- and low-order blanks are removed, the data is right-justified, and the field is padded to the left with EBCDIC zeros (X'F0') (see *Note*).
ALBHIF	All blanks are removed, the data is right-justified, and the field is padded with EBCDIC zeros (X'F0') on the left.

ALBNOF	All blanks are removed; the data must be equal in length to the field length specified. No padding is done.

Note: Two consecutive embedded blanks is the maximum number sent.

If the EDITn keyword is omitted or if EDITn is specified and the code is omitted, ALBLOF is assumed.

EDCHAR indicates that the characters specified in the EDCHAR keyword of the DFR macro are to be deleted from the field. If this parameter is omitted, the characters are not deleted.

FREND={YES | NO}
Indicates whether this is the last DLINT macro for the format record. NO indicates more DLINT macros follow; YES indicates this is the last one. If this keyword is omitted, NO is assumed.

DTFDU Macro

The DTFDU macro defines sequential (consecutive) processing for a file contained on a diskette. Note that special records (deleted or sequential relocated records) on an input file are skipped, and not passed to the user. The DTFDU macro cannot be used when a diskette file is to be processed under POWER. In this case, use the DTFDI macro.

A DTFDU entry is included for each sequential input or output diskette file processed in the program. The DTFDU header entry and a series of detail entries describe the file. Enter the symbolic name of the file in the name field and DTFDU in the operation field. The detail entries follow the DTFDU header card in any order. The entries for the DTFDU macro are discussed here and are summarized in Figure 2-11.

CMDCHN=nn
This operand is specified to indicate the number of Read/Write CCWS to be command chained. Valid entries are 1, 2, 13, or 26; 1 is assumed if this entry is omitted. For each CCW specified by this operand, one record is processed (for example, if CMDCHN=13, 13 records are command chained and are processed -- read or written -- as a group). For entries of 2, 13, or 26, either the IOREG operand or the WORKA operand must be specified.

DEVADDR=SYSxxx
This operand specifies the symbolic unit (SYSxxx) associated with the file if an extent statement specification is not provided. And EXTENT statement is not required for single-volume input files. If an EXTENT statement is provided, its specification overrides any DEVADDR specification. SYSxxx represents an actual I/O device address, and is used in the ASSGN job control statement to assign the actual I/O device address to this file.

A list of symbolic units applying to DTFDU can be found in the *Symbolic Unit Addresses* section of *The Macro System* chapter.

DEVICE=3540
This operand specifies that the file to be processed is on the 3540. If this parameter is unspecified, the 3540 is assumed.

EOFADDR=name
This operand specifies the symbolic name of your end-of-file routine. IOCS automatically branches to this routine on an end-of-file condition. You can perform any operations required for the end-of-file in this routine (you will generally issue the CLOSE or CLOSER macro).

ERREXT=YES
This operand enables your ERROPT routine to return to DUMODFx with ERET macro. It also enables permanent errors to be indicated to your program. For ERREXT facilities, the EROPT operand must be specified. However, to take full advantage of this option give the ERROPT=name operand.

ERROPT={IGNORE | SKIP | name}
Specify this operand if you don't want a job to be terminated when a permanent error cannot be corrected in the diskette error routine. If attempts to reread a chain of records are unsuccessful, the job is terminated unless the ERROPT entry is included. Either IGNORE, SKIP, or the name of an error routine can be specified. The functions of these parameters are described below.

IGNORE	The error condition is ignored. The records are made available for processing. On output, the error condition is ignored and the records are considered written correctly.

X		M	EOFADDR = xxxxxxxx	Name of your end-of-file routine. (Required for input only)
X	X	M	IOAREA1=xxxxxxxx	Name of first I/O area
X	X	M	RECSIZE=nnn	Length of one record in bytes
X	X	O	CMDCHN=nn	Number of read/write CCWs (records) to be command-chained
X	X	O	DEVADDR=SYSxxx	Symbolic unit, required only when not provided on an EXTENT statement
X	X	O	DEVICE=3540	Must be 3540. If omitted, 3540 is assumed.
X	X	O	ERREXT=YES	Indicates additional errors and ERET desired. Specify ERROPT
X	X	O	ERROPT=xxxxxxxx	IGNORE, or SKIP, or name of error routine
X	X	O	FEED=xxx	YES means feed at end-of-file. NO means no feed, YES assumed if omitted.
	X	O	FILESEC=YES	YES means create file secure.
X	X	O	IOAREA2=xxxxxxxx	Name of second I/O area, if two areas are used.
X	X	O	IOREG=(nn)	Register number. Omit WORKA. General register 2 to 12 in parentheses.
X	X	O	MODNAME=xxxxxxxx	Name of DUMODFx logic module for this DTF. If omitted, IOCS generates standard name.
X	X	O	RDONLY=YES	Generates a read-only module. Requires a module save area for each task using the module.
X	X	O	SEPASMB=YES	DTFDU is to be assembled separately.
X	X	O	TYPEFLE=xxxxxx	INPUT or OUTPUT. If omitted, INPUT is assumed.
X		O	VOLSEQ=YES	YES means OPEN is to check sequencing of multi-volume files.
X	X	O	WORKA=YES	GET or PUT specifies work area. Omit IOREG.
	X	O	WRTPROT=YES	File will be created with Write-Protect on (cannot be overwritten).

M=Mandatory; O=Optional

Figure 2-11 **DTFDU macro operands**

SKIP	No records in the error chain are made available for processing. The next chain of records is read from the diskette, and processing continues with the first record of that chain. On output, the SKIP option is the same as the IGNORE option.
name	IOCS branches to your error routine named by this parameter regardless of whether or not ERREXT=YES is specified. In this routine you can process or make note of the error condition as desired.

IF ERREXT is not specified, register 1 contains the address of the first record in the error chain. When processing in the ERROPT routine, reference records in the error chain by referring to the address supplied in register 1. The contents of the IOREG register or work area are variable and should not be used to process error records. Also, GET macros must not be issued for records in the error chain. 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, return control to IOCS by branching to the address in register 14. For a read error, IOCS skips that error chain of records, and makes the first record of the next chain available for processing in the main program.

If ERREXT is specified, register 1 contains the address of a two part parameter list containing the 4-byte DTFDU address and the 4-byte address of the first record in the error chain. Register 14 contains the return address. Processing is similar to that described above except for addressing the records in error.

At the end of its processing, the routine returns to LIOCS by issuing the ERET macro.

For an input file
- The program skips the error chain and reads the next chain with an ERET SKIP.
- Or, ignores the error with an ERET IGNORE.
- Or, it makes another attempt to read the error chain with an ERET RETRY.

For an output file
- The program ignores the error condition ERET IGNORE or ERET SKIP.
- Or, attempts to write the error chain with an ERET RETRY macro. Bad spot control records (1, 2, 13, or 26 records depending on the CMDCHN Factor) are written at the current diskette address, and the write chain is retried in the next 1, 2, 13, or 26 (depending on the CMDCHN factor) sectors on the disk.

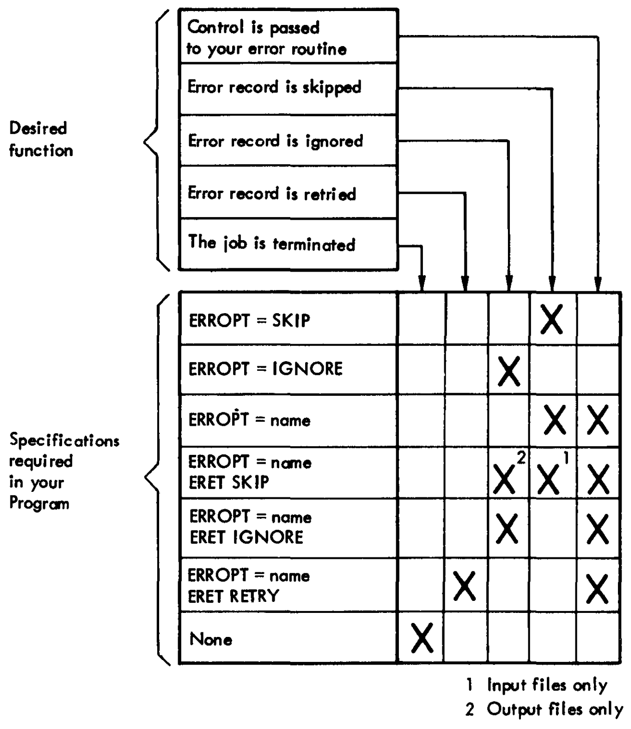
Also, for an output file, the only acceptable parameters are IGNORE or **name**.

The DTFDU error options are shown in figure 2-12. The figure is divided into two parts: the lower part lists the error conditions which you specify in the DTF, and the upper part shows the action resulting from these specifications when an error occurs.

FILESEC=YES
This operand applies to output only. On output it causes OPEN or OPENR to set the security flag in the file label. For subsequent input, the security flag causes an operator message to be written. The operator must then reply in order to make the file available to be read.

FEED={YES | NO}
If YES is specified for this operand, when end of file is reached, the diskette being processed is fed to the stacker and a new diskette is fed to the disk drive (providing another diskette is still in the hopper). If NO is specified, the diskette is left mounted for the next job.

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. Note that you should provide an I/O area equal in size to the result obtained from multiplying the RECSIZE entry by the CMDCHN entry.



Specifications required in your Program	The job is terminated	Error record is retried	Error record is ignored	Error record is skipped	Control is passed to your error routine
ERROPT = SKIP				X	
ERROPT = IGNORE			X		
ERROPT = name				X	X
ERROPT = name ERET SKIP			X[2]	X[1]	X
ERROPT = name ERET IGNORE			X		X
ERROPT = name ERET RETRY		X			X
None	X				

Desired function
Specifications required in your Program
1 Input files only
2 Output files only

Figure 2-12 **DTFDU error options**

IOAREA2=name
If two I/O areas are used by GET or PUT, this operand is specified. Note that you should provide an I/O area equal in size to the result obtained from multiplying the RECSIZE entry by the CMDCHN entry.

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 or OPENR time, for output files, IOCS puts the address of the area where the user can build a record in this register. The same register can 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. If this operand is specified, omit the WORKA operand.

This operand must be specified if the CMDCHN factor is more than one for either input or output and records are processed in one I/O area, or if two I/O areas are used and records are processed in both I/O areas.

MODNAME=name
This operand specifies the name of the logic module which is to process the file. If the logic module is assembled with the program, MODNAME must specify the same name as the DUMODx macro. If this entry is omitted, standard names are generated for calling the logic module. If two DTF macros 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 double-word aligned save area. Each task should have its own uniquely defined save area. When an imperative macro (except OPEN, OPENR) is issued, register 13 must contain the address of the save area associated with the task. 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).

If an ERROPT routine issues I/O macros using the same read-only module that caused control to pass to the error routine, your problem program must provide another save area. One save area is used for the normal I/O operations, and the second for input/output operations in the ERROPT 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 cannot be reentered and no save area need be established.

RECSIZE=nnn
This operand specifies (in bytes) the length of each record in the input/output area (1-128 bytes).

SEPASMB=YES
Include this operand only if the DTFDU 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 macro assumes that the DTF is being assembled with the problem program and no CATALR card is punched.

TYPEFLE={INPUT | OUTPUT}
This operand indicates whether the file is an input or output file.

VOLSEQ=YES
This operand is only valid on input. If specified, it causes OPEN or OPENR to ensure that the volume sequence numbers (if specified) of a multi-volume

file are in ascending and sequential order. If the volume sequence number of the first volume processed is blank, no volume sequence checking is done.

WORKA=YES
If I/O records are processed, or built, in work areas instead of in the I/O areas, specify this operand. You must set up the work area in storage. The address of the work area, on a general register containing the address, must be specified in each GET or PUT macro. For a GET or PUT macro, IOCS moves the record to or from, the specified work area.

When this operand is specified, the IOREG operand must be omitted.

WRTPROT=YES
This operand indicates that an output file will be created with Write-Protect (meaning that the file cannot be overwritten). For 3540 support, this has no affect on subsequent input processing of the file.

DUMODFx Macro

Two categories of file characteristics are defined for diskette unit module generation macros:

- DUMODFI - Diskette Unit **MOD**ule, Fixed length records, Input file.
- DUMODFO - Diskette Unit **MOD**ule, Fixed length records, **O**utput file.

The macro operation and the keyword operands define the characteristics of the module. The operands for the two macros are explained in the following section.

DUMODFx Operands

A module name can be contained in the name field of this macro. The macro operation is contained in the operation field, either DUMODFI (for input) or DUMODFO (for output). The operands are contained in the operand field.

ERREXT=YES
Include this operand if permanent 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.

ERROPT=YES
This operand applies to both DUMODFx macros. This operand is included if the module handles any of the error options for an error chain. Logic is generated to handle any of the three options (IGNORE, SKIP, or name) regardless of which option is specified in the DTF. This module also processes any DTF in which the ERROPT operand is not specified.

If this operand is not included, your program is canceled whenever a permanent error is encountered.

RDONLY=YES
This operand causes a read-only module to be generated. If this operand is specified, any DTF used with this module must have the same operand.

SEPASMB=YES
Include this operand only if the logic module is assembled separately. This causes a CATALR card with the module name (standard or user-specified) to be punched ahead of the object deck, and defines the module name as an ENTRY point in the assembly. If the operand is omitted, the program assumes that the logic module is being assembled with the problem program and no CATALR is punched.

Standard DUMOD Names

Each name begins with a 3-character prefix (IJN) and continues with a 5-character field corresponding to the options permitted in the generation of the module, as shown below. DUMODFx name = IJNabcde

a	= D	
b	= I	DUMODFI
	= O	DUMODFO
c	= C	ERROPT=YES and ERREXT=YES
	= E	ERROPT=YES
	= Z	neither is specified
d	= Z	
e	= Y	RDONLY=YES
	= Z	RDONLY not specified

Subset/Superset DUMOD Names

The following diagram illustrates the subsetting and supersetting allowed for DUMOD names.

```
        * + * *
I J N D I C Z Y
        O E   Z
          Z
```

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

DTFMR Macro

DTFMR defines an input file processed on a 1255, 1259, or 1419 magnetic character reader, or a 1270 or 1275 optical character reader/sorter. Some general characteristics of such processing are discussed below before the parameters of the DTFMR macro are described.

Characteristic of Magnetic Ink Character Reader (MICR) and Optical Reader/Sorter Processing
Important general characteristics of Magnetic Ink Character Reader (MICR) and Optical Reader/Sorter processing are given in the *DOS/VS Data Management Guide,* GC33-5372.

In addition, examples of GET-PUT document processing and multiple 1419 operation (either all single or dual) will be found in *DOS/VS System Generation,* GC33-5377.

MICR Document Buffer
The MICR Document Buffer provides you with processing status indicators and detected error indicators. Before you can begin any MICR programming, you must be aware of the purpose and format of this buffer.

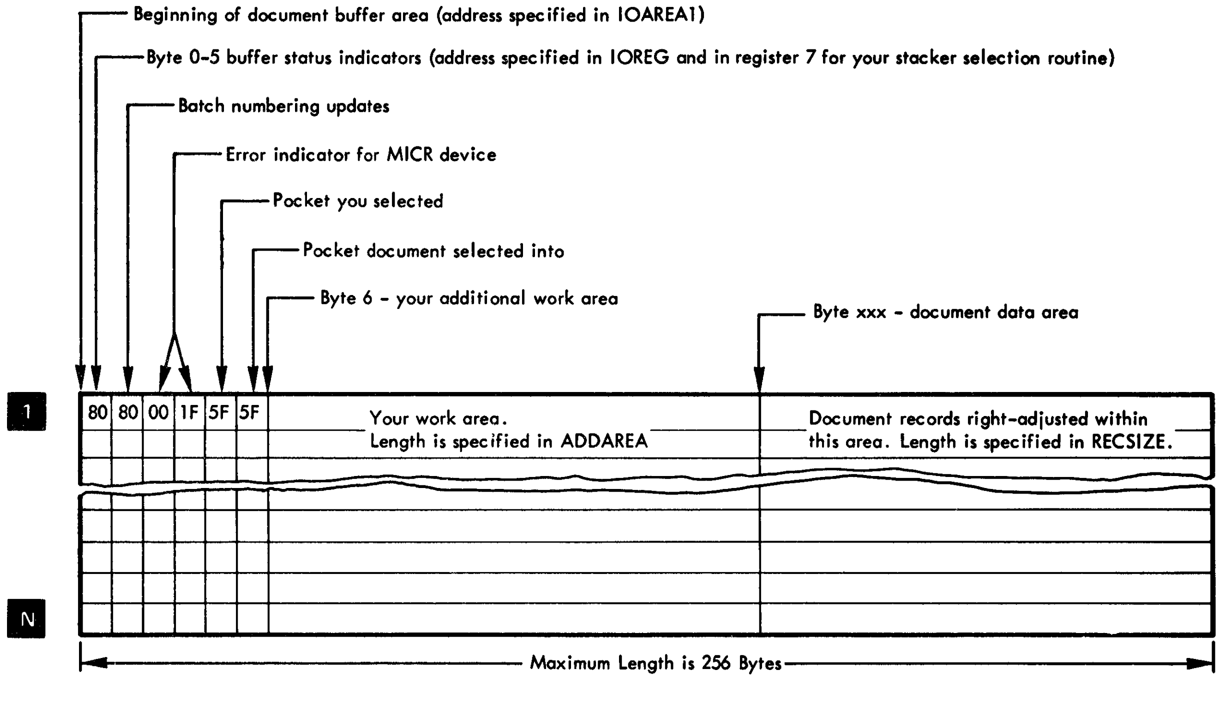
Figure 2-13 is a storage map of the document buffer. The minimum number of document buffers you may specify is 12, and the maximum number is 254. Before any data is read into the document buffer, logical IOCS sets the entire buffer, including the status indicators, to binary zeros. The processing macro--GET if your program uses one MICR device, or READ if your program uses more than one MICR device--then engages the device, and documents are read into the I/O area until the MICR device is out of documents, or until the I/O area is filled. The external interrupt routine of the supervisor continually monitors the reading of data so that processing of other document buffers is never disrupted. At the completion of each read for a MICR document, the external interrupt routine interrupts your program to give control to your stacker selection routine which then determines pocket selection for that document.

The MICR document buffer format is given in *Appendix E*.

Stacker Selection Routine for MICR
Your stacker selection routine resides in your program area and gains control of the system whenever a document is ready to be stacker selected. This routine determines the pocket (stacker) selected to receive the document 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, your 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 your stacker selection routine for any purpose. Because the program can be interrupted at any time, the contents of these registers are unpredictable.
5	When your stacker selection routine is entered, this register contains the address of the 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 the buffer (see *Appendix E*) indicate the read status of the document.

Before entering your 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 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. The reject code X'CF' indicating that the document is placed in the reject pocket is placed in byte 5.



Beginning of document buffer area (address specified in IOAREA1)
Byte 0-5 buffer status indicators (address specified in IOREG and in register 7 for your stacker selection routine)
Batch numbering updates
Error indicator for MICR device
Pocket you selected
Pocket document selected into
Byte 6 - your additional work area
Byte xxx - document data area
1
80 80 00 1F 5F 5F
Your work area. Length is specified in ADDAREA
Document records right-adjusted within this area. Length is specified in RECSIZE.
N
Maximum Length is 256 Bytes

1 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 3).

N Number of buffers is limited only by the amount of storage available (see BUFFERS operand).

Figure 2-13 MICR document buffer

The command codes to be used to select pockets are:

Pocket	Code	
A	X'AF'	(1270, except models 1 and 3, 1275, and 1419 only)
B	X'BF'	(1275 and 1419 only)
0	X'0F'	
1	X'1F'	
2	X'2F'	
3	X'3F'	
4	X'4F'	
5	X'5F'	
6	X'6F'	(except 1270 models 1 and 3)
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 macro with the MR operand (required method), you can request a batch numbering update. You can do this only within your 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), you 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 appropriate MICR publications listed in the latest SRL Newsletter for a more complete description of device timings.

Failure to reinstruct the 1255, 1259, 1270, 1275, 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).

Programming Considerations for 1419 or 1275 Stacker Selection
The stacker selection routine operates in the program state with the protection key of its program and with I/O and external interruptions disabled. If your stacker selection routine fails to return to the supervisor (loops indefinitely), there is no possible recovery. If such looping occurs, the system must be re-IPLed to continue operation. It is therefore recommended that you thoroughly debug your stacker selection routine in a dedicated environment.

In your stacker selection routine, no system macro 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 erroneous processing by any concurrent program using floating-point instructions.

When processing with the dual address adapter (1419 or 1275), you have more time for your stacker selection routine. The only additional processing you 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.

Checkpointing MICR Files
This topic is discussed in the section *Notes for DASD and MICR Files* under *CHKPT Macro* in the *Supervisor Macros* chapter.

DTFMR Operands
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 2-14.

ADDAREA=n
This operand must be included only if an additional buffer work area is needed. The parameter n specifies the number of additional bytes you desire in each buffer. The sum of the ADDAREA and RECSIZE specifications must be 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 the single address adapter is used, this operand must be omitted.

BUFFERS={<u>25</u> | n}
This operand is included to specify the number of buffers in the document buffer area. The limits for n are 12 and 254. 25 is assumed if this operand is omitted.

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

ERROPT=name
This operand may be included only if the CHECK macro 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 your responsibility to exit from this routine (see the *CHECK Macro* in the *Magnetic Reader Macros* section later in this chapter).

M	DEVADDR=SYSnnn	Symbolic unit assigned to the magnetic character reader.
M	IOAREA1 = xxxxxxxx	Name of the document buffer area.
O	ADDAREA=nnn	Additional document buffer area (ADDAREA+RECSIZE=250). If omitted, no area is alloted.
O	ADDRESS=DUAL	Must be included only if the device is a 1419 or 1275 with a dual address adapter.
O	BUFFERS=nnn	Specifies the number of buffers needed. If omitted, 25 is assumed.
O	ERROPT=xxxxxxxx	Name of your error routine. Required only if the CHECK macro is used.
O	EXTADDR=xxxxxxxx	Name of your stacker selection routine. Required only if SORTMDE=ON.
O	IOREG=(nn)	Pointer register number. If omitted, register 2 is assumed. General registers 2-12, written in parentheses.
O	MODNAME=xxxxxxxx	Name of your I/O module. Required only if a nonstandard module is referenced.
O	RECSIZE=nnn	Specifies the maximum record length. If omitted, 80 is assumed.
O	SECADDR=SYSnnn	Specifies secondary symbolic unit assigned to (dual address) 1275 or 1419. Required only if LITE macro is used.
O	SEPASMB=YES	Required only if the DTF is assembled separately; otherwise it should be omitted.
O	SORTMDE=xxx	ON-1255/1259/1270 or program sort mode used; OFF-1419/1275 sort mode used. If omitted, ON is assumed.

M=Mandatory; O=Optional

Figure 2-14 **DTFMR macro operands**

EXTADDR=name
This operand specifies the name of your 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.

IOAREA1=name
This operand specifies the name of the document buffer area used by the file. Figure 2-13 shows the format of the document buffer area.

IOREG={(2) | (r)}
This operand specifies the general-purpose register (2-12) that the IOCS routines and your 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. Register 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, your program may need to store the address supplied by IOCS for each record.

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

RECSIZE={80 | n}
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 of 80 is assumed. The sum of the ADDAREA and RECSIZE specifications must be 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 problem program and no CATALR card is punched.

SORTMDE={ON | OFF}
This operand specifies the method of sorting done on the 1419. SORTMDE=ON indicates that the program sort mode is being used. SORTMDE=OFF indicates that sorting is under control of the magnetic character reader. If the operand is omitted, the program sort mode is assumed.

MRMOD Macro

The first card contains MRMOD in the operation field and may contain a module name in the name field. If a module name is omitted, the following standard module name is generated by IOCS:

$$\text{IJU}\left\{\begin{matrix}\text{S}\\\text{D}\end{matrix}\right\}\text{ZZZZ}$$

(S = single address adapter, and D = dual address adapter). The operands you can specify for MRMOD are listed below.

ADDRESS={SINGLE | 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 (nnn) equal to the corresponding value specified in the DTFMR macro.

SEPASMB=YES
Include this operand only if the module is assembled separately. This causes a CATALR card with the module name (standard or user-specified) to be punched ahead of the object deck and defines the module name as an ENTRY point in the assembly. If the operand is omitted, the program assumes that the DTF is being assembled with the problem program and no CATALR card is punched.

Review Mandatory
1. Input
2. Output
3.

DTFMT Macro TAPE

A DTFMT macro is included for each EBCDIC or ASCII magnetic tape input or output file that is to be processed. Enter the symbolic name of the file in the name field and DTFMT in the operation field. The detail entries follow the header card in any order. The entries are discussed here and illustrated in Figure 2-15.

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

BLKSIZE=n
Enter the length 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 work files, the record to be read or written will be truncated to fit in the I/O area. The maximum block size is 32,767 bytes. The minimum size of physical tape record (gap to gap) is 12 bytes. A record of eleven bytes or less is treated as noise.

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.

Applies to					
Input	Output	Work			
x	x	x	M	BLKSIZE=nnnnn	Length of one I/O area in bytes (maximum = 32,767).
x	x	x	M	DEVADDR=SYSxxx	Symbolic unit for tape drive used for this file.
x		x	M	EOFADDR=xxxxxxxx	Name of your end-of-file routine.
x	x	x	M	FILABL=xxxx	(NO, STD, or NSTD). If NSTD specified, include LA-BADDR. If omitted, NO is assumed.
x	x		M	IOAREA1=xxxxxxxx	Name of first I/O area.
x	x		O	ASCII=YES	ASCII file processing is required.
x	x		O	BUFOFF=nn	Length of block prefix if ASCII=YES.
x			O	CKPTREC=YES	Checkpoint records are interspersed with input data re-cords. IOCS bypasses checkpoint records.
x	x	x	O	ERREXT=YES	Additional errors and ERET are desired.
x	x	x	O	ERROPT=xxxxxxxx	(IGNORE, SKIP, or name of error routine). Prevents job termination on error records.
x	x	x	O	HDRINFO=YES	Print header label information if FILABL=STD.
x	x		O	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. General registers 2-12, written in parentheses.
x	x		O	LABADDR=xxxxxxxx	Name of your label routine if FILABL=NSTD, or if FILABL=STD and user-standard labels are processed.
x			O	LENCHK=YES	Length check of physical records if ASCII=YES and BUFOFF=4.
x	x	x	O	MODNAME=xxxxxxxx	Name of MTMOD logic module for this DTF. If omitted, IOCS generates standard name.
		x	O	NOTEPNT=xxxxxx	(YES or POINTS). YES if NOTE, POINTW, POINTR, or POINTS macro used. POINTS if only POINTS macro used.

M=Mandatory; O=Optional

Figure 2-15 **DTFMT macro operands (part 1 of 2)**

Applies to

Input	Output	Work			
x	x	x	O	RDONLY=YES	Generate read-only module. Requires a module save area for each task using the module.
x		x	O	READ=xxxxxxx	(FORWARD or BACK). If omitted, FORWARD assumed.
x	x	x	O	RECFORM=xxxxxx	(FIXUNB, FIXBLK, VARUNB, VARBLK, SPNUNB, SPNBLK, or UNDEF). For work files use FIXUNB or UNDEF. If omitted, FIXINB is assumed.
x	x		O	RECSIZE=nnnn	If RECFORM=FIXBLK, no. of characters in record. If RECFORM=UNDEF, register number. Not required for other records. General registers 2-12, written in parentheses.
x	x	x	O	REWIND=xxxxxx	(UNLOAD or NORWD). Unload on CLOSE or end-of-volume, or prevent rewinding. If omitted, rewind only.
x	x	x	O	SEPASMB=YES	DTFMT is to be assembled separately.
	x		O	TRMARK=NO	Prevent writing a tapemark ahead of data records if FILABL=NSTD or NO.
x	x	x	O	TYPEFLE=xxxxxx	(INPUT, OUTPUT, or WORK). If omitted, INPUT is assumed.
	x		O	VARBLD=(nn)	Register number, if RECFORM=VARBLK and records are build in the output area. General registers 2-12 are written in parentheses.
x			O	WLRERR=xxxxxxxx	Name of wrong-length-record routine.
x	x		O	WORKA=YES	GET or PUT specifies work area. Omit IOREG.

M = Mandatory; O = Optional

Figure 2-15 DTFMT macro operands (part 2 of 2)

For ASCII tapes, 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={0 | n}
This operand indicates the length of the block prefix. Enter the length of the block prefix if processing of the block prefix is required. This operand can only be included when ASCII=YES is specified. The contents of this field are not passed on to you.

n can have the following values:

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 an input tape 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 to be associated with the file. An ASSGN job control statement assigns an actual channel and unit number to the unit. The ASSGN job control statement contains the same symbolic name as DEVADDR. When processing ASCII tapes, you must specify a programmer logical unit (SYSnnn).

EOFADDR=name
This operand specifies the name of your 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 your routine, you can perform any operations required for the end of file (generally you issue 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, if the unit is assigned to SYSRDR or SYSIPT when a /* is read. You must determine, in your routine, that this actually is the end of the file.

ERREXT=YES
This operand enables your ERROPT or WLRERR routine to return to MTMOD with the ERET (error return) macro. It also enables unrecoverable I/O errors occurring before data transfer takes place to be indicated to your 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 when an error block is encountered.

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. After this the job is automatically terminated unless this ERROPT entry is included to specify other procedures to be followed in case of 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 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 error is ignored and the physical record containing the error is treated as a valid record. 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 error is ignored and the physical record containing the error is treated as a valid record. The remainder, if any, of the spanned record segments are written.
name	IOCS branches to your error routine named by this parameter regardless of

whether ERREXT=YES is specified. In this routine, you process or make 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, refer the error block, or records in the error block to the address supplied in register 1. The contents of the IOREG register or work area (if either is specified) are variable and therefore should not be used for error processing. Furthermore, your routine must not issue any GET macros 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, return control to IOCS by branching to the address in register 14. IOCS skips the physical record in error and makes the next logical record available for processing in the main 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 code is generated and an MNOTE is issued. If an error condition occurs, the job is canceled.

Register 14 contains the return address. Processing is similar to that described for cases where 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 non-data 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 program must consider the device inoperative and must not attempt further processing on it. Any subsequent 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 results in job termination.

Figure 2-16 shows the DTFMT error options for various combinations of error specifications and errors.

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.

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

Note: For ASCII tapes, the pointer to the block in error indicates the first logical record following the block prefix.

FILABL={<u>NO</u> | STD | NSTD}
This operand specifies what type of labels are to be processed. STD indicates standard labels, NO indicates no labels, and NSTD indicates nonstandard labels. You must furnish a routine to check or create the nonstandard labels by using your own I/O area and an EXCP macro 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.

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 file with standard labels is opened. It also prints the filename, logical 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 name of 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 the name of a second 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 if:

- two input or output areas are used.
- blocked input or output records are processed in the I/O area.
- variable unblocked records are read.
- undefined records are read backwards.
- 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 you can build a record. Any register (2-12) may be specified.

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

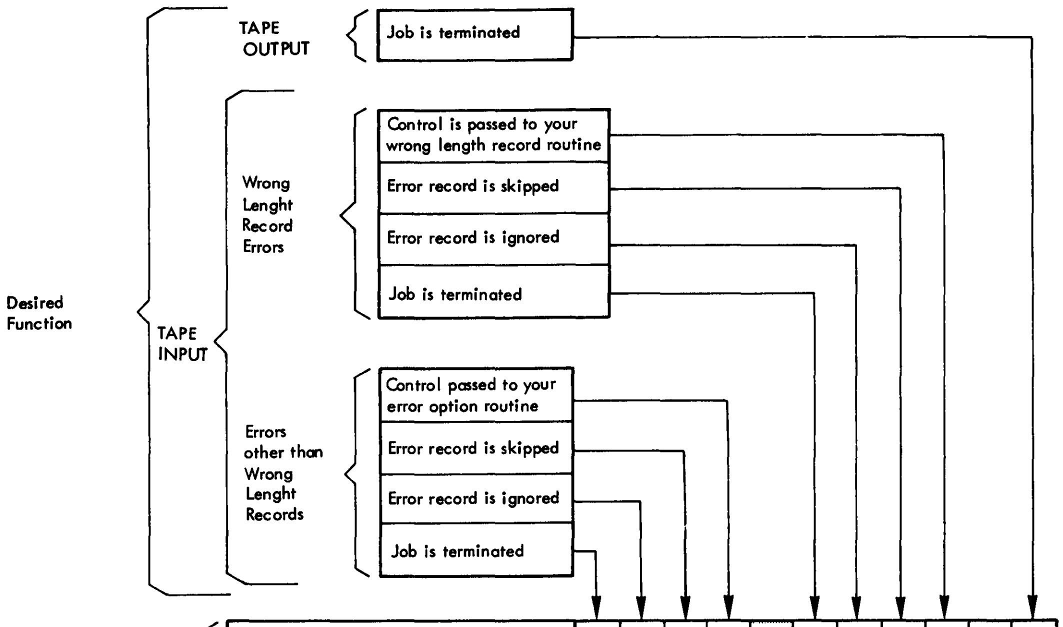
LABADDR=name
Enter the symbolic name of your routine to process user-standard or nonstandard labels. See the *Tape Input Files* section of the *Label Processing* chapter.

For ASCII tapes, this operand may only be used for writing and checking user standard labels which conform to American National Standards Institute, Inc., standards. You must process these labels in EBCDIC. Nonstandard user labels are not permitted.

LENCHK=YES
This operand applies only to ASCII tape input 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 is assembled with the program, MODNAME must specify the same name as the MTMOD macro. If this entry is omitted, standard names are generated for calling the logic module. If two DTF macros 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 macros are issued for a tape work file. If POINTS is specified, only POINTS macros can be issued for tape work files. The NOTEPNT operand must not be specified for ASCII tape files because work files are not supported.


Desired Function
TAPE OUTPUT
Job is terminated
TAPE INPUT
Wrong Lenght Record Errors
Control is passed to your wrong length record routine
Error record is skipped
Error record is ignored
Job is terminated
Errors other than Wrong Lenght Records
Control passed to your error option routine
Error record is skipped
Error record is ignored
Job is terminated
Specifications required in your Program

	Errors other than Wrong Lenght Records: Job is terminated	Errors other than Wrong Lenght Records: Error record is ignored	Errors other than Wrong Lenght Records: Error record is skipped	Errors other than Wrong Lenght Records: Control passed to your error option routine		Wrong Lenght Record Errors: Job is terminated	Wrong Lenght Record Errors: Error record is ignored	Wrong Lenght Record Errors: Error record is skipped	Wrong Lenght Record Errors: Control is passed to your wrong length record routine		Tape Output: Job is terminated
ERROPT = IGNORE		X					X				
ERROPT = name			X	X				X	X		X
ERROPT = SKIP			X					X			
WLRERR = name	X							X	X		
ERROPT = IGNORE, WLRERR = name		X						X	X		
ERROPT = name, WLRERR = name			X	X				X	X		
ERROPT = SKIP, WLRERR = name			X					X	X		
WLRERR = name ERET IGNORE		X					X				
ERROPT = name ERET RETRY	X					X					
WLRERR = name ERET SKIP			X	X				X	X		
WLRERR = name ERET IGNORE	X						X				
WLRERR = name ERET RETRY	X					X					
WLRERR = name ERET SKIP	X							X	X		
ERROPT = name, WLRERR = name ERET IGNORE		X					X				
ERROPT = name, WLRERR = name ERET RETRY	X					X					
ERROPT = name, WLRERR = name ERET SKIP			X	X				X	X		
NONE	X					X					X

ERET Macro Options:	DTF Parameters:
IGNORE	ERROPT = name
RETRY	ERROPT = IGNORE
SKIP	ERROPT = SKIP
	WLRERR = name

Figure 2-16 **DTFMT error options**

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 task should have its own uniquely defined save area and each time an imperative macro (except OPEN (or OPENR) or LBRET) is issued, register 13 must contain the address of the save area associated with the task. 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* in the *Multitasking Macros* chapter.

If an ERROPT or WLRERR routine issues I/O macros which use the same read-only module that caused control to pass to either error routine, your 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 error routine, register 13 must be set to the save area address originally specified for the task.

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

READ={<u>FORWARD</u> | BACK}
This operand specifies the direction in which the tape is read. If READ=BACK is specified and a wrong-length record smaller than the I/O area is encountered, the record is read into the I/O area right-justified.

RECFORM={<u>FIXUNB</u> | FIXBLK | VARUNB | VARBLK | SPNBLK | SPNUNB | UNDEF}
This operand specifies the type of EBCDIC or ASCII records in the input or output file. Enter one of the following parameters:

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

Work files may only use FIXUNB or UNDEF.

RECSIZE={n | (r)}
For fixed-length blocked records, this operand is required. It specifies the number of characters in each record.

When processing spanned records, you must specify RECSIZE=(r) where r is a register that contains the length of each record.

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, you must load the length of each record into the register before you issue 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 virtual storage.

REWIND={UNLOAD | NORWD}
If this specification is not included, tapes are automatically rewound to load point, but not unloaded, on an OPEN (or OPENR) or CLOSE (or CLOSER) macro or on an end-of-volume condition. If other operations are desired for a tape input or output file, specify:

UNLOAD	to rewind the tape on an OPEN (or OPENR) or to rewind and unload on CLOSE (or CLOSER) or 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 end-of-file condition.

SEPASMB=YES
Include this operand only if the DTFMT 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 problem program and no CATALR card is punched.

TPMARK=NO
A tapemark is normally written for an output file if nonstandard labels are specified (FILABL=NSTD).

If no tapemark is desired, this operand should be specified. This operand is ignored if standard labels are specified (FILABL=STD). For unlabeled tapes, TPMARK=NO is the default.

TYPEFLE={INPUT | OUTPUT | WORK}
Use this operand to indicate whether the file is used for input or output. 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 File Macros for Tape and Disk* under the *Processing Macros* section of the present chapter.

The specification of WORK in this operand is not permitted for ASCII files.

VARBLD=(r)
This entry is required whenever variable-length blocked records are built directly in the output area (no work area specified). It specifies the number (r) of a general-purpose register (2-12) that always contains 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 you in the VARBLD register after the PUT macro is issued for a variable-length record. You can then compare the length of the 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, issue a TRUNC macro to transfer the completed block of records to the tape. The current record is then built as the first record of the next block.

WLRERR=name
This operand applies only to tape input files. It specifies the name of your 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 that specified in 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 specified block size minus the residual count. If the block to be read is larger than that specified in 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 that specified in the BLKSIZE parameter, and the remainder of the original block is truncated.

Your WLRERR routine can perform any processing desired for wrong length records. However, it must not issue GET macros 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, either control is 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), the 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 specified in the 4-byte block prefix. In this case, the WLR bit (byte 5, bit 1) in the DTF table 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 conditions results:

- If the ERROPT entry is included for this file, the wrong-length record is treated as an error block, and handled according to your 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 in the I/O areas, specify this operand. You must set up the work area in virtual storage. The address of the work area, or a general-purpose 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.

MTMOD Macro

Listed here are the operands you can specify for MTMOD. The first card contains MTMOD in the operation field and may contain a 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 work files. If omitted, EBCDIC file processing is assumed.

CKPTREC=YES
Include this operand if input tapes processed by the module contain checkpoint records interspersed among the data records. The module also processes tapes 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 work files, 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. Code is generated to handle any of the three options (IGNORE, SKIP, or name). The module processes also 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}
This operand applies only to work files (EBCDIC only).

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

Modules specifying either one of the two options also process work files for which the NOTE/POINT operand is not specified in the DTF. Modules specifying YES also process work files specifying only POINTS.

RDONLY=YES
This operand causes a read-only module to be generated. Whenever this operand is specified, any DTF used with the module must have the same operand.

READ={FORWARD | BACK}
This operand generates a module that reads tape files forward or backward. If forward is specified, only code to read tape forward is generated. Any DTF used with the module must not specify BACK in the READ parameter statement.

If the parameter is BACK, code 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 multi-volume files.

This entry is not needed for work files.

**RECFORM={FIXUNB | FIXBLK | VARUNB |
VARBLK | SPNBLK | SPNUNB |
UNDEF}**
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 module is generated that allows processing of *both* fixed-length blocked and fixed-length unblocked records. Similarly, if VARUNB or VARBLK is specified, a 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 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 specifies RECFORM=FIXUNB, either RECFORM=FIXUNB or RECFORM=FIXBLK may be specified in the DTF.

This operand is not needed for work files.

If this operand is omitted, the module generated will allow processing of both fixed-length blocked and fixed-length unblocked records.

SEPASMB=YES
Include this operand only if the module is assembled separately. This causes a CATALR card with the module name (standard or user-specified) to be punched ahead of the object deck and defines the module name as an ENTRY point in the assembly. If the operand is omitted, the program assumes that the DTF is being assembled with the problem program and no CATALR card is punched.

TYPEFLE={<u>OUTPUT</u> | INPUT | WORK}
This operand generates a module that processes either GET/PUT macros or READ/WRITE, NOTE/POINT and CHECK macros for work files (EBCDIC only). If the parameter of the operand specifies WORK, code to process work files is generated. Otherwise, a module to handle both input and output files 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.

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.

Standard MTMOD Names
Each name begins with a 3-character prefix (IJF) and continues with a 5-character field containing 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 fixedlength (F,X) and undefined (U,N) records are mutually exclusive of each other and of all forms of the module that process variable-length records (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	
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

IOCS Subset/Superset Names in *The Macro System* chapter.

```
          * + + + +
    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.

DTFOR Macro

DTFOR is used to define an input file to be processed on a 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 2-17.

DTFOR is not used for the 3881 Optical Mark Reader. The 3881 uses DTFCD.

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={38 | n}
This operand indicates the size of the input area specified by IOAREA1. For journal tape processing, BLKSIZE specifies the maximum number of characters that can 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 is issued for a file. A CNTRL macro issues orders to the optical reader to perform nondata operations such as line marking, stacker selecting, document incrementing, etc.

Applies to

1285	1287T	1287D	1288			
x	x	x	x	M	COREXIT=xxxxxxxx	Name of your correction routine
x	x	x	x	M	DEVADDR=SYSnnn	Symbolic unit assigned to the optical reader
x	x	x	x	M	EOFADDR=xxxxxxxx	Name of your end-of-file routine
x	x	x	x	M	IOAREA1=xxxxxxxx	Name of first input area
x	x			O	BLKFAC=nn	If RECFORM=UNDEF in journal tape mode
x	x	x	x	O	BLKSIZE=nn	Length of I/O area(s). If omitted, 38 is assumed.
x	x	x	x	O	CONTROL=YES	If CNTRL macro is to be used for this file
x	x	x	x	O	DEVICE=xxxxx	(1287D or 1287T). For 1288, specify 1287D. If omitted, 1287D is assumed.
x	x	x		O	HEADER=YES	If a header record is to be read from the optical reader keyboard by OPEN, OPENR
		x	x	O	HPRMTY=YES	If hopper empty condition is to be returned.
x	x			O	IOAREA2=xxxxxxxx	If two input areas are used, name of second input area.
x	x			O	IOREG=(nn)	Reg. No. if 2 input areas or UNDEF records are to be used. If omitted, reg.2 is assumed. General registers 2-12, written in parentheses.
x	x	x	x	O	MODNAME=xxxxxxxx	Name of DTF's logic module. If omitted, IOCS generates a standard name.
x	x	x	x	O	RECFORM=xxxxxx	FIXBLK, FIXUNB, or UNDEF). If omitted, FIXUNB is assumed.
x	x	x	x	O	RECSIZE=(nn)	Reg. no. containing record size, if RECFORM=UNDEF. If omitted, reg. 3 is assumed.
x	x	x	x	O	SEPASMB=YES	If the DTFOR is to be assembled separately.
x	x			O	WORKA=YES	If records are to be processed in a work area. Omit IOREG.

M=Mandatory; O=Optional

Figure 2-17 **DTFOR macro options**

COREXIT=name
COREXIT provides an exit to your error correction routine for the 1287 or 1288. After a 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 your COREXIT routine. These should only be tested for nonrecovery (X'10') and (X'20') late stacker selection. For the 1287, a stacker select was given after the allotted elapsed time and the document was put in the reject pocket.
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 (1287T) or a hopper empty condition (see HPRMTY=YES operand) has occurred (1287D).
X'04'	A wrong-length record condition has occurred (for journal tapes, five read attempts were made; for documents, three read attempts were made). 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. You should test the RECSIZE register before moving records from the work area or the I/O area.
X'10'	A nonrecoverable error occurred.
X'40'	The 1287D scanner was unable to locate the reference mark (for journal tapes, ten read attempts were made; for documents, three read attempts were made).

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

If you issue I/O macros to any device other than the 1287 and/or 1288 in the COREXIT routine, you must save registers 0, 1, 14, and 15 upon entering the routine, and restore these registers before exiting. Furthermore, if I/O macros (other than the GET, WAITF, and/or READ, which cannot be used in COREXIT) are issued to the 1287 and/or 1288 in this routine, you 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 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 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, your routine must not branch to the address in register 14 from the COREXIT routine or a program loop will occur.

Following a nonrecoverable 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, your 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 routine must not branch to the address in register 14 from the COREXIT routine. If a nonrecoverable error occurs, the routine should ignore any output resulting from the document.

Eight binary error counters are used to accumulate totals of certain 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*, below).
2 filename+52	Equipment check uncorrectable after ten read attempts for journal tapes or three read attempts for documents (see *Note* below).
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 read attempts 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 executing 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. You may list the contents of these counters for analysis at end of file, or at end of job, or you may ignore the counters. The binary contents of the counters should be converted to a printable format.

DEVADDR=SYSnnn
This operand specifies the logical unit (SYSnnn) to be associated with the file. The logical 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={<u>1287D</u> | 1287T}
This operand specifies the I/O device associated with this file. 1287D specifies a 1287 or 1288 document file. 1287T specifies a 1287 journal tape file.

From this specification, IOCS sets up the device-dependent routines for this file. For document processing you must code the CCWs.

If this operand is omitted, 1287D is assumed.

EOFADDR=name
This operand specifies the name of your 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 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 your routine specified by EOFADDR. You must determine if the current roll is the last roll to be processed when handling journal tapes. Regardless of the situation, the tape file must be closed for each roll within your EOF routine. If the current roll is not the last, OPEN (or OPENR) must be issued. The OPEN (or OPENR) macro 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 (or OPENR) Macro* in the section *Imperative Macros* later in this chapter.

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

HPRMTY=YES
This operand is included if you want to be informed of the hopper empty condition. 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, your 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 *IBM 1287 Optical Reader Component Description and Operating Procedures*, GA21-9064). With this method of processing, the HPRMTY parameter allows you to check for a hopper empty condition in your COREXIT routine. You can then select into the proper hopper the previously ejected document before return from COREXIT (via register 14).

IOAREA1=name
This operand is included to specify the 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. The specified second I/O area is set to binary zeros before each input operation to this area occurs.

IOREG={(2) | (r)}
This operand specifies a general-purpose register (2-12) that the input routines use to indicate the beginning of records for a journal tape file. The same register may be specified in the IOREG operand for two or more files in the same program, if desired. In this case, your 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 macro 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 the leftmost position of the record.

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 (ORMOD) is assembled with the program, the MODNAME parameter in this DTF must specify the same name as the ORMOD macro.

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

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 | {(3) | (r)}
For fixed-length unblocked records, this operand should be omitted and no register is assumed.

For fixed-length blocked records (journal tape mode), this operand must be included to specify the number, n, of characters in an individual record. The input routines use this number 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 CCW chain which you supply. 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, you gain complete usage of the register normally used in the RECSIZE operand. You can do this by ensuring that the suppress-length-indication (SLI) flag is always on when processing undefined records.

SEPASMB=YES
Include this operand only if the DTFOR 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 problem program and no CATALR card is punched.

WORKA=YES
Input records (journal tape only) can be processed in work areas instead of in the input areas. If this is planned, the operand WORKA=YES must be specified,, and you must set up the work area in storage. The symbolic name of the work area, or a general-purpose 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 operand is included for a file, the DTFOR IOREG operand must be omitted.

1288 Optical Page Reader Programming Considerations

After a 1288 file is defined, the OPEN or OPENR macro makes it available for input. Processing is then accomplished by the CNTRL, READ, RESCN, and WAITF macros. When processing is completed, the CLOSE or CLOSER macro deactivates the file. 1288 processing adheres closely to the macros and DTF specifications used for 1287 document processing.

ORMOD Macro

Listed here are the operands you can specify for ORMOD. The first card contains ORMOD in the operation field and may contain a module name in the name field.

Note: ORMOD is not used for the 3881 Optical Mark Reader. The 3881 uses CDMOD.

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={<u>FIXUNB</u> | FIXBLK | UNDEF}
This operand generates a module that processes the specified record format. Any DTF used with the module must have the same operand.

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

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

DEVICE={1287D | 1287T}
This operand must be included to specify the I/O device associated with this file. 1287D specifies a 1287 or 1288 document file. 1287T specifies a 1287 journal tape file.

SEPASMB=YES
Include this operand only if the module is assembled separately. This causes a CATALR card with the module name (standard or user-specified) to be punched ahead of the object deck and defines the module name as an ENTRY point in the assembly. If the operand is omitted, the program assumes that the DTF is being assembled with the problem program and no CATALR card is punched.

WORKA=YES
Include this operand (journal tape only) if records are to be processed in work areas instead of in I/O areas. Any DTF used with the module must have the same operand.

Standard ORMOD Names

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 *IOCS Subset/Superset Names* in *The Macro System* chapter.

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

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

DTFPR Macro

Enter the symbolic name of the file in the name field and DTFPR in the operation field. The detail entries follow the DTFPR header card in any order. Figure 2-19 lists the keyword operands contained in the operand field.

ASOCFLE=filename
This operand is used together with the FUNC operand to define associated files for the 2560, 3525, or 5425. (For a description of associated files see the *DOS/VS Data Management Guide*, GC33-5372.) ASOCFLE specifies the filename of an associated read and/or punch file, and enables macro sequence checking by the logic module of each associated file. One filename is required per DTF for associated files.

Figure 2-18 defines the filename specified by the ASOCFLE operand for each of the associated DTFs.

	In ASOCFLE operand of ...		
FUNC=	read DTFCD, specify filename of	punch DTFCD, specify filename of	print DTFPR, specify filename of
RW	print DTFPR		read DTFCD
PW		print DTFPR	punch DTFCD
RPW	punch DTFCD	print DTFPR	read DTFCD

Figure 2-18 **ASOCFLE operand usage with print associated files**

For example, if FUNC=PW is specified, specify the filename of the print DTFPR in the ASOCFLE operand of the punch DTFCD, and specify the filename of the punch DTFCD in the print DTFPR. Or if FUNC=RPW is specified, specify the filename of the punch DTFCD in the ASOCFLE operand of the read DTFCD; specify the filename of the print DTFPR in the punch DTFCD; and specify the filename of the read DTFCD in the print DTFPR.

BLKSIZE= n
This operand specifies the length of IOAREA1. If the record format is variable or undefined, enter the length of the longest record. The maximum value which may be specified is:

- 151 for the 1403, 1443, 3203, or 3211
- 384 for the 2560
- 64 for the 3525
- 128 for the 5203 or 5425

If this entry is omitted:

- 121 is assumed for the 1403, 1443, 3203, or 3211
- 64 is assumed for the 2560 or 3525
- 96 is assumed for the 5203 or 5425

CONTROL=YES
This operand is specified if the CNTRL macro will be issued for the file. If this operand is specified, omit CTLCHR. This operand is not allowed for the 2560 or 5425.

CTLCHR={YES | ASA}
This operand is specified if firstcharacter-control is used. The parameter ASA specifies the American National Standards Institute, Inc. character set. The entry CTLCHR=YES specifies the System/370 character set. *Appendix A* contains the control character codes. If this parameter is specified, omit CONTROL. This operand must not be specified for the 2560 or 5425.

If CTLCHR=ASA is specified for the 3525, the + character is not allowed. When CTLCHR=ASA is specified for 3525 print (not associated) files, you must issue either a space 1 command or skip to channel 1 command to print on the first line of a card. For 3525 print associated files, you must issue a space 1 command to print on the first line of a card.

DEVADDR={SYSLOG | SYSLST | SYSnnn}
This operand specifies the symbolic unit to be associated with the printer. SYSLOG and SYSLST must not be specified for the 2560, 3525, or 5425.

DEVICE={<u>1403</u> | 1443 | 2560P | 2560S | 3203 |
3211 | 3525 | 5203 | 5425P | 5425S}
This operand specifies which device is used for the file. The "P" and "S" included with the "2560" and "5425" parameters specify primary or secondary input hoppers. If this operand is omitted 1403 is assumed.

ERROPT={RETRY | IGNORE | name}
This operand specifies the action to be taken in the case of an equipment check error. The functions of the parameters are described below.

RETRY can be specified only for the 3211. RETRY indicates that if an equipment check with command retry is encountered, the command is retried once. If the retry is unsuccessful a message is issued and the job canceled.

IGNORE can be specified only for the 3525. IGNORE indicates that the error is to be ignored. The address of the record in error is put in register 1 and made available for processing. Byte 3, bit 3 of the CCB is also set on (see Figure 6-2); you can check this bit and take the appropriate action to recover from the error. IGNORE must not be specified for files with two I/O areas or a work area.

The name parameter can be specified only for the 3211. It indicates that when an equipment check with command retry is encountered, the command is retried once. If the retry is unsuccessful a message is issued and the job canceled. With other types of errors (for these see the CCB, Figure 6-2) an error message is issued, error information is placed in the CCB, and control is given to your error routine, where you may perform whatever actions are desired. If any IOCS macros are issued in the routine, register 14 must be saved; if the operand RDONLY=YES is specified, register 13 must also be saved. To continue processing at the end of the routine, return to IOCS by branching to the address in register 14.

FUNC={W[T] | RW[T] | RPW[T] | PW[T]}
This operand specifies the type of file to be processed by the 2560, 3525, or 5425. W indicates print, R indicates read, P indicates punch, and T (for the 3525 only) indicates an optional 2-line printer.

RW[T], RPW[T], and PW[T] are used, together with the ASOCFLE operand, to specify associated files; when one of these parameters is specified for a printer file it must also be specified for the associated file(s).

If a 2-line printer is not specified for the 3525, multi-line print is assumed. T is ignored if CONTROL or CTLCHR is specified.

M	DEVADDR=SYSxxx	Symbolic unit for the printer used for this file.
M	IOAREA1=xxxxxxxx	Name for the first output area.
O	ASOCFLE=xxxxxxxx	Name of the associated file for FUNC=RW, RPW, PW.
O	BLKSIZE=nnn	Length of one output area, in bytes. If omitted, 121 is assumed for 1403, 1443, 3203 or 3211; 64 is assumed for 2560 or 3525, 96 is assumed for 5203 or 5425.
O	CONTROL=YES	CNTRL macro used for this file. Omit CTLCHR for this file. Not allowed for 2560 or 5425.
O	CTLCHR=xxx	(YES or ASA). Data records have control character. YES for S/370 character set; ASA for American National Standards Institute character set. Omit CONTROL for this file. Not allowed for 2560 or 5425.
O	DEVICE=nnn	(1403, 1443, 2560P, 2560S, 3203, 3211, 3525, 5203, 5425P, 5425S). If omitted, 1403 is assumed.
O	ERROPT=xxxxxxxx	RETRY or the name of your error routine for 3211. IGNORE for 3525. Not allowed for other devices.
O	FUNC=xxxx	(W, RW, RPW, PW) for 2560 or 5425. (W[T], RW[T], RPW[T], PW[T] for 3525.
O	IOAREA2=xxxxxxxx	If two output areas are used, name of second area.
O	IOREG=(nn)	Register number, if two output areas used and PUT does not specify a work area. Omit WORKA.
O	MODNAME=xxxxxxxx	Name of PRMOD logic module for this DTF. If omitted, IOCS generates standard name.
O	PRINTOV=YES	PRTOV macro used for this file. Not allowed for 2560 or 5425.
O	RDONLY=YES	Generate a read-only module. Requires a module save area for each task using the module.
O	RECFORM=xxxxxx	(FIXUNB, VARUNB, or UNDEF). If omitted, FIXUNB is assumed.
O	RECSIZE=(nn)	Register number if RECFORM=UNDEF.

M=Mandatory; O=Optional

Figure 2-19 DTFPR macro (part 1 of 2)

O	SEPASMB=YES	DTFPR is to be assembled separately.
O	STLIST=YES	1403 selective tape listing feature is to be used. Operand valid for DOS only.
O	UCS=xxx	(ON) process data checks, (OFF) ignores data checks. Only for printers with the UCS feature or 3211. If omitted, OFF is assumed.
O	WORKA=YES	PUT specifies work area. Omit IOREG.

M=Mandatory; O=Optional

Figure 2-19 DTFPR macro (part 2 of 2)

IOAREA1=name
This operand specifies the name of the output area.

IOAREA2=name
This operand specifies the name of a second output area.

IOREG=(r)
If two output areas and no work areas are used, this operand specifies the address of the area where you can build a record. For (r) specify one of the registers 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, MODNAME must specify the same name as the PRMOD macro. If this operand is omitted, standard names are generated for calling the logic module. If two DTF macros 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 is included in your program. This operand is not allowed for the 2560 or 5425.

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 task requires its own uniquely defined save area. Each time an imperative macro (except OPEN or OPENR) is issued, register 13 must contain the address of the save area associated with the task. 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* in the *Multitasking Macros* chapter.

If an ERROPT routine issues I/O macros which use the same read-only module that caused control to pass to either error routine, your 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 routine. Before returning to the module that entered the ERROPT routine, register 13 must be set to the save area address originally specified for the task.

If this operand is omitted, the module generated is not reenterable and no save area need be established.

RECFORM={<u>FIXUNB</u> | 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 of undefined format. The length of each record must be loaded into the register before issuing the PUT macro.

SEPASMB=YES
Include this operand only if the DTFPR 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 as-

sumes that the DTF is being assembled with the problem program and no CATALR card is punched.

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 will be ignored if specified. If this operand is specified, RECFORM must have the parameter FIXUNB.

UCS={OFF | ON}
For a 1403 or 5203 printer with the universal character set feature or for a 3203 or a 3211, this operand determines whether data checks occurring in case of unprintable characters are indicated to the operator, printed as blanks, or ignored. The entry is especially useful if you are using first-character forms control and 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 work areas instead of in the I/O areas, specify this operand. You must set up the work area in storage. The address of the work area, or a general-purpose register which contains the address, must be specified in each PUT macro.

PRMOD Macro

Listed here are the operands you can specify for PRMOD. The first card contains PRMOD in the operation field and may contain a module name in the name field.

CONTROL=YES
Include this operand if CNTRL macros are used with the associated DTFs. The module also processes files that do not use the CNTRL macro. If CONTROL is specified, the CTLCHR operand must not be specified.

The CONTROL operand is not allowed for the 2560 or 5425.

CTLCHR={YES | ASA}
Include this operand if first-character carriage control is used. Any DTF used with the module must have the same operand. If CTLCHR is specified, CONTROL must not be specified.

CTLCHR must not be specified for the 2560 or 5425. If CTLCHR=ASA is specified for the 3525, the + character is not allowed; when CTLCHR=ASA is specified for 3525 print (not associated) files, you must issue either a space 1 command or skip to channel 1 command to print on the first line of a card. For 3525 print associated files, you must issue a space 1 command to print on the first line of a card. If CTLCHR=ASA and RDONLY=YES are specified in a multitasking environment where more than one DTFPR uses the same module, overprinting may occur.

DEVICE={1403 | 1443 | 2560P | 2560S | 3203 | 3211 | 3525 | 5203 | 5425P | 5425S}
This operand specifies which device is used for the file. The "P" and "S" included with the "2560" and "5425" parameters specify primary or secondary input hoppers; regardless of which is specified, however, the module generated will handle DTFs specifying either hopper.

Any DTF to be used with this module must have the same operand (except as just noted concerning the "P" and "S" specification for the 2560 or 5425).

ERROPT=YES
This operand must be specified if ERROPT=name is specified in a DTFPR to be used with the module. (ERROPT=name is applicable to the 3211 only.) If ERROPT is not specified in the DTFPR, or if ERROPT=RETRY (3211) or ERROPT=IGNORE (3525) is specified, ERROPT=YES must be omitted.

FUNC={W[T] | RW[T] | RPW[T] | PW[T]}
This operand specifies the type of file to be processed by the 2560, 3525, or 5425. Any DTF used with the module must include the same operand. W indicates print, R indicates read, P indicates punch, and T (for the 3525 only) indicates an optional 2-line printer.

RW[T], RPW[T], and PW[T] are used to specify associated files; when one of these parameters is specified for a printer file it must also be specified for the associated file(s).

If a 2-line printer is not specified for the 3525, multi-line print is assumed. T is ignored if CONTROL or CTLCHR is specified.

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 macros are used with the associated DTFs. The module also processes any files that do not use the PRTOV macro.

This operand is not allowed for the 2560 or 5425.

RDONLY=YES
This operand causes a read-only module to be generated. Whenever this operand is specified, any DTF used with the module must have the same operand.

RECFORM={FIXUNB | VARUNB | UNDEF}
This operand causes a module to be generated that processes the specified record format: fixed-length, variable-length, or undefined. Any DTF used with the module must include the same operand.

SEPASMB=YES
Include this operand only if the module is assembled separately. This causes a CATALR card with the module name (standard or user-specified) to be punched ahead of the object deck and defines the module name as an ENTRY point in the assembly. If the operand is omitted, the program assumes that the DTF is being assembled with the problem program and no CATALR card is punched.

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. If this operand is specified, RECFORM must have the parameter FIXUNB.

WORKA=YES
Include this operand if records are processed in work areas instead of in I/O areas. Any DTF used with the module must have the same operand.

Standard PRMOD Names
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	none of these is specified
	= T	DEVICE=3525 with 2-line printer
	= U	DEVICE=2560
	= V	DEVICE=5425
c	= B	ERROPT=YES and PRINTOV=YES
	= P	PRINTOV=YES, DEVICE is not 3525, and ERROPT is not specified
	= I	PRINTOV=YES, DEVICE=3525, and FUNC=W[T] or omitted
	= F	PRINTOV=YES, DEVICE=3525, and FUNC=RW[T]
	= C	PRINTOV=YES, DEVICE=3525, and FUNC=PW[T]
	= D	PRINTOV=YES, DEVICE=3525, and FUNC=RPW[T]
	= Z	PRINTOV=YES and ERROPT are not specified and DEVICE is not 2560, 3525, or 5425
	= O	PRINTOV=YES not specified, DEVICE=3525, and FUNC=W[T] or omitted
	= R	PRINTOV=YES not specified, DEVICE=3525, and FUNC=RW[T]
	= S	PRINTOV=YES not specified, DEVICE=3525, and FUNC=PW[T]
	= T	PRINTOV=YES not specified, DEVICE=3525, and FUNC=RPW[T]
	= E	ERROPT=YES and PRINTOV=YES is not specified
	= U	FUNC=W or omitted and DEVICE=2560 or 5425
	= V	FUNC=RW and DEVICE=2560 or 5425
	= W	FUNC=PW and DEVICE=2560 or 5425
	= X	FUNC=RPW and DEVICE=2560 or 5425
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. No subsetting/supersetting of PRMOD names is allowed for the 2560 or 5425. See *IOCS Subset/Superset Names* in *The Macro System* chapter.

```
      * * * * *
I J D F A U I V
      V Y V Z W
      U S W   Y
        T X   Z
        U +
        V P
        + Z
        C +
        Z I
          O
          +
          F
          R
          +
          C
          S
          +
          D
          T
          +
          B
          E
```

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

DTFPT Macro

A DTF entry is included for every paper tape input or output file that is processed by the program. The characteristics of a paper tape file are given in the *DOS/VS Data Management Guide*, GC33-5372.

The first entry must be the DTFPT header entry. Enter the symbolic name of the file in the name field, and DTFPT in the operation field. The detail entries follow the DTFPT header card in any order. Figure 2-20 lists the keyword operands contained in the operand field.

BLKSIZE=n
This operand specifies the length of the input or output area. The maximum block size is 32,767 bytes.

Input: For undefined records, this area must be at least one byte larger than the longest record including all shift and delete characters included in the 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 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.

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 you 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 1018 has the error correction feature.

DEVADDR=SYSnnn
This operand specifies the logical unit (SYSnnn) associated with this file. An actual channel and unit are assigned to the unit by an ASSGN 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.

Applies to

Input	Output			
X	X	M	BLKSIZE=n	Length of your I/O areas.
X	X	M	DEVADDR=SYSnnn	Symbolic unit to be associated with this file.
X	X	M	IOAREA1=xxxxxxxx	Name of first I/O area.
X		O	EOFADDR=xxxxxxxx	Name of your end-of-file routine.
	X	O	DELCHAR=x'nn'	Delete character.
X	X	O	DEVICE=nnnn	(2671, 1017, 1018). If omitted, 2671 is assumed.
	X	O	EORCHAR=x'nn'	End-of-record character. (For RECFORM=UNDEF).
X	X	O	ERROPT=xxxxxxxx	(IGNORE, SKIP, or error routine name). Prevents job termination on error records.
	X	O	FSCAN=xxxxxxxx	(For shifted codes). Name of your scan table used to select figure groups.
X		O	FTRANS=xxxxxxxx	(For shifted codes). Symbolic address of your figure shift translate table.
X	X	O	IOAREA2=xxxxxxxx	Name of second I/O area.
X	X	O	IOREG=(nn)	Used with two I/O areas. Register (2-12) containing current record address.
	X	O	LSCAN=xxxxxxxx	(For shifted codes). Name of your scan table used to select letter groups.
X		O	LTRANS=xxxxxxxx	(For shifted codes). Name of your letter shift translate table.
X	X	O	MODNAME=xxxxxxxx	For module names other than standard.
X	X	O	OVBLKSZ=n	Used if I/O records are compressed or expanded.
X	X	O	RECFORM=xxxxxx	(FIXUNB or UNDEF). If omitted, FIXUNB is assumed.

M=Mandatory; O=Optional

Figure 2-20 DTFPT macro operands (part 1 of 2)

Applies to

Input	Output			
x	x	O	RECSIZE=(nn)	Register containing the record length.
x		O	SCAN=xxxxxxxx	Name of your scan table for shift or delete character.
x	x	O	SEPASMB=YES	DTF is assembled separately.
x	x	O	TRANS=xxxxxxxx	Name of your table for code translation.
x		O	WLRERR=xxxxxxxx	Name of wrong-length-record error routine.

M=Mandatory; O=Optional

Figure 2-20 **DTFPT macro operands (part 2 of 2)**

EOFADDR=name
This operand specifies the name of your 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 routine can execute any operation required for the end-of-file, issue the CLOSE or CLOSER macro 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 1018.

EORCHAR=X'nn'
This operand specifies the user-defined end-of-record (EOR) character, where nn is two hexadecimal digits. It 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 you do 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, IGNORE allows IOCS to handle the record as if no errors were detected. If SKIP 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. 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 or CLOSER macro checks the last record, and the ERROPT=name option is treated as the ERROPT=IGNORE option.

If IGNORE and SKIP are not specified, the name of your 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 error routine. At the end of the error routine, return to IOCS by branching to the address in register 14. The next record is then read or written. You must not issue any GET or PUT macros 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. For an input file, omit this operand. It specifies the name of a scan table in your program used to select groups of figures. This table must conform to the specifications of the machine instruction TRT. 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.

FTRANS=name
This operand must be included for every input file using a shifted code and is not permitted for output files. It specifies the name of a figure shift table in

your program. This table must conform to the specifications of the machine instruction TR.

IOAREA1=name
This operand specifies the name of an input or output area.

IOAREA2=name
This operand specifies the name of a second input or output area. When this operand is 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 your 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 and is not permitted for input files. It specifies the name of a scan table in your program used to select groups of letters. This table must conform to the specifications of the machine instruction TRT. 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.

LTRANS=name
This operand must be included for every input file using a shifted code and is not permitted for output files. It specifies the name of a letter shift table in your program. This table must conform to the specifications of the machine instruction TR.

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 this DTF must specify the same name as the PTMOD macro. 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 (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.

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 from a PUT macro. OVBLKSZ is used only when LSCAN and FSCAN are specified with the FIXUNB format. If OVBLKSZ is specified 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. 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 files. 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 into the specified register.

For output files, this entry must be included for undefined records. Before translation, your program must load each record length into the designated register before issuing the PUT macro for the record.

SCAN=name
This operand must be included for all input files using shifted codes. It may also be included if you wish to delete certain characters from each record. The SCAN entry specifies the symbolic name of a table provided by your program. This table must conform to the specifications of the machine instruction TRT. It must contain 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 hexadecimal 08 for the letter shift character. Delete entries must be hexadecimal 0C. All other entries in the table must be hexadecimal 00. Other-

wise, incorrect translation results and may produce a program check.

The table must be large enough to hold the maximum 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
Include this operand only if the DTFPT 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 problem program and no CATALR card is punched.

TRANS=name
The TRANS operand specifies the symbolic name of a table provided within your program. This table must conform to the specifications of the machine instruction TR. For input files, include this entry if a nonshifted code is to be translated into internal System/370 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 System/370 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 omitted 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, return to IOCS by branching to the address in register 14. The next IOCS read operation will normally cause the remainder of the overlength, undefined record to be read. If any other IOCS macros are included in the record-length error routines, the contents of register 14 must be saved and restored.

Note: A wrong-length condition appears during the first read operation on a 1017 if the combined length of the tape leader and the first record is greater than the length of the longest record anticipated (the length specified in BLKSIZE).

Paper Tape Processing Considerations

EOF Condition (Input Only)

The EOF condition occurs with an end-of-tape condition when the EOF switch is on. When IOCS detects this EOF condition (unit exception flag on in first CSW status byte), it automatically branches to your end-of-file routine. However, at the end of the routine, you 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 1017, and causes the broken tape bit (bit 7) to appear in the sense byte. The broken tape condition may occur in addition to the EOF 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):

$$\frac{\text{Byte Count } + 2 \text{ inches}}{10}$$

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, and a trailer greater than

$$\frac{\text{BLKSIZE } + 2 \text{ inches}}{10}$$

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 (for input records).
- Translated, expanded, and punched, before taking the error option specified by the problem program (for output records).

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 operand. 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 REC-SIZE operand 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 irrecoverable 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
- retry the operation (for 2671 only)

Following an ignore decision, logical IOCS takes action in accordance with the parameter specified in ERROPT.

ERROPT=IGNORE	The record is handled as if no errors were detected.
ERROPT=SKIP	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 your error routine. At the end of this routine, return to IOCS by branching to the address in register 14. The next record is then 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 irrecoverable error occurs and ERROPT=name or ERROPT=IGNORE was specified in the DTFPT.
- In the case of output records with two I/O areas, the CLOSE or CLOSER macro checks the successful completion of the last operation.

Note 3: No error condition occurs on the 1018 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, restore the CCW address in the CCB before using the EXCP macro.

Programming Considerations
For information about special equipment considerations for paper tape devices, refer to *IBM 2671 Paper Tape Reader and IBM 2822 Paper Tape Reader Control*, GA24-3388, and *IBM System/360 Component Descriptions: 2826 Paper Tape Control Unit, 1017 Paper Tape Reader, 1018 Paper Tape Punch*, GA33-4500.

PTMOD Macro

Listed here are the operands you can specify for PTMOD. The first card contains PTMOD in the operation field and may contain a module name in the name field.

DEVICE={<u>2671</u> | 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. 2671 is assumed if this operand is omitted.

RECFORM={<u>FIXUNB</u> | UNDEF}
Required only if the operand 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
Include this operand only if the module is assembled separately. This causes a CATALR card with the module name (standard or user-specified) to be punched ahead of the object deck and defines the module name as an ENTRY point in the assembly. If the operand is omitted, the program assumes that the DTF is being assembled with the problem program and no CATALR card is punched.

TRANS=YES
Required only if records using an unshifted code are translated and if the operand SCAN=YES is not specified.

Summary of PTMOD
Figure 2-21 shows the only possible combinations of PTMOD operands and describes the resultant modules.

Standard PTMOD Names
Each name begins with a 3-character prefix (IJE) and continues with a 5-character field corresponding to the options permitted in the generation of the module.

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

Subset/Superset PTMOD Names
The following chart shows the PTMOD names. No subsetting or supersetting is allowed. See *IOCS Subset/Superset Names* in *The Macro System* chapter.

```
          *  *  *  *
    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.

Operand*				Resulting Module
DEVICE	RECFORM	SCAN	TRANS	
-				Does not handle translation or shift or delete characters
2671				
-			YES	Handles translation of unshifted codes, but not delete characters
2671			YES	
-		YES		Handles shift and delete characters for records of fixed unblocked format
-	FIXUNB	YES		
2671		YES		
2671	FIXUNB	YES		
-	UNDEF	YES		Handles shift and delete characters for records of undefined format
2671	UNDEF	YES		
1017				Does not handle translation or shift or delete characters
1017			YES	Handles translation of unshifted codes, but no delete characters.
1017		YES		Handles shift and delete characters for records of fixed unblocked format
1017	FIXUNB	YES		
1017	UNDEF	YES		Handles shift and delete characters for records of undefined format
1018				Handles translation of unshifted codes, if specified in DTFPT, for records of fixed unblocked format
1018			YES	
1018	FIXUNB			
1018	FIXUNB		YES	
1018	UNDEF			Handles translation of unshifted codes, if specified in DTFPT, for records of undefined format
1018	UNDEF		YES	
1018		YES		Handles shift characters for records of fixed unblocked format
1018	FIXUNB		YES	
1018	UNDEF	YES		Handles shift characters for records of undefined format

* In all cases, SEPASMB=YES may either be specified or omitted.

Figure 2-21 PTMOD operand combinations

DTFSD Macro

The DTFSD macro defines sequential (consecutive) processing for a file contained on a DASD. Only IBM standard label formats are processed.

A DTFSD entry is included for each sequential input or output DASD file that is processed in the program. The DTFSD header entry and a series of detail entries describe the file. Enter the symbolic name of the file in the name field and DTFSD in the operation field. The detail entries follow the DTFSD header card in any order. Figure 2-22 lists the keyword operands contained in the operand field.

BLKSIZE=n
Enter the length 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 size of the I/O area must be at least as large as the smaller of the values shown in Figure 2-25. For output files, the first 8 bytes of IOAREA1 must be alloted for IOCS to construct a count field.

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 or CLOSER macro is not to delete the Format-1 and Format-3 label for a work file. The operand applies to work files only.

DEVADDR=SYSnnn
This operand must specify the symbolic unit associated with the file if an extent is not provided. An EXTENT job control statement is not required for single-volume input files. If an EXTENT job control statement is provided, its specification overrides any DEVADDR specification. SYSnnn represents an actual I/O address, and is used in the ASSGN job control 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 of *The Macro System* chapter. The only symbolic unit within this list which is not applicable is SYSLOG.

DEVICE={<u>2311</u> | 2314 | 2321 | 3330 | 3340}
This operand is included to specify the device on which the file is located. If no device is specified, 2311 is assumed. Specify 2314 for 2319 and 3330 for 3333.

EOFADDR=name
This operand specifies the name of your end-of-file routine. IOCS automatically branches to this routine on an end-of-file condition. You can perform any operations required for the end of the file in this routine (you generally issue the CLOSEor CLOSER macro).

ERREXT=YES
This operand enables your 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 your 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 a job is not to be 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 operand is specified. Either IGNORE, SKIP, or the name of an error routine can be specified. The functions of these parameters are described below.

IGNORE	The error condition is ignored. The records are made available for processing. When reading spanned records, the whole spanned record or block of spanned records is returned, 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.

Applies to					
Input	Output	Work			
x	x	x	M	BLKSIZE=nnnn	Length of one I/O area, in bytes
x		x	M	EOFADDR=xxxxxxxx	Name of your end-of-file routine
x	x		M	IOAREA1=xxxxxxxx	Name of first I/O area
x	x	x	O	CONTROL=YES	CNTRL macro used for this file
		x	O	DELETFL=NO	CLOSE, CLOSER macro is not to delete Format-1 and Format-3 labels for work file
x	x	x	O	DEVADDR=SYSnnn	Symbolic unit required only when not provided on an EXTENT statement
x	x	x	O	DEVICE=nnnn	(2311, 2314, 2321, 3330, 3340). If omitted, 2311 is assumed
x	x	x	O	ERREXT=YES	Additional error and ERET are desired. Specify ERROPT
x	x	x	O	ERROPT=xxxxxxxx	(IGNORE, SKIP, or name of error routine). Prevents job termination on error records. Do not use SKIP for output files
x	x		O	FEOVD=YES	Forced end of volume for disk is desired
x		x	O	HOLD=YES	Employ the track hold function
x	x		O	IOAREA2=xxxxxxxx	If two I/O areas are used, name of second area
x	x		O	IOREG=(nn)	Register number. Use only if GET or PUT does not specify work area or if two I/O areas are used. Omit WORK A
x	x		O	LABADDR=xxxxxxxx	Name of your routine to check/write user-standard labels
x	x		O	MODNAME=xxxxxxxx	Name of SDMODxx logic module for this DTF. If omitted, IOCS generates standard name
		x	O	NOTEPNT=xxxxxxx	(YES or POINTRW). YES if NOTE/POINTR/POINTW/POINTS used. POINTRW if only NOTE/POINTR/POINTW used
x	x	x	O	RDONLY=YES	Generates a read-only module. Requires a module save area for each task using the module
x	x	x	O	RECFORM=xxxxxx	(FIXUNB, FIXBLK, VARUNB, VARBLK, SPNUNB, SPNBLK, or UNDEF). For work files use FIXUNB or UNDEF. If omitted, FIXUNB is assumed

M=Mandatory; O=Optional

Figure 2-22 DTFSD macro operands (part 1 of 2)

Applies to

Input	Output	Work			
x	x		O	RECSIZE=nnnnn	If RECFORM=FIXBLK, number of characters in record. If RECFORM=SPNUNB, SPNBLK, or UNDEF, register number. Not required for other records
x	x		O	SEPASMB=YES	DTFSD is to be assembled separately.
x	x		O	TRUNCS=YES	RECFORM=FIXBLK or TRUNC macro used for this file
x	x	x	O	TYPEFLE=xxxxxx	(INPUT, OUTPUT, or WORK). If omitted, INPUT is assumed
x		x	O	UPDATE=YES	Input file or work file is to be updated
	x		O	VARBLD=(nn)	Register number if RECFORM=VARBLK and records are built in the output area. Omit if WORKA=YES
	x	x	O	VERIFY=YES	Check disk records after they are written. For DEVICE=2321, YES is assumed
x			O	WLRERR=xxxxxxxx	Name of your wrong-length-record routine
x	x		O	WORKA=YES	GET or PUT specifies work area. Omit IOREG. Required for RECFORM=SPNUNB or SPNBLK.

M=Mandatory; O=Optional

Figure 2-22 DTFSD macro operands (part 2 of 2)

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 an UPDATE=YES file, 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 your error routine named by this parameter regardless of whether or not ERREXT=YES is specified. In this routine you 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, 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 work area (if either is specified) are variable and therefore should not be used for error block processing. Also, GET macros must not be issued 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, 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 program.

A sequence error may occur if LIOCS is searching for the first segment of a logical spanned record and fails to 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 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 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 ERET 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 2-23. The figure is divided into two parts: the lower part lists the error conditions which you specify in the DTF, and the upper part shows the action resulting from these specifications when an error occurs. For example, assume that WLRERR=name is specified and the ERET macro is used with the RETRY option. The upper part of the table then shows that the job is terminated regardless of whether or not the error was due to a wrong-length record.

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 is issued, the current volume is closed, and I/O processing continues on the next volume.

HOLD=YES
This operand may be specified only if the track hold function was specified at system generation time and if it is employed when a data file or a work file is referenced for updating. See *DASD Track Protection Macros* in the *Multitasking Macros* chapter 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 variable-length or undefined records, this area must be large enough to contain the largest block or record. 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.

When processing spanned records, the size of the I/O area must be at least as large as the smaller of the values shown in Figure 2-24.

Desired Functions → / Specifications required in your Program ↓	Error other than Wrong Length Records: Job is terminated	Error other than Wrong Length Records: Error record is retried	Error other than Wrong Length Records: Error record is ignored	Error other than Wrong Length Records: Error record is skipped	Error other than Wrong Length Records: Control is passed to your error option routine		Wrong Length Record Errors: Job is terminated	Wrong Length Record Errors: Error record is ignored	Wrong Length Record Errors: Error record is skipped	Wrong Length Record Errors: Control is passed to your wrong length record
ERROPT = IGNORE			X					X		
ERROPT = name				X	X				X	X
ERROPT = SKIP				X					X	
WLRERR = name	X								X	X
ERROPT = IGNORE, WLRERR = name			X						X	X
ERROPT = name , WLRERR = name				X	X				X	X
ERROPT = SKIP, WLRERR = name				X					X	X
ERROPT = name ERET IGNORE			X					X		
ERROPT = name RERET RETRY		X					X			
ERROPT = name ERET SKIP			X[2]	X[1]	X[1]			X[2]	X[1]	X[1]
WLRERR = name ERET IGNORE	X							X		
WLRERR = name ERET RETRY	X						X			
WLRERR = name ERET SKIP	X								X	X
ERROPT = name , WLRERR = name ERET IGNORE			X					X		
ERROPT = name, WLRERR = name ERET RETRY		X					X			
ERROPT = name, WLRERR = name ERET SKIP			X[2]	X[1]	X[1]				X	X
NONE	X						X			

ERET Macro Options:
IGNORE
RETRY
SKIP

DTF Parameters:
ERROPT = name
ERROPT = IGNORE
ERROPT = SKIP
WLRERR = name

1 Input files only
2 Output files only
3 Record length not checked for DTFSD undefined records

Figure 2-23 **DTFSD error options**

Device	Length (Decimal)
2311 Disk Drive	3625* or BLKSIZE
2314, 2319 Disk Drive	7294* or BLKSIZE
2321 Data Cell	2000* or BLKSIZE
3330 and 3333 Disk Storage	13, 030* or BLKSIZE
3340 Disk Storage	8368* or BLKSIZE
* Add 8 for output files	

Figure 2-24 **I/O area requires when processing spanned records**

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 in the register specified the address of the area where you 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 program must store the address supplied by IOCS for each record.

This operand 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 name of the routine that enables you to process your own labels. See the sections *Writing User Standard Labels on Disk* and *Checking User Standard Labels on Disk* in the *Label Processing* chapter for a discussion of what the LABADDR routine should do.

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, MODNAME must specify the same name as the SDMODxx macro.

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

NOTEPNT={POINTRW | YES}
The parameter POINTRW is specified if a NOTE, POINTR, or POINTW macro is issued for the file. If the parameter YES is specified, NOTE, POINTR, POINTW, and POINTS macros may be issued for 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 task should have its own uniquely defined save area. When an imperative macro (except OPEN, OPENR, or LBRET) is issued, register 13 must contain the address of the save area associated with the task. 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* in the *Multitasking Macros* chapter.

If an ERROPT or WLRERR routine issues I/O macros using the same read-only module that caused control to pass to either error routine, your 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 error routine, register 13 must be set to the save area address originally specified for the task.

If the operand is omitted, the module generated is not reenterable and no save area need be established.

RECFORM={<u>FIXUNB</u> | FIXBLK | VARUNB | VARBLK | SPNUNB | SPNBLK | UNDEF}
This operand specifies the type of records for input or output. Enter one of the following parameters:

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 is specified and RECSIZE=(r) is not specified, an assembler diagnostic (MNOTE) is issued, and register 2 is assumed. If WORKA=YES is 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 in each record.

When processing spanned records, you 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, you must load the length of each record into the designated register before issuing a PUT macro. If specified for input, IOCS provides the length of the record transferred to virtual storage.

SEPASMB=YES
Include this operand only if the DTFSD 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 problem program and no CATALR card is punched.

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 for Tape and Disk* in the *Processing Macros* section later in this chapter.) If INPUT/OUTPUT is specified, the GET/PUT macros can be used. If WORK is specified, the READ/WRITE, NOTE/POINT, and CHECK macros can be 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. 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 you in the designated register after the PUT macro is issued for a variable-length record. You then compare the length of your 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, you issue a TRUNC macro 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 you want 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 a disk are not verified.

WLRERR=name
This operand applies only to disk input files. It does not apply to undefined records. WLRERR specifies the symbolic name of your 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.

Your WLRERR routine performs any processing desired for wrong-length records. However, GET macros 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, return 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 results:

- If the ERROPT entry is included for this file, the wrong-length record is treated as an error block and handled according to your specifications for an error (IGNORE, SKIP, or name of error routine).

- If the ERROPT entry is not included, the error is ignored.

Undefined records are not checked for incorrect record length. The record is truncated when the BLKSIZE specification is exceeded.

WORKA=YES
If I/O records are processed, or built, in work areas instead of in the I/O areas, specify this operand. You must set up the work area in storage. The address of the work area, or a general-purpose register which contains the address, must be specified in each GET or PUT macro. For a GET or PUT macro, IOCS moves the record to, or from, the specified work area. WORKA=YES is required for SPNUNB and SPNBLK. When this operand is specified for a file, the IOREG operand must be omitted.

SDMODxx Macro

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 associated with it (see Figure 2-25).

Macro	**Module Generated**
SDMODFI	Sequential DASD Module, Fixed-length records[1], Input file
SDMODFC	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
SDMODVC	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

Figure 2-25 SDMOD macros

The macro operation code and the keyword operands define the characteristics of the module.

Modules for a specific file can thus be generated more quickly than if there were only one macro.

The operands for the ten macros are shown in Figure 2-26 and explained in the following section.

SDMODxx Operands
A module name may be contained in the name field of the macro. The macro operation code 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 for the file. This entry applies to all SDMODxx macros. 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 an ERROPT routine in your program or if the ERET macro is used with the DTF and module. If ERREXT is specified ERROPT must also be specified.

ERROPT=YES
This operand applies to all SDMODxx macros. It 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 name) regardless of which option is specified in the DTF. The module also processes any DTF in which the ERROPT operand is not specified.

If this operand is not included, your program is canceled whenever any uncorrectable error except a 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. Any DTF used with the module must have the same operand.

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 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. It is included if any NOTE, POINTR, POINTS, or POINTW macros 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 causes a read-only module to be generated. Whenever this operand is specified, any DTF used with the module must have the same 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
Include this operand only if the module is assembled separately. This causes a CATALR card with the module name (standard or user-specified) to be punched ahead of the object deck and defines the module name as an ENTRY point in the assembly. If the operand is omitted, the program assumes that the DTF is being assembled with the problem program and no CATALR card is punched.

TRUNCS=YES
This operand applies to all SDMOD macros for fixed-length records. It generates a logic module which can handle the TRUNC macro. This operand is assumed for VARBLK output files. This operand is ignored if specified for VARBLK input or update 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 which can handle the WRITE UPDATE macro with work files.

Operand	Required	Comments
CONTROL=YES	If the CNTRL macro is to be issued for 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 for 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 for the file.	This parameter applies to SDMODW only. The operand POINTW 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 us to be issued for the file. Assumed for output files consisting of variable-length blocked records.	Applies to all SDMODs for fixed-length records.
UPDATE=YES	If SDMODW is to process the WRITE UPDATE macro.	Applies to SDMODW only.

Figure 2-26 **SDMODxx operands**

Standard SDMOD Names
Each name begins with a 3-character prefix (IJG) and continues with of a 5-character field corresponding to the options permitted in the generation of the module.

In SDMOD there are two module classes:

- Those which handle GET/PUT functions
- Those which handle 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 *IOCS Subset/Superset Names* in *The Macro System* chapter.

```
      + * + + +
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
```

DTFSR Macro

The DTFSR macro is provided only for the convience of those who are converting to DOS/VS from the Basic Operating System/360. If you are not a former Basic Operating System/360 user, there is no reason to use the DTFSR macro. Use the DTFCD, DTFCN, DTFMT, DTFOR, DTFPR, DTFPT, and DTFSD macros instead, as they are easier to use, are more flexible, and take full advantage of the features of DOS/VS.

For those using DTFSR, enter the symbolic name of the file in the name field and DTFSR in the operation field. A *begin-definition* card follows, with DTFBG punched in the operation field and DISK in the operand field. The name field is blank.

Detail entries follow the DTFBG card in any order. Figure 2-27 lists the keyword operands contained in the operand field.

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 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 operand is not used with RECFORM=FIXBLK, because the blocking factor is determined from the BLKSIZE and RECSIZE operands. If the BLKFAC 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 of characters that may be transferred to or from the area at 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 operand is not specified. If the BLKFAC operand 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, the CTLCHR operand is specified) and/or record-length fields for variable-length records (RECFORM=VARUNB), the BLKSIZE value must include the extra bytes allotted in the 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/VS.

CKPTREC=YES
This operand is required if an input tape 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 is specified if a CNTRL macro is to be issued for a file. If this operand is specified, CTLCHR must be omitted.

COREXIT=name
COREXIT provides an exit to your error correction routine for the 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 at the address filename+80. Filename+80 contains the following hexadecimal values 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 was detected. Normally, on an EOP indication, the program ejects and stacker-selects the document.
X'01'	A data check has occured. 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 (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. Test the RECSIZE register before moving records from the I/O or work area(s). The test conditions are:

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 should be checked to determine the reason for entry into the error correction routine. You may then perform whatever action is appropriate to recover from the error.

If you issue I/O macros to any device other than the 1287 and/or 1288 in the COREXIT routine, save registers 0, 1, 14, and 15 upon entering the routine, and restore these registers before exiting. Also, if I/O macros (other than GET and/or READ, which cannot be used in COREXIT) are issued to the 1287 and/or 1288 in this routine, you must first save, and later restore registers 14 and 15. All exits 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: Do not issue a GET or a READ macro to the 1287 or 1288 in the error correction routine. Do not process records in this routine. The record that caused the exit to the error routine is available for processing upon return to the main program. Any processing attempted in the error routine would be duplicated after return to the main program.

When processing journal tapes, a nonrecoverable error (torn tape, tape jam, etc.) normally requires the tape to be completely reprocessed. In this case, your routine must not branch to the address in register 14 from the COREXIT routine or a program loop will result. Following a nonrecoverable 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, your 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, you must not branch to the address in register 14 from the COREXIT routine. You should ignore any output resulting from the document under any circumstances.

Eight binary counters are used to accumulate totals of certain 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:

1 filename+48	Incomplete read (eqipment check)
2 filename+52	Incomplete read uncorrectable after ten read attenpts 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. You may list the contents of these counters for analysis at end of file, or at end of job, or you may ignore the counters. The 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 2520 or 2540. 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 you specified. If the CRDERR operand is specified, 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 virtual 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 ASSGN job control 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* in *The Macro System* chapter. 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 job control statement), 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 an EXTENT job control statement.

DEVICE=
This operand must be included to state the I/O device associated with this file. Enter one of the following:

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, 1443 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 console printer keyboard or the display operator console.
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.

This operand causes IOCS to set up the device-dependent routines for a file. For document processing on the 1287 or 1288 optical reader, or 1288 optical page reader, you have to code your own CCWs.

If this operand is omitted, 1287D is assumed.

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 your 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 the characters /* 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 the characters /* 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. You must determine, in your 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 you supplied.

- **Optical Reader Input**. When reading data from documents on a 1287 or 1288, end-of-file 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, 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 your routine specified by EOFADDR. If journal tapes are being processed, it is your responsibility to determine if the current roll is the last roll to be processed. 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 or OPENR macro must be issued to allow 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 in *OPEN and OPENR Macros* in the *Imperative Macros* section later in this chapter.

ERROPT={IGNORE | SKIP | name}
This operand applies to disk or magnetic tape input files. 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 name of an error routine can be specified in this entry. The functions of these three parameters are:

IGNORE	The error condition is completely ignored, and the records are made available 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 your routine, where you may perform whatever functions you desire 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 your error routine, 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. Do not issue any GET macros for records in the error block. If you use any other IOCS macros in your routine, save and later restore the contents of register 14. At the end of the routine 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 your program.

The ERROPT entry does not apply to disk or tape output files. The job is automatically terminated if a parity error still existed after IOCS attempted 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 operand 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 specifies what type of labels are to be processed. Enter one of the following parameters:

NO	For a tape that does not contain labels. The entry FILABL=NO may be omitted, if desired, and IOCS assumes that there are no labels.
STD	For tape input if standard labels are checked by IOCS, or for tape output if standard labels are written by IOCS.
NSTD	For tape input or output with nonstandard labels. You may process these labels yourself (see *Writing Nonstandard Labels on Tape* and *Checking Nonstandard Labels on Tape* in the *Label Processing* chapter). 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 1287 keyboard. The OPEN or OPENR routine reads the header information only when this operand is present. If the entry is omitted, OPEN or OPENR assumes no header information is to be read. The header record size can be as large as the BLKSIZE specification and is read into the high-order positions of IOAREA1. This operand cannot be used for 1288 files.

HPRMTY=YES
This operand is included if you want to be informed of the hopper empty condition. This condition occurs when a READ is issued and no document is present. When hopper empty is detected, your 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 *IBM 1287 Optical Reader Component Description and Operating Procedures*, GA21-9064. If the HPRMTY parameter is used with this method of processing, you are able to check for a hopper empty condition in your COREXIT routine. This allows you then to select the proper stacker for the previously ejected document, before returning from the COREXIT routine (via register 14).

INAREA=name
This operand applies only to a card file in a 1442 that is updated (TYPEFLE=CMBND) and for which separate input and output areas are required. INAREA specifies the 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.

This entry does not apply to combined files in a 2520 or 2540. 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.

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 of characters that are transferred to the input area (INAREA) at any one time. Whenever this operand is included, OUBLKSZ must also be included, and BLKSIZE must be omitted.

IOAREA1=name
This operand is included to specify the 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, reserve eight bytes at the beginning of your 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 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 operand must be included to specify the name of the second I/O area.

For a disk output file, reserve eight bytes at the beginning of your 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 the 1287 reader (journal tape only) this area is set to binary zeros before each input operation to this area.

This operand must not be specified if DEVICE=READ87D or if TYPEFLE=CMBND. In the latter case, IOAREA1 must be used for both the input and output areas.

IOREG=(r)
This operand specifies a general-purpose register (2-12) 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, your 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 WORKA operand must be omitted, and the GET or PUT macros 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.

LABADDR=name
You may require one or more of your own disk or tape labels in addition to the standard file header label or trailer label (on tape). If so, include your own routine to check or build the label(s). The name of your 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 your program. (FILABL=NSTD is specified.)

LABADDR allows you to specify a single label routine for all types of labels for the file: header labels, end-of-file labels, and end-of-volume labels. For an input file, you 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 you the type of label by supplying 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.

You can test this byte in your routine and then build the appropriate type of label. At the end of the routine, return to IOCS by use of the LBRET macro. You may not issue a macro that calls in a transient routine (OPEN, OPENR, CLOSE, CLOSER, DUMP, PDUMP, CANCEL, or CHKPT). For a more complete discussion, see the *Label Processing* chapter.

OUAREA=name
This operand is used with INAREA for a combined file on a 1442 that requires separate input and output areas. It specifies the 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 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 is used in your program.

READ={FORWARD | BACK}
This operand may be included for magnetic tape input to specify the direction in which the tape is to be read. If this entry is omitted, IOCS assumes forward reading. BACK specifies that the tape is to be read backwards.

RECFORM={FIXUNB | FIXBLK | VARUNB | VARBLK | UNDEF}
This operand specifies the type of records in the input or output file. Enter one of the following:

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 follows:

- 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.
- Console printer-keyboard or display operator console 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 operand specifies the number of characters in an individual record. The input/output routines use this number to block or deblock records, and to check record length of input records.

For undefined records, this operand 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, you must load the length in bytes of each record into the specified register before issuing the PUT macro 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 (the CTLCHR operand is 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 CCW chain which you supply.

Note: When processing undefined records on an optical reader in document mode, you gain 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}
This operand may be specified with one of the following parameters:

UNLOAD	To rewind the tape on OPEN or OPENR, and to rewind and unload on CLOSE or CLOSR or on an end-of-volume condition.
NORWD	To prevent rewinding the tape at any time.

If this operand is omitted, tapes are automatically rewound, but not unloaded, on an OPEN or OPENR or CLOSE or CLOSER macro or on an end-of-volume condition.

TPMARK=NO
This operand is included if you do not want a tapemark written as the first record on a tape output file if labels are not specified. This operand is also included if no tapemark is written following nonstandard header labels. If this operand is omitted for a tape output file, a tapemark is the first record if no labels are specified. If this operand is omitted, a tapemark is written following nonstandard header labels.

TRANS=name
This operand applies to input data read from the 2671 paper tape reader: it specifies the name of a code translation table. The table must conform to the specifications of the TR machine instruction.

The input 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/370 programming. For IOCS to perform this translation, you provide a translation table and specify the name of the table in this TRANS operand. The logical IOCS routines then translate the paper tape code and make the record available to you 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 is issued in your program.
- For an input file if the TRUNC macro was issued to write short blocks when the file was originally created.

TYPEFLE={<u>INPUT</u> | 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 (with or without updating)
- 2400, 3420 magnetic tape input
- 1442, 2501, 2520, 2540 card input
- 3210 or 3215 keyboard input (both GET and PUT macros may be issued)
- 1287, 1288 optical reader input

OUTPUT must be specified for:

- 2311 disk output
- 2400, 3420 magnetic tape output
- 1403, 1404, 1443, 3211 printer output
- 1442, 2520, 2540 card output
- 3210 or 3215 printer output (only PUT macros 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.

If the TYPEFLE operand 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 of a general-purpose register (2-12), which always contains the length of the available space remaining in the output area.

After a PUT macro is issued for a variable-length record, IOCS calculates the space still available in the output area and supplies it to you in the designated register. You then compare the length of your 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, issue a TRUNC macro 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 you want 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 name of your routine to which IOCS branches if a wrong-length record is read. In your routine, any operation desired for wrong-length records can be performed. GET macros, however, cannot be used in your routine. Also, if you use any other IOCS macros in your routine, save the contents of register 14. The address of the wrong-length record is supplied by IOCS in register 1. At the end of the routine 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 the WLRERR operand is omitted and a wrong-length record is detected by IOCS, one of the following conditions results:

- If the ERROPT operand is specified for this file, the wrong-length record is treated as an error block and handled according to your specifications for an error (IGNORE, SKIP, or name of error routine).
- If the DTFSR ERROPT operand is not included, the wrong-length record is ignored.

The WLRERR operand does not apply to undefined records because undefined records are not checked for incorrect record length.

WORKA=YES
If records are processed in work areas instead of in the I/O areas, specify this operand. You must set up the work area in storage. The address of the work area, or a general-purpose register which contains the address, must be specified in each GET or PUT macro.

Whenever this operand is specified for a file, the IOREG operand must be omitted. For optical character records, a work area 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 or DASD file. If two or more DTFSR macros are used in the same program, they must 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 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/VS interprets the DTFEN card as a signal to begin generation of the required disk or tape I/O modules.

X	X	X	M	DEVICE=READnn	(01, 20, 40, or 42). For 2501, 2520, 2540, 1442, respectively.
X	X	X	M	DEADDR=SYSxxx	Symbolic unit for reader-punch used for this file.
X		X	M	EOFADDR=xxxxxxxx	Name of your end-of-file routine.
X	X	X	M	TYPEFLE=xxxxxx	(INPUT, OUTPUT, or CMBND). CMBND does not apply to 2501.
X	X		O	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	X	X	O	CONTROL=YES	CNTRL macro used for this file. Omit CTLCHR for this file. Does not apply to 2501.
	X	X	O	CRDERR=RETRY	Retry if punching error is detected. Applies only to 2520 OUTPUT and to 2540 OUTPUT or CMBND.
	X		O	CTLCHR=YES	Data records have control character in first position. Omit CONTROL for this file.
		X	O	INAREA=xxxxxxxx	Name of sep. input area for 1442 CMBND file. Also specify OUAR-EA, and omit IOAREA1 and IOAREA2. Applies only to 1442.
		X	O	INBLKSZ=nn	Length of INAREA. Also specify OUBLKSZ, and omit BLKSIZE.
X	X	X	O	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	X	X	O	IOAREA2=xxxxxxxxx	If two I/O areas are used, name of second I/O area.
X	X		O	IOREG=(nn)	Register number, if two I/O areas are used and GET or PUT does not specify a work area. Omit WORKA.
	X		O	OUAREA=xxxxxxxx	Name of sep output area for 1442 CMBND file. Also specify INAR-EA, and omit IOAREA1 and IOAREA2. Applies to 1442 only.
	X		O	OUBLKSZ=nn	Length of OUAREA. Also specify INBLKSZ and omit BLKSIZE.
X	X	X	O	RECFORM=xxxxxx	(FIXUNB) if TYPEFLE=INPUT. (FIXUNB, VARUNB, or UNDEF) if TYPEFLE=OUTPUT. If omitted, FIXUNB is assumed.
	X		O	RECSIZE=nnnn	Register number if RECFORM=UNDEF.
X	X	X	O	WORKA=YES	GET or PUT specifies work area. Omit IOREG.

M=Mandatory; O=Optional

Figure 2-27 (part 1 of 7) DTFSR macro operands - card

X	X	M	DEVICE=DISK11	
X	X	M	BLKSIZE=nnnn	Length of I/O area, in bytes
X		M	EOFADDR=xxxxxxxx	Name of your end-of-file routine
X	X	M	IOAREA1=xxxxxxxx	Name of first I/O area.
X	X	M	TYPEFLE=xxxxxx	(INPUT or OUTPUT)
X	X	O	CONTROL=YES	CNTRL macro used for this file
X		O	ERROPT=xxxxxxxx	(IGNORE, SKIP, or name of error routine). Prevents job termination on error records.
X	X	O	IOAREA2=xxxxxxxx	If two I/O areas are used, name of second area
X	X	O	IOREG=(nn)	Register number. Use only if GET or PUT does not specify work area. Omit WORKA
X	X	O	LABADDR=xxxxxxxx	Name of your routine to check/write user-standard labels.
X	X	O	RECFORM=xxxxxx	(FIXUNB, FIXBLK, VARUNB, VARBLK, or UNDEF). If omitted, FIXUNB is assumed.
X	X	O	RECSIZE=nnnnn	If RECFORM=FIXBLK, no. of characters in record. If RECFORM=Undef, register no. Not required for other records.
X	X	O	TRUNCS=YES	TRUNC macro used for this file
X		O	UPDATE=YES	Input file is to be updated
	X	O	VARBLD=(nn)	Register number if RECFORM=VARBLK and records are built in the output area.
	X	O	VERIFY=YES	Check disk records after they are written.
X		O	WLRERR=xxxxxxxx	Name of your wrong-length-record routine.
X	X	O	WORKA=YES	GET or PUT specifies work area. Omit IOREG.

M=Mandatory; O=Optional

Figure 2-7 (part 2 of 7) DTFSR macro operands - 2311 disk

x	x	x	x	M	COREXIT=xxxxxxxx	Name of your error correction routine.
x	x	x	x	M	DEVADDR=SYSnnn	Symbolic unit assigned to the optical reader.
x	x	x	x	M	DEVICE=xxxxxxxx	(READ87T or READ87D). For 1288, specify READ87D. If omitted, READ87D is assumed.
x	x	x	x	M	EOFADDR=xxxxxxxx	Name of your end-of-file routine.
x	x	x	x	M	IOAREA1=xxxxxxxx	Name of your first input area.
x	x			O	BLKFAC=nn	If RECFORM=UNDEF in journal tape mode.
x	x	x	x	O	BLKSIZE=nn	Length of I/O area(s). If omitted, 38 is assumed.
x	x	x	x	O	CONTROL=YES	If CNTRL macro is to be used for this file.
x	x	x		O	HEADER=YES	If a header record is to be read from the optical reader keyboard by OPEN or OPENR.
		x	x	O	HPRMTY=YES	If hopper empty condition is to be returned.
x	x			O	IOAREA2=xxxxxxxx	If two input areas are used, name of second input area.
x	x			O	IOREG=(nn)	Reg. number, if two input areas or undefined records are to be used and a work area is not specified.
x	x	x	x	O	RECFORM=xxxxxx	(FIXBLK, FIXUNB, or UNDEF). If omitted, FIXUNB is assumed.
x	x	x	x	O	RECSIZE=(nn)	Register number if RECFORM=UNDEF. If omitted, register 3 is assumed.
x	x	x	x	O	TYPEFLE=xxxxxx	If not specified, INPUT IS ASSUMED.
x	x			O	WORKA=YES	If GET is specified with a work area. Omit IOREG.

M=Mandatory; O=Optional

Figure 2-27 (part 3 of 7) **DTFSR macro operands - optical reader**

M	DEVICE=PRINTER	
M	BLKSIZE=nnn	Length of one output area, in bytes.
M	DEVADDR=SYSnnn	Symbolic unit for the printer used for this file.
M	IOREA1=xxxxxxxx	Name of first output area.
M	TYPEFLE=OUTPUT	
O	CONTROL=YES	CNTRL macro used for this file. Omit CTLCHR for this file.
O	CTLCHR=YES	Data records have control character in first position. Omit CONTROL for this file.
O	IOAREA2=xxxxxxxx	If two output areas are used, name of second area.
O	IOREG=(nn)	Register number, if two output areas are used and PUT does not specify a work area. Omit WORKA.
O	PRINTOV=YES	PRTOV macro is used for this file.
O	RECFORM=xxxxxx	(FIXUNB, VARUNB, or UNDEF). If omitted, FIXUNB assumed.
O	RECSIZE=nnnn	Register number if RECFORM=UNDEF.
O	WORKA=YES	PUT specifies work area. Omit IOREG.

Figure 2-27 (part 4 of 7) **DTFSR macro operands - printer**

M	DEVICE=CONSOLE	
M	BLKSIZE=nnn	Length of I/O area, in bytes.
M	DEVADDR=SYSnnn	Symbolic unit for the console printer-keyboard used for this file.
M	IOAREA1=xxxxxxxx	Name of I/O area.
M	TYPEFLE=xxxxxx	(INPUT or OUTPUT).
O	RECFORM=xxxxxx	(FIXUNB or UNDEF). If omitted, FIXUNB is assumed.
M	RECSIZE=nnnn	Register number if RECFORM=UNDEF.
O	WORKA=YES	GET or PUT specifies work area. Omit IOREG.

M=Mandatory; O=Optional

Figure 2-27 (part 5 of 7) **DTFSR macro operands - console printer-keyboard**

X	X	M	DEVICE=TAPE	
X	X	M	BLKSIZE=nnnnn	Length of one I/O area, in bytes.
X	X	M	DEVADDR=SYSnnn	Symbolic unit for the tape drive used for this file.
X		M	EOFADDR=xxxxxxxx	Name of your end-of-file routine.
X	X	M	IOAREA1=xxxxxxxx	Name of first I/O area.
X	X	M	TYPEFLE=xxxxxx	(INPUT or OUTPUT).
X	X	O	ALTTAPE=SYSnnn	Symbolic unit for alternate tape drive used for this file.
X	X	O	CHECKPT=n	Number assigned to identify 4-byte field for each tape drive specified in first parameter of CHKPT macro (as: 1, 2, or 3).
X		O	CKPTREC=YES	Checkpoint records are interspersed with input data records.
X	X	O	CONTROL=YES	CNTRL macro used for this file.
X		O	ERROPT=xxxxxxxx	(IGNORE, SKIP, or name of error routine). Prevents job termination on error records.
X	X	O	FILABL=xxxx	(STD, NSTD, or NO). If NSTD specified, include LABADDR. If omitted, NO is assumed.
X	X	O	IOAREA2=xxxxxxxx	If two I/O areas are used, name of second area.
X	X	O	IOREG=(nn)	Register number. Use only if GET or PUT does not specify work area. Omit WORKA.
X	X	O	LABADDR=xxxxxxxx	Name of your label routine if FILABL=NSTD or if FILABL=STD and user-standard labels are processed.
X		O	READ=xxxxxxx	(FORWARD or BACK). If omitted, FORWARD is assumed.

M=Mandatory; O=Optional

Figure 2-27 (part 6 of 7) **DTFSR macro operands - tape**

X	X	O	RECFORM=xxxxxx	(FIXUNB, FIXBLK, VARUNB, VARBLK, or UNDEF). If omitted, FIX-UNB is assumed.
X	X	O	RECSIZE=nnnnn	If RECFORM=FIXBLK, number of characters in record. If RECFORM=UNDEF, register number. Not required for other records.
X	X	O	REWIND=xxxxxx	(UNLOAD or NORWD). Unload on CLOSE or CLOSER or end of volume, or prevent rewinding.
	X	O	TPMARK=NO	Prevent writing a tapemark ahead of data records if FILABL=NSTD or NO.
	X	O	VARBLD=(nn)	Register number if RECFORM=VARBLK and records are built in the output area.
X		O	WLRERR=xxxxxxxx	Name of wrong-length-record routine.
X	X	O	WORKA=YES	GET or PUT specifies work area. Omit IOREG.

M=Mandatory; O=Optional

Figure 2-27 (part 7 of 7) **DTFSR macro operands - tape**

IMPERATIVE MACROS

After the SAM files are defined by the declarative macros, the following groups of imperative macros are used to operate on the files:

- Initialization macros
- Processing macros
- Completion macros

Turn back to Figure 2-1 for a summary of both the imperative and declarative macros which may be used for SAM processing on a given I/O device.

Initialization Macros

The initialization macros OPEN and OPENR are used to ready a file for processing. These macros associate the logical file declared in your program with a specific physical file on an I/O device. Thus OPEN or OPENR must be issued for any file before any processing is done for that file; an exception is that OPEN need not be issued for DTFCN and DTFPT files.

The association by OPEN or OPENR of your program's logical file with a specific physical file remains in effect throughout your processing of the file until you issue a completion macro.

OPEN and OPENR also check or write standard or nonstandard DASD or magnetic tape labels. Information on labels is contained in the *Label Processing* chapter.

Included here under the category of initialization macros is the LBRET macro, which is connected only with label processing. LBRET is used to return to IOCS from a subroutine of your program which writes or checks labels.

A description of these macros follows.

OPEN and OPENR Macros

The OPEN or OPENR macro activates all files.

When OPENR is specified, the symbolic address constants that OPENR generates from the parameter list are self-relocating. When OPEN is specified, the symbolic address constants are not self-relocating. The format of these macros is:

Op	Operand
for self-relocating programs	
OPENR	$\begin{Bmatrix}\text{filename1}\\ \text{(r1)}\end{Bmatrix}$ $\left[,\begin{Bmatrix}\text{filename2}\\ \text{(r2)}\end{Bmatrix} \ldots, \begin{Bmatrix}\text{filenamen}\\ \text{(rn)}\end{Bmatrix}\right]$
for programs that are not self-relocating	
OPEN	$\begin{Bmatrix}\text{filename1}\\ \text{(r1)}\end{Bmatrix}$ $\left[,\begin{Bmatrix}\text{filename2}\\ \text{(r2)}\end{Bmatrix} \ldots, \begin{Bmatrix}\text{filenamen}\\ \text{(rn)}\end{Bmatrix}\right]$

To write the most efficient code in a multiprogramming environment it is recommended that OPENR be used.

Self-relocating programs using LIOCS must use OPENR to activate all files, including console files. In addition to activating files for processing, OPENR relocates all address constants within the DTF tables (zero constants are relocated only when they constitute the module address).

If OPEN or OPENR attempts to activate a LIOCS file (DTF) whose device is unassigned, the job is terminated. If the device is assigned IGN, OPEN or OPENR 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 issuing an OPEN or OPENR, input/output operations should not be performed for the file.

Enter the symbolic name of the file (DTF filename) 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, you can load the address of the DTF filename into a register and specify the register using ordinary register notation. The high-order 8 bits of this register must be zeros. For OPENR, the address of filename may be preloaded into any of the registers 2-15. For

OPEN, the address of filename may be preloaded into register 0 or any of the registers 2-15.

Notes: If you use register notation, we recommend that you follow the standard practice of using only registers 2-12. 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 you plan to process user-standard labels (UHL or UTL) or nonstandard tape labels, you 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 OPENR, or issue a separate OPEN or OPENR 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 files for card devices, printers, consoles, magnetic character readers, optical readers, and paper tape devices, OPEN or OPENR simply makes the file available for input or output. For an output file with two I/O areas, OPEN or OPENR loads your IOREG with the address of an I/O area.

If you want, device type checking can be performed when a DTFCN (console) file is opened. The logical unit assignment must be a 3210 or 3215 console printer-keyboard, a display operator console, or a printer. An assignment to any device other than these is invalid and the job is canceled.

If OMR or RCE is specified for a 3505 card reader or if RCE is specified for a 3525 card punch, OPEN or OPENR retrieves the data from the first data card and analyzes this data to verify the presence of a format descriptor card. If a format descriptor card is found, OPEN or OPENR builds an 80-byte record corresponding to the format descriptor card. If a format descriptor card is not found, a message is issued and the job is canceled.

For a 2560, 3525, or 5425 print only file, OPEN or OPENR will feed the first card to ensure that a card is at the print station.

For 2560, 3525, or 5425 associated files, all of the associated files must be opened before a GET or PUT is used for any of the files.

For printers with the Universal Character Set feature, data checks are ignored and blanks are printed unless you specify UCS=ON in the DTFPR.

For MICR devices, OPEN or OPENR sets the entire I/O area to binary zeros.

When LIOCS is used for processing journal tapes on the 1287 optical reader, OPEN or OPENR may be issued at the beginning of each input roll.

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 or OPENR is not reissued, no header information can be entered between tapes.

When processing documents on the 1287, OPEN or OPENR 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 you write, addressability must be provided for your data addresses.

OPEN or OPENR allows header (identifying) information to be entered at the 1287 keyboard for journal tape or cut documents. When header information is entered, it is always read into IOAREA1, which must be large enough to accommodate the desired header information.

When opening a 3886 optical reader file, OPEN or OPENR loads the appropriate format record (as specified in the DTFDR) into the 3886 control unit.

DASD Output

When a multi-volume DASD file is created using SAM, 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 or OPENR 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 or OPENR 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 or OPENR creates the standard label(s) for the file and writes the label(s) in the VTOC.

If you wish to create your own user standard labels (UHL or UTL) for the file, include the DTF LABADDR operand. OPEN or OPENR reserves the first track of the first extent for the user header and trailer labels. Then, your 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, your LABADDR routine is given control and 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 or OPENR 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 or OPENR issues an operator message. The operator has the option of canceling the job or typing in an extent on the console printer-keyboard or the display operator console and continuing the job.

DASD Input
In a multi-volume file 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 or OPENR checks the standard VOL1 label and goes to the VTOC to check the file label(s). OPEN or OPENR 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 or OPENR makes the user standard header labels (UHL) available to you one at a time for checking.

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 or OPENR makes the first extent of the new volume available, it makes the extent(s) from the previous volume unavailable.

Diskette Output
When a multi-volume diskette file is created, feeding from diskette to diskette is automatically performed by IOCS. If the file was defined by a DTFDI macro, the last diskette is ejected automatically by IOCS. If the DTFDU macro was used, the ejection of the last diskette is controlled by the FEED operand of this macro.

When a file is opened, OPEN or OPENR checks the VTOC on the diskette and:

- ensures that the file to be created does not have the same name as an existing unexpired file.
- ensures there is at least one track available to be allocated.

- allocates space for the file, starting at the track following the last unexpired or write-protected file on the diskette.

Diskette Input
When a multi-volume diskette file is read, feeding from diskette to diskette is automatically performed by IOCS. If the file was defined by a DTFDI macro, the last diskette is not ejected. If the DTFDU macro was used, the ejection is controlled by the FEED operand of this macro.

When a file is opened, OPEN or OPENR checks the VTOC on the diskette and determines the extent limits of the file from the file label.

After the label is 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 (if the statements are not numbered, job control numbers them consecutively). The same extent statements used to build the file can be used when the file is used as input.

LBRET Macro

Name	Operation	Operand
[name]	LBRET	{1 | 2 | 3}

The LBRET macro is issued in your subroutines when you have completed processing labels and wish 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--1, 2, or 3--depends on the function to be performed. The functions and operands are explained below. See also the *Label Processing* chapter.

Checking User Standard DASD Labels: IOCS passes the labels to you one at a time until the maximum allowable number is read (and updated), or until you signify you want no more. In the label routine, use LBRET 3 if you want IOCS to update (rewrite) the label just read and pass you the next label. Use LBRET 2 if you simply want IOCS to read and pass 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 you want to eliminate the checking of one or more remaining labels, use LBRET 1.

Writing User Standard DASD Labels: Build the labels one at a time and use LBRET to return to IOCS, which writes the labels. Use LBRET 2 if you want control returned to you after IOCS writes the label. If, however, IOCS determines that the maximum number of labels has already been written, label processing is terminated. Use LBRET 1 if you wish to stop writing labels before the maximum number of labels is written.

Checking User Standard Tape Labels: IOCS reads and passes the labels to you one at a time until a tapemark is read, or until you signify you do not want any more labels. Use LBRET 2 if you want to process the next label. If IOCS reads a tapemark, label processing is automatically terminated. Use LBRET 1 if you want to bypass any remaining labels.

Writing User Standard Tape Labels: Build the labels one at a time and return to IOCS, which writes the labels. When LBRET 2 is used, IOCS returns control to you (at the address specified in LABADDR) after writing the label. Use LBRET 1 to terminate the label set.

Writing or Checking Nonstandard Tape Labels: You must process all your nonstandard labels at once. Use LBRET 2 after all label processing is completed and you want to return control to IOCS. *Appendix C* shows an example of this.

Processing Macros

The processing macros permit you to store and retrieve records without coding your own blocking/deblocking routines. You can thereby concentrate on processing your data.

This description of processing macros begins with the general macros which can be used for many different devices--such as GET, which can be used for every input device; PUT, which can be used for every output device; and CNTRL, which can be used for many input and output devices. Then follow descriptions of processing macros for more specific applications: *Magnetic Reader Macros*, *Optical Reader Macros*, and *Work File Macros for Tape and Disk.*

A major feature of the processing macros is the ability to use one or two I/O areas and to process records in either a work area or in an I/O area.

The SAM routines provide for overlapping the physical transfer of data with processing. The amount of overlapping actually achieved is governed by your program through the assignment of I/O areas and work areas. An I/O area is that area of storage to or from which a block of data is transferred by LIOCS. A work area is an area used for processing an individual record (part of the block of data). A work area cannot be used with paper tape records or for the 3881 Optical Mark Reader. The I/O area(s) is specified in the DTF macro, while the work area is specified in the processing macro.

The following combinations of I/O areas and work areas are possible (except in the cases of spanned records and associated DTFCD files):

1. One I/O area **without** a work area
2. One I/O area **with** a work area
3. Two I/O areas **without** a work area
4. Two I/O areas **with** a work area

When processing spanned records, you may use:

1. One I/O area **with** a work area
2. Two I/O areas **with** a work area

Although two I/O areas are permitted, normal overlap is curtailed because each I/O command you issue may require multiple I/O operations by MTMOD or SDMODxx.

When processing associated DTFCD files, you may use:

1. One I/O area **without** a work area
2. One I/O area **with** a work area

Although two I/O areas are not permitted for associated files, a type of overlapped processing can nevertheless be achieved-see the *DOS/VS Data Management Guide*, GC33-5372, for details.

GET Macro

Name	**Operation**	**Operand**
[name]	GET	$\left\{\begin{matrix}\text{filename}\\ 1\end{matrix}\right\}\left[\begin{matrix}\text{, workname}\\ 0\end{matrix}\right]$

GET makes the next sequential logical record from an input file available for processing in either an input area or in a specified work area. 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: This operand is required. 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 necessary to make your programs self-relocating.

workname: This is an optional parameter specifying the work area name or a register (in either special or ordinary register notation) containing the address of the work area. The work area address should never be preloaded into register 1. This parameter is used if records are to be processed in a work area which you define yourself (for example, using a DS instruction). If the operand is specified, all GETs for the named file must use a register or a workname. Using the second operand causes GET to move each individual record from the input area to a work area. The workname parameter is not valid for 3881. You also cannot specify the WORKA operand in the DTFCD for the 3881.

All records from a logical file may be processed in the same work area, or different records from the same logical file may be processed in different work areas. In the first case, each GET for a file specifies the same work area. In the second case, different GET macros specify different work areas. It might be advantageous to plan two work areas, for example, and to specify each area in alternate GET macros. This would permit you to compare each record with the preceding one, checking, for instance, for a change in a control field within the record. However, only one work area can be specified in any one GET macro.

Required DTF Macro Entries
The input area must be specified in the IOAREA1 operand of the DTF macro. For any file other than a combined file or an associated 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.

For associated files only one input area may be specified.

When records are processed in the input area(s), a register must be specified in the IOREG operand of the DTF macro if:

1. Records are blocked, (chained if on diskettes) 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

The specified 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.

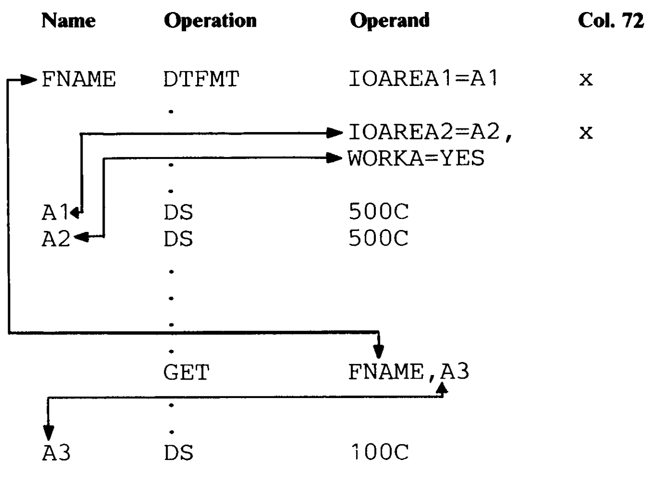
If a work area is used, WORKA=YES must be specified. IOREG should not be specified.

When the GET macro detects an end-of-file condition, IOCS branches to your end-of-file routine (specified by EOFADDR). For MICR document processing, you do 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 macro processing is shown in Figure 2-28. The operand IOAREA1 points to the first I/O area for this file. IOAREA2 points to the second I/O area. The operands of the GET macro point to the DTF and to the work area A3 to which logical records are moved from areas A1 and A2 by LIOCS.

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 work area if one is specified in the



```
Name     Operation     Operand          Col. 72

FNAME    DTFMT         IOAREA1=A1          x
          .
                       IOAREA2=A2,         x
          .            WORKA=YES
          .
A1       DS            500C
A2       DS            500C
          .
          .
          .
          .
         GET           FNAME,A3
          .
          .
A3       DS            100C
```

Figure 2-28 GET macro processing example

GET macro. If two input areas are specified, each GET makes the last record that was transferred to virtual storage available for processing in the input area or work area. The same GET also starts the transfer of the following record to the other input area.

When a 2540 is used for a card input file, each GET macro normally reads the record from a card in the read feed. However, if the 2540 has the special punch-feed-read feature installed and if TYPEFILE=CMBND is specified in the DTFCD macro, 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 macro is issued. (See *Updating* in the discussion of the PUT macro, below.)

Blocked Records
When records on DASD or magnetic tape are specified as blocked in the DTF RECFORM operand, 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 macro 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 work area.

2. Subsequent GET macros either add an indexing factor to the register or move the proper record

to the specified work area, until all records in the block are processed.

3. Then, the next GET makes a new block of records available in virtual storage, and either initializes the register or moves the first record.

Command Chained Records
When records on diskettes are specified as command chained in the DTFDU CMDCHN operand, each individual record must be located for processing. Therefore, command chained records are handled as follows:

1. The first GET macro transfers a chain of records (2, 13, or 26 records depending on the CMDCHN operand) from diskette to the input area. It also intializes the specified register to the absolute address of the first data record, or it transfers the first record to the specified work area.

2. Subsequent GET macros either add an indexing factor to the register or move the proper record to the specified work area, until all records in the block are processed.

3. Then, the next GET makes a new chain of records available in virtual storage, and either initializes the register or moves the first record.

Spanned Records
When unblocked or blocked spanned records are processed, the operand RECFORM=SPNUNB or RECFORM=SPNBLK, respectively, must be included in both the file definition (DTFMT or DTFSD) and the appropriate module (MTMOD, SDMODVI, or SDMODVU). GET assembles spanned record segments into logical records in your work area. Null segments are recognized. They are not assembled into logical records, but skipped. The length of the logical record is passed to you through the register specified in the DTF RECSIZE operand.

If you choose 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 processed, the operand RECFORM=UNDEF must be included in the DTF. GET treats undefined records as unblocked, and you 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 by IOCS to be variable in length. No other characteristics of the record are known or assumed by IOCS. Determining these characteristics is your responsibility.

Associated Files
For associated files on the 2560, 3525, and 5425 card devices, the sequence in which you issue GET, CNTRL, and PUT macros is important. See the section *GET/CNTRL/PUT Sequence for Associated Files* under *PUT Macro*, below.

Reading Magnetic Tape Backwards
If records on magnetic tape are read backwards (the DTF operand READ=BACK is specified), blocks of records are transferred from tape to virtual storage in reverse order. The last block is read first, the next-to-last block is read second, etc. For blocked records, each GET macro 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 work area).

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	$\left\{\begin{matrix}\text{filename}\\(1)\end{matrix}\right\}\left[,\left\{\begin{matrix}\text{workname}\\(0)\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\left\{\begin{matrix}\text{STLSP=controlfield}\\(r)\end{matrix}\right\}\\\left\{\begin{matrix}\text{STLSK=controlfield}\\(r)\end{matrix}\right\}\end{matrix}\right\}\right]$

PUT writes or punches logical records which are built directly in the output area or in a specified work area. 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 for a printer, the printer automatically spaces one line. Neither the CNTRL macro nor a control character need be specified.

filename: This operand is required. 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 necessary to make your programs self-relocating.

workname: An optional parameter specifying the work area name or a register (in either special or ordinary register notation) containing the address of the work area. The work area address should never be preloaded into register 1. This parameter is used if records are built in a work area which you define yourself (for example, using a DS instruction). If the operand is specified, all PUTs to the named file must use a register or a workname. Using the second operand causes PUT to move each record from the work area to the output area.

Individual records for a logical file may be built in the same work area or in different work areas. Each PUT macro specifies the work area where the completed record was built. However, only one work area can be specified in any one PUT macro.

Whenever a PUT macro transfers an output data record from an output area (or work area) to an I/O device, the data remains in the area until it is either cleared or replaced by other data. IOCS does not clear the area. Therefore, if you plan to build another record whose data does not use every position of the output area or work area, you must clear that area before you build the record. If this is not done, the new record will contain interspersed characters from the preceding record.

STLSP=control field: This optional operand specifies a control byte that allows for spacing while using the selective tape listing feature on the 1403 printer. To use this feature, the operand STLST=YES must be specified in the DTFPR. Up to 8 paper tapes may be independently spaced. The control byte is set up like any other data byte in virtual storage. You can also use ordinary register notation to provide the address of the control byte. Registers 2 through 12 are available without restriction. You determine the spacing (which occurrs after printing) by setting on the bits corresponding to the tapes you want to space. The correspondence between control byte bits and tapes is as follows:

Control 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 control byte that allows for skipping while using the selective tape listing feature on the 1403 printer. To use this feature, the operand STLIST=YES must be specified in the DTFPR. Up to 8 paper tapes may be independently skipped. The control byte is set up like any other data byte in virtual storage. You can also use ordinary register notation to provide the address of the control byte. Registers 2 through 12 are available without restriction. You determine the skipping (which occurs after printing) by setting on the bits corresponding to the tapes you want to skip. The correspondence between control byte bits and tapes is shown in the figure under *STLSP=control field*, above.

Required DTF Operands

The output area must be specified in the IOAREA1 operand of the DTF macro. For any file other than a combined file or an associated 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.

For associated DTFCD files only one output area may be specified.

When records are built in the output area(s), a register must be specified in the IOREG operand 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. You should always address the I/O areas by using the IOREG as the base register and should not make any assumptions about which I/O area is presently being used. If a work area 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 VARBLD operand. IOCS stores the number of bytes remaining in the output area in the VARBLD register each time a PUT macro is executed.

Unblocked Records
Records transferred to any output file are always considered fixed unblocked unless otherwise specified in the DTF RECFORM operand.

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 work area is specified in the PUT macro, the record is first moved from the work area 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 before it can be 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 work area. Each PUT macro for these records either adds an indexing factor to the register (IOREG), or moves the completed record from the specified work area to the proper location in the output area. When an output block of records is complete, a PUT macro causes the block to be transferred 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 in a work area. The length of each variable-length record must be determined by your program and included in the output record as it is built. Your 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 work area, the PUT macro 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 into the work area.

However, if variable-length blocked records are built directly in the output area, the DTF VARBLD operand, the TRUNC macro, and additional programming are required. Your 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 is supplied to your program in a register by the IOCS routines if VARBLD is specified in the DTF. This register is in addition to the register specified in IOREG. Each time a PUT macro is executed, IOCS loads into the specified register the number of bytes remaining in the output area. Your program uses this to determine if the next variable-length record will fit. If it will not fit, a TRUNC macro 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.

Command Chained Records
When command chained records are written on diskettes, the individually built records must be formed into a chain in the output area before they can be transferred to the output file.

Command chained records can be built directly in an output area or in a work area. Each PUT macro for these records either adds an indexing factor to the register (IOREG), or moves the completed record to the proper location in the output area. When an output chain of records is complete, a PUT macro causes the chain of records to be transferred to the output file and initializes the register, if one is used.

Spanned Records
When PUT handles unblocked or blocked spanned records, the operand RECFORM=SPNUNB or RECFORM=SPNBLK (whichever applies) must be included in both the file definition (DTFMT, DTFSD) and the appropriate module (MTMOD, SDMODVO, or SDMODVU). Records in your work area are divided into spanned record segments according to the length specified in the RECSIZE operand. In constructing the segments, full use is made of the space available in each physical record and device track. For disk 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 record.

Undefined Records
When undefined records are processed, PUT treats them as unblocked. You must provide any blocking desired. You must also determine the length of each record (in bytes) and load it in a register before issuing the PUT macro for that record. The register used for this purpose must be specified in the DTF RECSIZE operand.

Updating
A DASD record may be read, modified, and written back to the same DASD location from which it was read. This is possible with all DASD devices. A card record may, with some devices, be read and then have additional information punched back into the same card. This is possible with the 1442, 2520, 2560, 3525, and 5425, and with the 2540 equipped with the special punch-feed-read feature.

For the card devices, there are two ways of specifying in the DTFCD that such updating is desired; which way is used depends on device type.

- For the 1442, 2520, or 2540 equipped with the punch-feed-read feature, use a combined file by specifying TYPEFLE=CMBND in the DTFCD. An example of a combined card file is given below. For the 2540 with the punch-feed-read feature, the file to be updated must be in the punch feed.
- For the 2560, 3525, or 5425, use associated files. Associated files are described in the *DOS/VS Data Management Guide*, GC33-5372; they are defined in the associated file declarations (DTFCD and DTFPR) by the ASOCFLE and FUNC operands.

When updating a file, one I/O area can be specified (using the IOAREA1 operand) for both the input and output of a card record. If a second I/O area is required, it can be specified with the IOAREA2 operand. For associated DTFCD files, however, two I/O areas are not allowed.

A PUT for a DASD file or for a combined card file must always be followed by a GET before another PUT is issued: GETs, however, can be issued as many times in succession as desired. The corresponding rules for an associated card file are given in the section *GET/CNTRL/PUT Sequence for Associated Files*, below. 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 using the 2540 with the punch-feed-read special feature, a PUT macro must be issued for each card. For a 1442 or 2520 file, however, a PUT macro may be omitted if a particular card does not require punching. The operator must run out the 2540 punch following a punch-feed-read job.

In the following combined card file example, data is punched into the same card which 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
        .
        .
```

GET/CNTRL/PUT Sequence for Associated Files
For 2560, 3525, or 5425 associated files, GET, CNTRL, and PUT macros must be used with the files in the sequence given in Figure 2-29. For example, to process a card of a read-punch associated file requires first issuing a GET macro for the file defined in the read DTFCD, and then issuing a CNTRL macro (if desired) for the file declared in the punch DTFCD, and then issuing a PUT macro for the file declared in the punch DTFCD.

GET/PUT sequences other than those given in Figure 2-29 will cause an abnormal termination with an illegal supervisor call 32 message. The use of CNTRL in sequences other than those shown in Figure 2-29 will cause unpredictable results.

Improved performance for the 2560 or 5425 may be achieved by a type of overlapped processing through the use of dummy PUTs. Details of this technique may be found in the *DOS/VS Data Management Guide*, GC33-5372.

To:	Issue:	For file declared in:	FUNC=
READ/PUNCH	GET	DTFCD (read file)	RP
	[CNTRL]*	DTFCD (punch file)	
	PUT	DTFCD (punch file)	
READ/PUNCH/PRINT	GET	DTFCD (read file)	RPW
	[CNTRL]*	DTFCD (punch file)	
	PUT	DTFCD (punch file)	
	[PUT]**	DTFPR	
READ/PRINT	GET	DTFCD	RW
	[CNTRL]*	DTFCD	
	[PUT]**	DTFPR	
PUNCH/PRINT	[CNTRL]*	DTFCD	PW
	PUT	DTFCD	
	[PUT]**	DTFPR	

* Optional. If used, however, the sequence is as shown.
** Optional provided you do not want to print on the card. If used, however, the sequence is as shown.

Figure 2-29 GET/CNTRL/PUT macro usage to process one card of an associated file

2560 Printing
The 2560 has a maximum of 6 print heads, one for each line of print. For a description of how the print heads may be set, see *IBM 2560 Multi-Function Card Machine Component Description and Operating Procedures*, GA26-5893. The output area can be as large as 384 bytes, printed 64 bytes per line.

With one PUT macro, one logical line of up to 384 characters in length is printed. This logical line is split up into 6 physical lines. Thus a single PUT macro prints all the information for a card. The next PUT macro will cause printing for the next card.

3525 Printing
For a 3525 with a 2-line printer, output is automatically printed on lines 1 and 3.

When automatic line positioning is used for a print only file on a 2-line printer, the one PUT macro causes line 3 to be printed and the other PUT causes a new card to be fed and the printing of line 1 to be started.

With a multiline printer, card feeding is caused by the PUT macro which follows the PUT causing printing on line 25. This PUT macro also starts the printing on line 1 of the next card. If you want to control line positioning either the CONTROL operand or the CTLCHR operand must be specified in the DTFPR macro. (CONTROL and CTLCHR are not valid for card feeding when they are specified for a printer file associated with a read or punch file.) You are then responsible for all spacing and skipping during printing. If CTLCHR=YES is specified, you are also responsible for card feeding.

The following restrictions apply to user-controlled line positioning:

1. Any attempt to print on lines other than lines 1 or 3 on a 2-line printer results in a command reject. Otherwise, 2-line printer support is identical to multiline printer support.

2. A space after printing command for line 25 results in positioning on line 1 of the next card.

3. Any attempt to print and suppress spacing results in a command reject.

4. Any skip command to a channel number less than or equal to the present channel position results in line positioning at that channel position on the next card.

5. If CONTROL or CTLCHR is specified, FUNC is ignored for 2-line printer support.

5425 Printing
The 5425 has a maximum of 4 print heads, one for each line of print. The output area can be as large as 128 bytes, printed 32 bytes per line.

With one PUT macro, one logical line of up to 128 characters in length is printed. This logical line is split up into 4 physical lines. Thus a single PUT macro prints all the information for a card. The next PUT macro will cause printing for the next card.

Card Device and Printer Control
Output stacker selection for a card device, and line spacing or skipping for a printer, can be controlled either by specified control characters in the data records or by the CNTRL macro. Either method, but not both, may be used for a particular file. For use of the latter method, see *CNTRL Macro*, below.

When control characters in data records are used, the DTF CTLCHR operand must be specified, and every record must contain a control character in the virtual-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 macro is executed, the control character in the data record determines the command code (byte) of the CCW that IOCS establishes. The control character is used as follows:

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 a 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').

If a program using ASA control characters prints on the 3525, you must use a space 1 control character (a blank) to print on the first line of a card.

The particular character to include in the record depends on 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 A* gives a complete list of control characters.

PUTR Macro

Name	Operation	Operand
[name]	PUTR	$\begin{Bmatrix}\text{filename}\\(1)\end{Bmatrix}$ $\left[,\begin{Bmatrix}\text{workname1,workname2}\\(0)\quad(2)\end{Bmatrix}\right]$

The PUTR (PUT with reply) macro is used for the display operator console, to issue a message to the operator which requires operator action and which will not be deleted from the display screen until the operator has issued a reply.

You may also use PUTR with the 3210 or 3215 console printer-keyboard, in which case PUTR functions the same as PUT followed by GET for these devices, but provides the message non-deletion code for the display operator console. Use of PUTR for the 3210 or 3215 is therefore recommended for compatibility if your program may at some time be run on the display operator console instead of the 3210 or 3215.

Use PUTR for fixed unblocked records (messages). Issue PUTR after a record has been built.

filename: This operand is required. The parameter value must be the same as specified in the header entry of the DTFCN for the file being built. The filename can be specified as a symbol or in either special or ordinary register notation. The latter is necessary to make your programs self-relocating.

workname1: An optional parameter specifying the output work area name or a register (in either special or ordinary register notation) containing the address of the output work area. The work area address should never be preloaded into registers 1 or 2. This parameter is used if records are built in a work area which you define yourself (for example, using a DS instruction). The length of the work area is defined by the BLKSIZE parameter of the DTFCN macro. If workname1 is specified, workname2 must also be specified.

workname2: An optional parameter specifying the input work area name or a register (in either special or ordinary register notation) containing the address of the input work area. The work area address should never be preloaded into registers 0 or 1. This parameter is used if records are built in a work area which you define yourself (for example, using a DS instruction). The length of the work area is defined by the INPSIZE parameter of the DTFCN macro. If workname2 is specified, workname1 must also be specified.

RELSE Macro

Name	Operation	Operand
[name]	RELSE	$\begin{Bmatrix}\text{filename}\\(1)\end{Bmatrix}$

The RELSE (release) macro 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 you to skip the remaining records in a block and continue processing with the first record of the next block when the next GET macro 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 a CNTRL macro with the codes FSL or BSL (tape spacing for spanned records), then the FSL or BSL logical record spacing is ignored.

The symbolic name of the file, specified in the DTF header entry, is the only parameter required for this macro. It can be specified as a symbol or in register notation.

RELSE discontinues the deblocking of the present block of records, which may be either fixed or variable length. RELSE causes the next GET macro 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 work area.

For example, RELSE may be used in 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	$\left\{\begin{matrix}\text{filename}\\(1)\end{matrix}\right\}$

The TRUNC (truncate) macro is used with blocked output records written on DASD or magnetic tape. It allows you to write a short block of records. Blocks do not include padding. Thus, the TRUNC macro can be used for a function similar to that of the RELSE macro for input records. That is, when the end of a category of records is reached, the last 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 work area and the program determines that a record belongs in a new block, TRUNC should be issued first to write the block. This should be followed by a PUT for this particular record to move the record into the new block. If records are built in the output area, however, you must determine if a record belongs in the block before you build the record.

Whenever variable-length blocked records are built directly in the output area, the TRUNC macro must be used to write a complete block of records. When a PUT is issued after each variable-length record is built, the output routines supply you with the space (number of bytes) remaining in the output area. From this, you determine if your next variable-length record fits in the block. If it does not fit, issue the TRUNC macro to write the block and make the entire output area available to build the next record. The amount of remaining space is supplied in the register specified in the DTF VARBLD operand.

CNTRL Macro

Name	Operation	Operand
name]	CNTRL	$\left\{\begin{matrix}\text{filename}\\(1)\end{matrix}\right\}$, code[,n1][,n2]

The CNTRL (control) macro provides commands for magnetic tape units, card devices, printers, DASDs, and optical readers. Commands apply to physical nondata operations of a unit and are specific to the unit involved. They specify such functions as rewinding tape, card stacker selection, and line spacing on a printer. 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 you, except when certain mnemonic codes are specified for optical readers.

CNTRL 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 (see Figure 2-30).

The third parameter, n1, is required whenever a number is needed for stacker selection, immediate printer carriage control, or for line or page marking on the 3886. The fourth parameter, n2, applies to delayed spacing or skipping, or to timing mark check on the 3886. In the case of a printer file, the parameters n1 and n2 may be required.

The CNTRL macro 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 your program, the DTF CONTROL operand must be included (except for DTFMT and DTFDR) 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
1442, 2520 Card Read Punch	SS		1 2	Select Stacker 1 or 2
	E			Eject to Stacker 1 (1442 only)
2540 Card Read Punch 3504, 3505 Card Readers 3525 Card Punch	PS	1 2 3		Select Stacker 1, 2, or 3 (For 3504, 3505, and 3525, 3 Defaults to Stacker 2)
2560 Multi-Function Card Machine	SS	1 2 3 4 5		Select Stacker 1, 2, 3, 4, or 5
2596 Card Read Punch	SS	1 2		Select Stacker 1 for Read, or Stacker 3 for Punch Select Stacker 2 for Read, or Stacker 4 for Punch
5425 Multi-Function Card Unit	SS	1 2 3 4		Select Stacker 1, 2, 3, or 4
1403, 1443, 3203, 3211, 5203 Printers 3525 Card Punch with Print Feature	SP	See Note c	d	Carriage Space 1, 2, or 3 lines
	SK	c	d	Skip to Channel c and/or d (For 3525, a Skip to Channel 1 is Valid Only for Print Only Files.)
1403, 5203 Printers with Universal Character Set Feature or 3203, 3211 Printers	UCS	ON OFF		Data Checks are Processed with an Operator Indication 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, 2314, 2319, 3330, 3333, 3340 DASD	SEEK			Seek to Address
3881 Optical Mark Reader	PS	1 2		Select stacker 1 or 2
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
3886 Optical Character Reader	DMK	name (r) number		Page mark the document when it is stacker selected as specified in parameter n1.
	LMK	name (r) number, number		Line mark the document when it is stacker selected as specified in parameter n1.
	ESP	1 2	name (r) number	Eject and stacker select the current document to stacker A or B. Perform line mark station timing mark check as indicated in parameter n2.

Note: c = An Integer that Indicates Immediate Printer Control (before printing).
d = An Integer that Indicates a Delayed Printer Control.

Figure 2-30 CNTRL macro command codes

Magnetic Tape Unit Codes
The CNTRL macro 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.

This allows the tape to be positioned at the desired location for opening a file, so that:

- The tape can be positioned to a file located in the middle of a multi-file-reel.
- Rewinding of the tape can be performed even if NORWD was specified in the DTF REWIND operand.

Note: If you are using a self-relocating program, you must open the file before issuing any commands for it.

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 you to skip over a physical tape record (from one interrecord gap to the next). The record passes without being read into storage. The FSF (or BSF) function permits you to skip to the end of the 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 you do not want to process the remaining records in the block, nor one or more succeeding blocks, issue a RELSE macro before the CNTRL macro. The next GET then 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 work area, IOCS always reads one tape block ahead of the one that is being processed. Thus, the next block after the current one is in storage ready for processing. Therefore, if a CNTRL FSR is given, the second block beyond the present one is passed without being read into 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.
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 can be used for forms control on any printer. 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, n1, is required for immediate spacing and skipping (before printing). The fourth parameter, n2, 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.

Printer With the UCS Feature
The CNTRL macro can be used before a PUT for the file to change the method of processing data checks. Data checks can be either processed with an indication given to the operator, or ignored with blanks printed in place of the unprintable characters.

A data check occurs when a character except null, (hexadecimal 00), or blank, (hexadecimal 40) sent to the printer does not match any of the characters in the UCS buffer.

Before opening a file, the BLOCK parameter of the UCS job control command determines for a 1403 whether data check processing takes place. For another UCS printer, the NOCHK option of the SYSBUFLD program (see *DOS/VS System Control Statements*, GC33-5373) has the same meaning.

If several DTFPRs are assigned to the same physical unit, the UCS parameter of the DTF last opened determines whether data check processing takes place. If a DTFDI is opened for a UCS printer, it has the effect of a NOCHK option. This change is operated on the physical device and is valid for all DTFs assigned to this device.

If the UCS form of the CNTRL macro is used for a printer without the UCS feature, the CNTRL macro is ignored.

Except on a 1403, 3203, or 5203, the CNTRL macro can also be used before a PUT for the file to control the printing of lower-case letters. 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 UCB 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 is printed. If UNFOLD is specified, the character equivalent of the EBCDIC byte is printed.

1442 and 2520 Card Read Punch Codes
Cards fed in the 1442 and 2520 are normally directed to the stacker specified in the DTF SSELECT operand. If SSELECT is omitted, they go to stacker 1. The CNTRL macro can be used to temporarily override the normally selected stacker.

Input File: CNTRL can be used only when one I/O area, with or without a work area, is specified for the file. To stack a particular card, the CNTRL macro should be issued after the GET for that card, and before the GET macro 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 work areas. To stack a particular card, the CNTRL macro 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 work areas. If a particular card is to be selected, the CNTRL macro 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.

2540 Card Read Punch Codes
Cards read or punched on the 2540 normally fall into the stacker specified in the DTF SSELECT operand (or the R1 or P1 stacker 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. The possible selections are shown below. (These selections are also those which may be specified in the DTF SSELECT operand.)

Feed	**Stacker**	**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 work area, is specified for the file. To stack a particular card, the CNTRL macro should be issued after the GET for that card. Before the next GET macro is executed, the card is stacked into the specified stacker.

Note: If your program indicates that operator intervention is required on a 2540 (for example, to correct a card out of sequence in a card deck), your program has specified CONTROL=YES in CDMOD, and you do not use the CNTRL macro, then you should issue a CNTRL macro before the operator intervention is requested. Issuing CNTRL 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 work areas. When you want to select a particular card, CNTRL must be issued before the PUT for that card. However, CNTRL does not have to precede every PUT.

2560 and 5425 Card Device Codes
Cards fed into the 2560 or 5425 are normally directed to the output stacker specified in the DTF SSELECT operand. If SSELECT is omitted, cards which came from hopper 1 go to output stacker 1; and cards which came from hopper 2 go to output stacker 5 for the 2560, or output stacker 4 for the 5425. The CNTRL macro can be used to temporarily override the normally selected stacker.

Single File: CNTRL cannot be used for a print file. For a read file, to stack a particular card the CNTRL macro must be issued after the GET for that card. For a punch file, or a punch/interpret file (DTFCD FUNC=I), to stack a particular card CNTRL must be issued before the PUT for that card.

Associated File: The sequence of CNTRL macro usage with associated files is described below and is summarized in the section *GET/CNTRL/PUT Sequence for Associated Files* under *PUT Macro*, above. CNTRL must be used with only one of the associated files:

- With the read file if the associated file is read/print. In this case, to stack a particular card CNTRL must be issued after the GET and before any PUT for that card. If no PUT is issued for that card, then CNTRL must be issued after the GET for that card and before the GET for the next card.

- With the punch file if the associated file is anything other than read/print. In this case, to stack a card CNTRL must be issued before the PUT which punches that card.

2596 Card Read Punch Codes
Cards fed into the 2596 are normally directed to the stacker specified in the DTF SSELECT operand. If SSELECT is omitted, cards go to stacker 1 for read and stacker 3 for punch. The CNTRL macro can be used to temporarily override the normally selected stacker. The possible selections are shown in Figure 2-25. (These selections are also those which may be specified in the DTF SSELECT operand.)

Input File: CNTRL can be used only when one I/O area, with or without a work area, is specified for the file. To stack a particular card, the CNTRL macro should be issued after the GET for that card, and before the GET for the next card. When the next card is read, the previous card is stacked in the specified stacker.

Output File: CNTRL can be used with any permissible combination of I/O areas and work area. To stack a particular card, the CNTRL macro should be issued before the PUT for that card. After the card is punched it is stacked into the specified stacker immediately.

3504 and 3505 Card Readers and 3525 Card Punch Codes
Cards read on the 3504 or 3505 or punched on the 3525 are normally directed to the stacker specified in the DTF SSELECT operand. If SSELECT is omitted, stacker 1 is assumed. The CNTRL macro can be used to temporarily override the normally selected stacker. For input files, CNTRL can be used only when one I/O area is specified for the file.

3525 Card Printing Codes
The CNTRL macro can control spacing and skipping to a specific line on a card for the 3525 card print feature. The command code SP is used to direct the 3525 to space one, two, or three lines on a card; and SK is used to skip to a channel (1-12) on a card. (See the section *3525 Printing* under *PUT Macro*, above.

The 3525 print channels correspond to specific rows on a printed card. The channels and their corresponding card rows are shown in Figure 2-31.

Line Number	Channel Number
1	1
2	
3	2
4	
5	3
6	
7	4
8	
9	5
10	
11	6
12	
13	7
14	
15	8
16	
17	9 (overflow)
18	
19	10
20	
21	11
22	
23	12 (overflow)
24	
25	

Figure 2-31 **3525 print channels**

DASD Codes
The CNTRL macro to seek (SEEK) for any DASD device, or to restore (RESTR) for the 2321, permits access movement to begin for the next READ, WRITE, GET, or PUT macro. While the arm is moving for a SEEK or the strip is being restored on a data cell, you can process data and/or request I/O operations on other devices.

IOCS seeks the track that contains the next block for that file without your having to 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 macro is issued.

3881 Optical Mark Reader Codes
Documents read by the 3881 are directed to the stacker specified in the CNTRL macro or to the stacker specified on the format control sheet. Stacker 1 is the normal stacker and stacker 2 is the select stacker. If you use both the CNTRL macro and the format control sheet to control stacker selection and either specifies stacker 2, data documents are stacked in stacker 2. The DTF SSELECT operand is not valid for the 3881.

1287 and 1288 Optical Reader Codes
The CNTRL macro for the 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 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 1287 keyboard when processing journal tapes. This permits the operator to key in a complete line on the keyboard if a 1287 read error makes this type of correction necessary. When IOCS exits to your COREXIT routine, you may issue the CNTRL macro to read from the keyboard. The 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, you 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 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 must be issued in your 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. 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 with the SSD mnemonic.

The CNTRL macro 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.

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. Only selection numbers 1 and 3 are available for the 1288.

If a CNTRL macro is issued in a COREXIT routine and a late stacker select or nonrecoverable 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 CCW 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 separately. 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.

3886 Optical Character Reader Codes

When you are using the 3886 Optical Character Reader, you can use the CNTRL macro to perform the following operations:

- Page mark the current document
- Line mark the current document
- Eject and stacker select the current document
- Perform timing mark check

When the operation has been completed successfully, control is returned to the next instruction in your program. If the operation does not complete successfully, the COREXIT routine receives control. The end-of-file routine receives control when an operation is requested but no documents are available and the end-of-file key has been pressed.

The contents of parameters n1 and n2 vary depending on the mnemonic operation code specified. Therefore, this discussion treats each mnemonic code separately.

DMK,n1: Specifies that the document currently being processed is to be marked when the next eject/stacker-select command is issued. The digits to be printed on the page are specified by the four low-order bits of the field indicated in parameter n1. The sum of the mark digits printed will equal the value specified in the four bits. The high-order four bits of the field are not used. You can specify the digits you want printed in one of three ways:

- name specifies the symbolic name of a one-byte field in your program in which the low-order four digits indicate the combination of digits to be printed.

- (r) indicates the number of the register that contains the address of the one-byte field used for page marking.

- number indicates the sum of the digits to be printed. The decimal number may be from 1 through 15.

LMK,n1: Specifies a line on the current document that is to be line-marked when the eject/stacker-select command is issued. The digits to be printed and the line on which they should be printed are specified in a two-byte field. The digits to be printed are specified in the low-order four bits of the first byte as in the document marking operation. The line to be marked is specified in binary in the low-order six bits of the second byte of the field. You can specify the mark digits and line number in three ways:

- name specifies the symbolic name of a two-byte hexadecimal field in your program that contains the necessary information.

- (r) indicates the number of the register that contains the address of the two byte field with the necessary information.

- number,number provides first, the sum of the decimal digits to be printed (from 1 through 15) and second, the decimal line number to be marked (from 1 through 33).

ESP,n1,n2: n1 specifies that the current document should be ejected immediately and routed to stacker 1 or 2. (The valid entries are 1 and 2). A request for timing mark check can also be made in this parameter. If the number of timing marks on the document disagrees with the number you specify, either a nonrecovery error or timing mark check error occurs. You can specify the number of timing marks, by using parameter n2, in three ways:

- name specifies the name of a one-byte hexadecimal field in your program that indicates the number of timing marks that should be on the document.

- (r) specifies the number of the register that contains the address of the one-byte hexadecimal field containing the expected number of timing marks.

- number is a decimal number from 0 through 33 specifying the number of timing marks there should be on the document.

  If the number of timing marks is not specified or if zero is specified, no timing mark check is performed.

CHNG Macro

Name	Operation	Operand
[name]	CHNG	SYSnnn

This macro is provided only for Basic Programming Support and Basic Operating System upward compatibility. No code is generated from this macro. In DOS/VS tape channel switching is handled automatically by PIOCS.

ERET Macro

Name	Operation	Operand
[name]	ERET	$\left\{\begin{array}{l}\underline{\text{SKIP}}\\ \text{IGNORE}\\ \text{RETRY}\end{array}\right\}$

This macro enables your program's ERROPT or WLRERR routine to return to IOCS and specify an action to be taken. The macro applies only to DTFMT, DTFSD and DTFDU 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. For disk or diskette output, an ERET SKIP is treated as an ERET IGNORE.
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. With MTMOD for any error or with SDMOD wrong-length record errors, RETRY cancels the job with an invalid SVC message.

PRTOV Macro

Name	Operation	Operand
[name]	PRTOV	$\left\{\begin{array}{c}\text{filename}\\ (1)\end{array}\right\},\left\{\begin{array}{c}9\\ 12\end{array}\right\}$ $\left[,\left\{\begin{array}{c}\text{routine-name}\\ (0)\end{array}\right\}\right]$

The PRTOV (printer overflow) macro is used with a printer file to specify the operation to be performed when a carriage overflow condition occurs. To use this macro, the PRINTOV=YES operand must be included in the DTFPR or DTFSR.
PRTOV requires either two or three parameters. The first parameter must be the filename, written 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. The third parameter is required if you prefer to branch to your own routine on an overflow condition, rather than skipping directly to channel 1. It specifies the name of the routine, as a symbol or in

register notation. However, the name should never be preloaded into register 1.
If you specify the third parameter, IOCS does not restore the carriage to channel 1. In your routine you may issue any IOCS macro to perform whatever functions desired. If IOCS macros are used in the routine, register 14 must be saved. The CNTRL macro cannot be issued to the file unless CONTROL=YES is specified in the DTF. For example, you can print total lines, skip to channel 1, and print overflow page headings. At the end of the routine, return to IOCS by branching to the address in register 14.
The PRTOV macro causes a skip to channel 1, or branches to your 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 which causes carriage movement (PUT or immediate CNTRL), you should issue a PRTOV macro before issuing the next CNTRL or PUT. This ensures that your overflow option is executed at the correct time.
On the 3525 card punch, a channel 9 test indicates print line 17. A channel 12 test indicates print line 23. An overflow condition from either of these channels causes:

- a transfer of control to the overflow routine specified in the PRTOV macro, or
- a skip to channel one to begin printing on the next card for print only files.

When the PRTOV macro is used on a 3525 2-line printer, the result of the test is always negative since lines 17 and 23 are not available. The test is logically a no-operation.
Notes: PRTOV without the routine name option is invalid for 3525 associated files. A skip to channel one is valid only for 3525 print only files. PRTOV is not allowed for the 2560 or 5425.

Magnetic Reader Macros

Within a particular program, you should utilize either the GET macro or the READ, CHECK, WAITF combination. The READ, CHECK, and WAITF macros are described below. 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 (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 a GET macro is executed, no more GETs can be issued for the file. If an unrecoverable I/O error occurs when using the READ, CHECK, WAITF combination or when document processing for that file is complete, you 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 the document buffer in *Appendix E*).

READ Macro

The READ macro makes the next sequential buffer available to you, 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	$\left\{\begin{matrix}\text{filename}\\ (1)\end{matrix}\right\}$,MR

The first operand specifies the name of the file associated with the record to be read. This name is the same as that specified in the DTFMR header entry for the file, written as a symbol or in register notation. The second operand signifies that the file is for a magnetic ink character reader (MICR).

CHECK Macro

Name	Operation	Operand
[name]	CHECK	$\left\{\begin{matrix}\text{filename}\\ (1)\end{matrix}\right\}$ $\left[,\left\{\begin{matrix}\text{control address}\\ (0)\end{matrix}\right\}\right]$

The CHECK macro examines the buffer status indicators. A READ macro must therefore already have been issued to the file before a CHECK macro is issued.

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.

The second operand indicates the address to which control passes when a buffer is waiting for data or when the file is closed. Both parameters can be specified either as a symbol or in register notation.

CHECK determines 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 you to examine the posted error conditions before determining your action. (See byte 0, bits 2, 3, and 4, of the document buffer in *Appendix E.*) 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 if the file is closed, and the control address is not present, control is given to you at your 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 you 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*).

WAITF Macro

Name	Operation	Operand
[name]	WAITF	$\begin{Bmatrix}\text{filename1}\\ \text{(r1)}\end{Bmatrix} \left[,\begin{Bmatrix}\text{filename2}\\ \text{(r2)}\end{Bmatrix}\ldots,\begin{Bmatrix}\text{filenamen}\\ \text{(rn)}\end{Bmatrix}\right]$

The WAITF (wait multiple) macro is essential when processing in a multiprogramming 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.

One WAITF macro must be issued after a set of READ-CHECK combinations before your program attempts to return to a previously issued combination. Thus, the WAITF macro must be issued between successive executions of a particular READ macro.

The operands required are the names of the files waiting to be processed. The names are the same as those specified in the DTFMR header entries.

DISEN Macro

This macro 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. You should continue to issue GETs or READs until the unit exception bit (byte 0, bit 3), of the buffer status indicators is set on (see *Appendix E*).

Name	Operation	Operand
[name]	DISEN	$\begin{Bmatrix}\text{filename}\\ \text{(1)}\end{Bmatrix}$

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 specified either as a symbol or in register notation.

LITE Macro

Name	Operation	Operand
[name]	LITE	$\begin{Bmatrix}\text{filename}\\ \text{(1)}\end{Bmatrix} \left[,\begin{Bmatrix}\text{light switches}\\ \text{(0)}\end{Bmatrix}\right]$

This macro lights any combination of pocket lights on a 1419 magnetic character reader or 1275 optical reader/sorter. Before using the LITE macro, the DISEN macro must be issued to disengage the device. Processing of the documents should be continued until the unit exception bit (byte 0, bit 3) of the buffer status indicators is set on (see *Appendix E*). 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.

Bits	0	1	2	3	4	5	6	7	8	9	A	B	CDE	F
Pocket Lights	A	B	0	1	2	3	4	5	6	7	8	9	Reserved with binary zeros	Error indicator bit

Figure 2-32 Bit configuration for pocket light switch area of the 1419

The first operand is the name of the file; this name is the same as that specified for the DTFMR header entry for the file. 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.

The bit configuration for the pocket light switch area is shown in Figure 2-32. 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.

Optical Reader Macros—1287

GET Macro

See *GET Macro*, above in this chapter.

CNTRL Macro

See *CNTRL Macro*, above in this chapter.

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). You 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.

DSPLY 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, you must determine the load format CCW address. The third parameter specifies a general-purpose register (2-12) into which you place 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, you must determine whether the operator was able to recognize the unreadable line of data.

Note: When using the DSPLY macro, you 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 causes the next sequential 1287 or 1288 optical reader (document mode only) record to be read.

Name	Operation	Operand
[name]	READ	{filename / (1)} ,OR, {name / (r)}

The first parameter is the symbolic name specified in the DTFOR header for the file; it is always required. The parameter OR is required to indicate an optical character reader. The parameter name is always required; it specifies the address of the CCW list which you provide to be used to read a document from the 1287 or 1288. The register entry may be used in this parameter to provide the address of the CCW list. The first CCW in the list must not 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 CCW list addressed by the READ macro. This technique results in increased processing throughput. This method is preferable to using the CNTRL macro for document control.

The WAITF macro must be issued after the READ macro and before the program attempts to process an input record of the file.

RESCN Macro

Name	Operation	Operand
[name]	RESCN	{filename / (1)} , (r1) , (r2) [,n1] [,n2]

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.

Notes: 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. When 1288 unformatted fields are read the RESCN macro should not be used.

The first parameter specifies the symbolic name of the 1287D file as specified in the DTFOR header entry for the file. The second parameter specifies a general-purpose register (2-12) into which the 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, you must determine the load format CCW address.

The third parameter specifies a general-purpose register (2-12) into which the 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 you 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 which 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 RESCN macro, you 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.

RDLNE Macro

Name	Operation	Operand
[name]	RDLNE	{filename / (1)}

The RDLNE macro provides selective on-line correction when processing journal tapes on the 1287 optical reader. This macro reads a line in the on-line correction mode while processing in the off-line correction mode. RDLNE should be used in the COREXIT routine only, or else the line following the one in error will be read in on-line correction mode.

If the 1287 cannot read a character, IOCS first resets the input area to binary zeros and then retries the line containing the character which could not be read. If the read is unsuccessful, you are informed of this condition via your 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 cannot be read, the character is displayed on the 1287 display scope. The operator keys in the correct character, if possible. If the operator cannot readily identify the defective character, he may enter the reject character in the error line. This condition is posted in filename+80 for your examination. Wrong-length records and incomplete read conditions are also posted in filename+80.

This macro requires only one parameter, the symbolic name of the 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 is used to ensure that the transfer of a 1287 or 1288 optical reader record (document mode only) is completed. It requires 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 accomplishes all checking for read errors on the 1287 or 1288 file and exits to your COREXIT routine for handling of these conditions, if necessary.

Optical Character Reader Macros-3886

READ Macro

Name	Operation	Operand
[name]	READ	{filename / (1)} ,DR, {name / (r) / number , number}

The READ macro reads one line of data from the document.

Note: You must not issue any I/O macros to the 3886 between READ and WAITF macros. This would cause any errors detected during the read operation to be lost and the COREXIT routine would not be entered for that error.

The first parameter specifies the name of the DTFDR macro for the file, filename, or indicates that the address of the DTFDR is in register one, (1).

The second parameter, DR, is a required parameter that indicates a 3886 Optical Character Reader is the input device.

The third parameter specifies the line number to be read and the format record for the line in one of three ways:

- name provides the symbolic address of a two-byte hexadecimal field containing the line number in the first byte and the format record number in the second byte.
- (r) provides the register number that contains the address of the two-byte hexadecimal field.
- number,number provides the decimal line number to be read (1-33), followed by the format record number used to read the line (0-63).

Note: The line number specified must always be equal to or greater than the line number specified for the previous read operation on the current document; otherwise, a permanent error occurs.

WAITF Macro

Name	Operation	Operand
[name]	WAITF	{filename (1)}

The WAITF macro is used to ensure that an I/O operation is completed before execution continues. If the operation is not completed when the WAITF macro is issued, the active partition is placed in a wait condition until the I/O operation is completed. The completed operation is then tested for errors. If no errors were detected, control is returned to the next instruction in your program.

If an error occurs during the I/O operation, control is passed to the COREXIT routine. If an I/O operation is requested, no more documents are available, and the end-of-file key has been pressed, control is given to the end-of-file routine.

The only parameter of the WAITF macro specifies the name of the DTFDR macro for the file, filename, or indicates the address of the DTFDR is in register one (1).

CNTRL Macro

See *CNTRL Macro* above in this chapter.

SETDEV Macro

Name	Operation	Operand
[name]	SETDEV	{filename (1)} , {phasename (r)}

The SETDEV macro changes format records during execution of the program. When the new format record has been loaded into the 3886, control returns to the next sequential instruction in your program. If the operation is not successful, the completion code is posted at EXITIND and control is passed to the COREXIT routine, or the job is canceled. If you issue the SETDEV macro and no documents remain to be processed and the end-of-file key has been pressed on the device, control is passed to the end-of-file routine.

The first parameter specifies the name of the DTFDR macro, filename, for this file, or indicates the address of the DTFDR is in register 1, (1).

The second parameter specifies the name of the format record to be loaded, phasename; or indicates the register containing the address of an 8-byte area that contains the phasename, (r).

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, the work file macros READ, WRITE, and CHECK may be used. Also, if NOTEPNT=YES is specified, the work file macros NOTE, POINTR, POINTW, and POINTS may be used. Work files may contain only fixed-length unblocked records and undefined record formats. Tapes written in ASCII mode cannot be used for work files.

A tape work file is a single-volume file used for both input and output. 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, OPENR or CLOSE or CLOSER processing. Work files are specified by the DTFMT and MTMOD TYPEFLE=WORK operand, and are accessed by the READ/WRITE and CHECK macros.

The first time the volume for a work file is opened, it is opened as an output file. OPEN or OPENR examines the tape to determine whether it contains standard labels. The DTFMT FILABL operand, if present, 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. Job control label information cards are not required. If the tape does not already contain standard labels, labels are not created for the tape. Trailer labels are not processed.

If a work file with standard labels is reopened, OPEN or OPENR 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 you specified in the EOFADDR operand of DTFMT.

Disk work files are supported as single-volume single-pack files. They are always opened as output files. You must supply standard label information.

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 DTFSD DELETFL=NO operand must not be used. OPEN or OPENR creates a format 1 label for the file, and CLOSE or CLOSER 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 statement must not be the current date. The DTFSD DELETFL=NO operand must be specified. OPEN or OPENR creates a format 1 label, but CLOSE or CLOSER does not delete it. Thus, the file can be saved until the expiration date is reached.

Deleting an Unexpired File: When you try 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 or OPENR creates a label for the new file and the job continues.

READ Macro

The READ macro makes the next sequential record, or part of it, available to you. The record is read into the area of virtual storage indicated by the third operand.

The DTF READ=FORWARD or BACK operand should specify the direction of reading for tape.

The CHECK macro must be issued after the READ macro and before your program attempts to process an input record for the file.

Name	Operation	Operand
[name]	READ	$\left\{\begin{matrix}\text{filename}\\(1)\end{matrix}\right\}$,SQ,$\left\{\begin{matrix}\text{area}\\(0)\end{matrix}\right\}$ $\left[,\left\{\begin{matrix}\text{length}\\(r)\\S\end{matrix}\right\}\right]$

The first parameter specifies the name of the file from which the record is 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 specified as a symbol or in register notation.

The second parameter, SQ (for sequential), is always required.

The third parameter, 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 fourth parameter, length, 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), this parameter must not be specified in the READ macro. In this case, the number of characters to be read is specified in the BLKSIZE operand. You can change this number (which is stored in the DTF table) at any time by referencing the halfword filenameL.

WRITE Macro

The WRITE macro writes a record from the indicated area into the file named. The record is stored following the last record written in this file.

The CHECK macro must be issued after the WRITE macro to allow for completion of the input/output operation.

Name	Operation	Operand
[name]	WRITE	$\left\{\begin{matrix}\text{filename}\\(1)\end{matrix}\right\}$, $\left\{\begin{matrix}\text{SQ}\\\text{UPDATE}\end{matrix}\right\}$, $\left\{\begin{matrix}\text{area}\\(0)\end{matrix}\right\}$ $\left[,\left\{\begin{matrix}\text{length}\\(r)\end{matrix}\right\}\right]$

The first parameter specifies the name of the file to which the record is 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 written 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. A CLOSE or CLOSER macro (following an update write) protects the updated file by not writing an end-of-file record. If SQ is specified and a CLOSE or CLOSER immediately follows an OPEN or OPENR (no formatting WRITE commands were issued), an end-of-file record is not written.

The third parameter, area, specifies the name, as a symbol or in register notation, of the output area used by the file.

The fourth parameter, length, 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. You 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.

CHECK Macro

Name	Operation	Operand
[name]	CHECK	{filename / (1)}

This macro must be used after each READ or WRITE. It prevents 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 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 obtains identification for a physical record that is read or written during processing. 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 physical record number is returned in binary in the three low-order bytes of register 1. The high-order byte contains binary zero.

You must store the identification so that it can be used later in either a POINTR or POINTW macro.

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 cchr, where

- cc = Cylinder number,
- h = Track number,
- r = Record number within the track.

cc, h, and r are binary numbers. If NOTE follows a READ or WRITE to a disk file, the unused space remaining on the track following the end of the identified record is returned in register 0 as the binary number 00nn.

You must construct a six-byte field and store in it the identification of the record and the remaining track capacity (in the form cchrnn) so that it can be used later in a POINTR or POINTW macro to find the noted record again. The nn of cchrnn is needed only for WRITE SQ.

POINTR Macro

Name	Operation	Operand
[name]	POINTR	{filename / (1)} , {address / (0)}

The POINTR macro repositions the file for reading a record identified by the NOTE macro.

For magnetic tape, address specifies a 4-byte virtual 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. 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 four-byte virtual storage location containing the required record identification. If WRITE SQ is used, two bytes containing the remaining track capacity must also be supplied. The disk address can be expressed as a symbol or in register notation. The four- or six-byte number must be supplied in the form obtained from the NOTE macro (cchr or cchrnn, where nn is the length remaining on the track). POINTR repositions the file to read the record identification 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 macros following a NOTE macro. If a POINTR macro is used and the work file records are in undefined format, there may be occasions when a replacement record longer than 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 repositions a file to write a record.

For magnetic tape, address specifies a four-byte virtual 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 four-byte virtual storage location containing the required record identification. If WRITE SQ is used, two bytes containing the remaining track capacity must also be supplied. The disk address can be expressed as a symbol or in register notation. The four- or six-byte number must be supplied in the form obtained from the NOTE macro (cchr or cchrnn, 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 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, in which case the record identified by the NOTE is the record read.

Some programs using disk work files may include multiple WRITE macros following a NOTE macro. If a POINTW macro 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, 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 repositions a file to its beginning.

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 work-file 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

The completion macros CLOSER and CLOSE are used after the processing of a file is completed. These macros end the association of the logical file declared in your program with a specific physical file on an I/O device. CLOSER or CLOSE must be issued to deactivate all files (with the exception of DTFCN files, for which CLOSER and CLOSE must not be issued--no deactivation of DTFCN files is necessary).

Included here under the category of completion macros are the FEON and FEOVD macros, which are used to force an end-of-volume condition for magnetic tape or disk, respectively. Use of these macros does not preclude the requirement to issue CLOSER or CLOSE when processing of a file is complete.

A description of these completion macros follows.

FEOV Macro

Name	Operation	Operand

[name]	FEOV	{filename}
		{(1)}

The FEOV (force end-of-volume) macro is used for either input or output files on magnetic tape (programmer logical units only); for system logical units see *SEOV Macro* in the *Physical IOCS Macros* chapter) 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.

The name of the file, specified in the header entry, is the only parameter required. The name can be specified either as a symbol or in register notation.

When LIOCS macros 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 you to check any user-standard header labels if LABADDR is specified. If nonstandard labels are specified (FILABL=NSTD), FEOV allows you 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 header label(s) on the new volume (as specified in the DTFMT REWIND, FILABL, LABADDR operands, and the ASSGN cards). If nonstandard labels are specified, FEOV allows you 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 / (1)}

The FEOVD (force end-of-volume for disk) macro 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 for 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 for 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 for 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(R) macro, the end-of-volume marker is rewritten as an end-of-file marker, and the file is closed as usual.

CLOSE and CLOSER Macros

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)}]

The CLOSER or CLOSE macro must be issued to deactivate any file which was previously opened. Console files, however, cannot be closed.

When CLOSER is specified, the symbolic address constants that CLOSER generates from the parameter list are self-relocating. When CLOSE is specified, the symbolic address constants are not self-relocating.

To write the most efficient code in a multiprogramming environment it is recommended that CLOSER be used.

A CLOSE or CLOSER normally deactivates an output file by writing an EOF record and output trailer labels, if any. CLOSE(R) 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.

For diskette files, CLOSE or CLOSER sets the multivolume indicator in the HDR1 label to indicate the last volume of the file. Then, it sets up the end-of-data address in the HDR1 label and feeds the last diskette, determined by the FEED operand in the DTF.

After a CLOSE or CLOSER, 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.

Enter the symbolic name of the file (DTF filename) in the operand field. A maximum of 16 files may be closed by with one CLOSE or CLOSER by entering the filenames as additional operands. macro. Alternately, you can load the address of the filename into a register and specify the register using ordinary register notation. The high-order 8 bits of this register must be zeros. For CLOSER, the address of filename may be preloaded into any of the registers 2-15. For CLOSE, the address of filename may be preloaded into register 0 or any of the registers 2-15.

Notes:

- If you use register notation, we recommend that you follow the standard practice of using only registers 2-12.
- If CLOSE(R) is issued to a magnetic tape input file that has not been opened, the option specified in the DTF REWIND operand is performed. If CLOSE(R) 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(R) checks for the successful completion of the last operation.

For the 2560, 3525, or 5425, when CLOSE(R) is issued for a file it must also be issued for any associated files without any intervening input/output operations. Reopening one associated file requires reopening the others.

For 2560 or 5425 read associated files, the last card must not be punched or printed. When a read file (single or associated) is closed the last card read will be selected into the output stacker when 2560 "unit exception" has occurred--that is, when there is no following card. Two extra feed cycles are executed to perform this. When a punch or print file (without an associated read file) is closed, LIOCS performs one feed cycle to select the last card into the output stacker. When an associated punch-print file is closed, LIOCS performs one feed cycle to select the last card into the output stacker; if a print PUT was not specified for the last card, LIOCS executes the punch PUT before performing one feed cycle to select the card into the output stacker.

When O or R have been included in the parameter specified in the DTFCD MODE operand for a 3504, 3505 or 3525 running batched jobs, a non-data card must follow the card which causes your program to close the file.

For the 3525, Figure 2-33 shows the card movement caused by issuing CLOSE(R).

File Type	Feed Caused by Close of:
Read	Read*
Punch	Punch
Print	Print
Read/Print	Print*
Read/Punch/Print	Print**
Read/Punch	Punch**
Punch/Print	Print
Punch/Interpret	Punch

* A card feed us executed only if R has been specified in the DTFCD MODE operand. Programs using read column eliminate mode must detect an end-of-file condition themselves.

** Delimiter cards cannot be punched or printed in these files. CLOSE or CLOSER always issues a feed command.

Figure 2-33 **CLOSE or CLOSER card movement for the 3525**

PART 3

DIRECT ACCESS METHOD

CONCEPTS OF DAM

With DAM you can process DASD records in random order. You specify the address of the record to IOCS and issue a READ or WRITE macro to transfer the specified record.

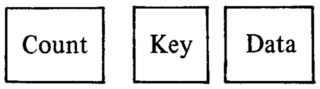
Variations in the parameters of the READ or WRITE macros permit records to be read, written, updated, or replaced in a file.

Whenever DAM is used, the file must be defined by the declarative macro DTFDA (**D**efine **T**he **F**ile for **D**irect **A**ccess). The detail entries for this macro are described in the *Declarative Macros* section later in this chapter. In order to understand the use of some of these entries, however, it is first necessary to indicate how DAM processing uses them.

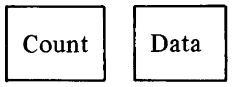
Record Types

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 the *DOS/VS Data Management Guide*, GC33-5372.

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.

When the DTFDA KEYLEN operand 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.

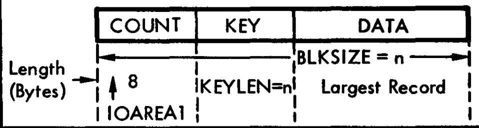
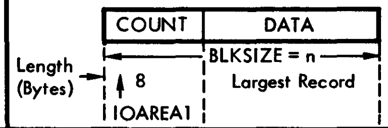
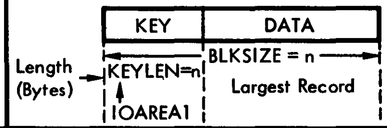
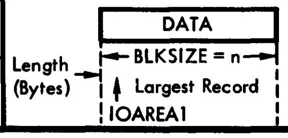
IOCS considers all records as unblocked; if you want blocked records, you must perform your own blocking and deblocking. Records are also considered to be either fixed, variable, or undefined length. A spanned record indicates variable blocks where the size of each segment is a function of the track size and record size. The record size is set by a formatting WRITE macro (WRITE AFTER). 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 RECFORM operand. Whenever records specified as undefined are written, you must determine the length of each record and load it in a register (specified by the DTFDA RECSIZE operand) before issuing the WRITE macro for that record.

IOAREA Specification

The DTFDA IOAREA1 operand defines an area of virtual 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 operands: AFTER, KEYLEN, READID, WRITEID, READKEY, and WRITEKY. Figure 3-1 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 operand. 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.

DTFDA MACRO ENTRIES						I/O AREA DEFINED
AFTER	KEYLEN	READID	WRITEID	READKEY	WRITKY	
X	X	●	●	●	●	 COUNT KEY DATA Length (Bytes) 8 KEYLEN=n BLKSIZE = n Largest Record IOAREA1
X		●	●			 COUNT DATA Length (Bytes) 8 BLKSIZE = n Largest Record IOAREA1
	X	□	□	●	●	 KEY DATA Length (Bytes) KEYLEN=n BLKSIZE = n Largest Record IOAREA1
		□	□			
	X			□	□	

X = Specified

● = May also be specified

□ = Of two entries, one <u>and/or</u> the other is specified

Figure 3-1 **I/O areas resulting from different DTFDA macro entries for fixed unblocked and undefined record formats**

Contents

Figures 3-1 and 3-2 give a summary of what the contents of IOAREA1 are for the various types of DTF macros. These contents are provided by, or to, IOCS when an imperative macro is issued. When you build a record, you must place the contents shown in Figures 3-1 and 3-2 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 3-1, the data would be located to the right of the count or key area.

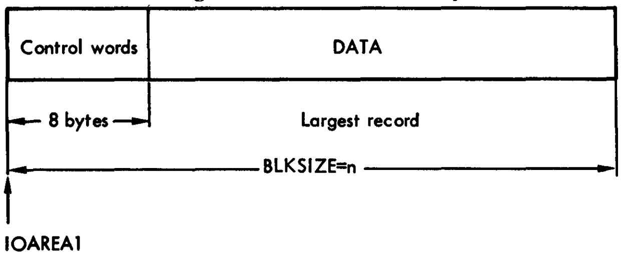


Figure 3-2 **I/O areas for variable length and spanned unblocked DTFDA record format**

As oppposed to fixed unblocked and undefined records, the IOAREA1 for variable length and spanned unblocked records is independent of the DTFDA macro entries. If you specify the KEYLEN entry of the DTFDA macro, the key is transmitted to or from the field you specified on the KEYARG entry. The count field, if desired, is taken from an area reserved automatically by logical IOCS.

The control words are built by logical IOCS except for the case when you create your file or add records to it by using the WRITE AFTER macro. You must, in that case, insert the data length of the record (plus four) into the 5th and 6th bytes of the control words. When you read a variable length or spanned unblocked record these bytes will contain the length of the record. When updating records you should not change any parts of the control words.

The maximum length of a logical record plus its key and control words, if any, is shown in Figure 3-3.

Device	RECFORM	
	FIXUNB VARUNB UNDEF	SPNUNB
2311	3625	32767
2314, 2319	7294	32767
2321	2000	32767
3330/3333	13,030	32767
3340	8,535	32767

Figure 3-3 **Maximum length of DTFDA records including key and control words**

Reference Methods

With DAM, 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. If a specified record is not found, IOCS sets a no-record-found indication in your error/status byte specified by the DTFDA ERRBYTE operand. This indication can be tested and appropriate action can be taken to suit your 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 the no-record-found bit in ERRBYTE.

When an I/O operation is started, control returns immediately to your 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.

After a READ or WRITE macro for a specified record has been executed, IOCS can make the ID of the next record available to your program. The WAITF macro should be issued to assure that the data transfer is complete. You must set up a field (in which IOCS can store the ID) to request that IOCS supply the ID. You must also specify the symbolic address of this field in the DTFDA IDLOC operand.

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 you are processing consecutively on the basis of the next ID and do not have an end-of-file record, you can check the ID supplied by IOCS against your file limits to determine when the end of the file has been reached.

Track Reference

To provide IOCS with the track reference, you set up a track reference field in virtual storage, assign a name in the DTFDA SEEKADR operand, and determine by DTFDA operand specifications which type of addressing system to use. Before issuing any READ or WRITE macro for a record, you must store the proper track identifier in either the first seven hexadecimal bytes (mbbcchh), the first three hexadecimal bytes (ttt), or the 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 3-4. For reference to records by record number, r or rr is used (see *Identifier (ID) Reference*, below). When the READ or WRITE is executed, IOCS refers to this field to select the specific track on the appropriate DASD.

Record Reference

DAM allows records to be specified by either record key or record identifier.

Key Reference

If records contain key areas, the records on a particular track can be randomly searched by their keys. This allows you 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 you must specify the name of a key field in virtual storage in the DTFDA KEYARG operand. You then store each desired key in this field.

Identifier (ID) Reference
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 physical 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. You may, however, use the relative track notation instead of cylinder and head notation if specified in the DSKXTNT and RELTYPE operands. 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 (see Figure 3-4) must contain the number of the desired record. When a READ or WRITE macro 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

Your 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 the IBM-supplied Clear Disk utility program. The Clear Disk utility also resets the capacity record to reflect an empty track.

In addition to reading, writing, and updating records randomly, DAM permits you 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 macro 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 your error/status byte specified

Bytes	Decimal Identifier	Contents in Zoned Decimal	Information
0-7	ttttttt	0-16,777,215	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 3-4 **Types of track reference fields (part 1 of 2)**

Bytes	Physical Identifier	Contents in Hexadecimal	Information
0	m	00-FF	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 result 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	bb	0000 (disk) 0000-0009 (2321)	For 3221 the first byte is zero. The cell number (0-9) is specified in the second byte. These two bytes are always zero for disk references.

3-4	cc	0000-00C7 (2311,2314,2319)
		0000-0193 (3330,3333)
		0000-015B (3348 model 35)
		0000-02B7 (3348 model 70)
		0000-1309 (2321)
			For disk, the maximum number of the cylinder in which the record can be located is:
			for 2311, 2314, 2319 : 199
			for 3330, 3333 : 403
			for 3340 with 3348 model 35 : 347
			for 3340 with 3348 model 70: 695
			These two bytes (cc), together with the next two (hh), provide the track identification. DAM does not permit the use of different data module sizes in a multivolume file on a 3340. For 2321, the number of 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	hh	0000-0009 (2311) 0000-0013 (2314) 0000-0012 (3330) 0000-0012 (3333) 0000-000B (3340) 0000-0413 (2321)	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 specifies one of the five head bar positions (0-4, equivalent to cylinders on disk). The second byte specifies one of the twenty head elements (0-19).
7	r	0-FF	Sequential number of the record on the track. **Note:** r=0 if reference is by key.

Figure 3-4 **Types of track reference fields (part 2 of 2)**

by the DTFDA ERRBYTE operand. 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 3-5) has a count area and a data area, and contains the following:

Count Area

Flag (normally not transferred to virtual storage)

Physical Identifier

Key Length (KL)

Data Length (DL)

Data Area

Physical ID of last record written on track (cchhr)

Number of unused bytes remaining on track

Flag used only for operating systems other than DOS/VS

Each time a WRITE AFTER macro is executed, IOCS updates the data area of this record.

Additional Information

For more information about DAM processing--such as the organization of overflow areas--see the *DOS/VS Data Management Guide*, GC33-5372.

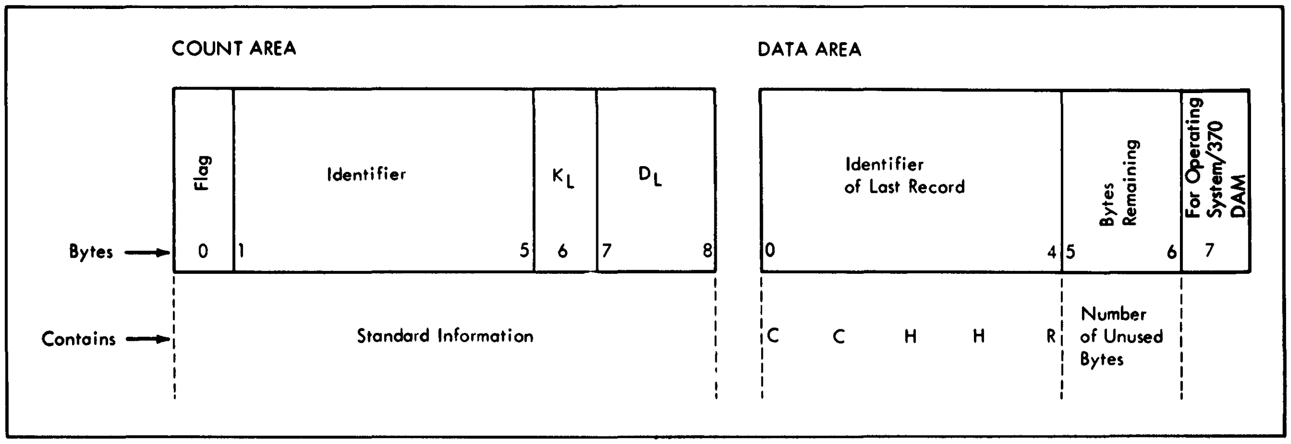


Figure 3-5 **Contents of record 0 for capacity-record option**

DECLARATIVE MACROS

DAM files must first be defined by the declarative macros before the imperative macros, described later in this chapter, are used to operate on them.

There are two related types of declarative macros--DTFDA and DAMOD. These two macros are described below.

DTFDA Macro

Enter the symbolic name of the file (filename) in the name field and DTFDA in the operation field. The detail entries follow the DTFDA header card in any order. Figure 3-6 lists the keyword operands contained in the operand field.

DTFDA should not be used to define SYSIPT if the program may be invoked by a catalogued procedure and SYSIPT contains data. In this case, the program must process the data sequentially, and the DTFDI macro should be used. This macro is described in Part 2: *Sequential Access Method*.

Applies to				
Input	Output			
x	x	M	BLKSIZE=nnnn	Length of one I/O area, in bytes
x	x	M	DEVICE=nnnn	(2311, 2314, 2321, 3330, 3340). If omitted, 2311 is assumed
x	x	M	ERRBYTE=xxxxxxxx	Name of 2-byte field for error/status codes supplied by IOCS
x	x	M	IOAREA1=xxxxxxxx	Name of I/O area
x	x	M	SEEKADR=xxxxxxxx	Name of track-reference field
x	x	M	TYPEFLE=xxxxxx	(INPUT or OUTPUT)
	x	O	AFTER=YES	WRITE filename, AFTER or WRITE filename, RZERO macro is used for this file
x	x	O	CONTROL=YES	CNTRL macro is used for this file
x	x	O	DEVADDR=SYSnnn	Symbolic unit required only when no extent statement is provided
x	x	O	ERREXT=YES	Nondata transfer errors are to be indicated in ERRBYTE
x		O	FEOVD=YES	Support for sequential disk end of volume records is desired
x		O	HOLD=YES	Employ the track hold function
x	x	O	DSKXTNT=n	Indicates the number (n) of extent for a relative ID

M=Mandatory; O=Optional

Figure 3-6 DTFDA macro (part 1 of 2)

Applies to

Input	Output			
x	x	O	IDLOC=xxxxxxxx	Name of field in which IOCS stores the ID of a record
x	x	O	KEYARG=xxxxxxx	Name of key field if READ filename, KEY or WRITE filename, KEY or WRITE filename, AFTER macro is used for this file
x	x	O	KEYLEN=nnn	Number of bytes in record key if keys are to be processed. If omitted, IOCS assumes zero (no key)
x	x	O	LABADDR=xxxxxxxx	Name of your routine to check/write user labels
x	x	O	MODNAME=xxxxxxx	Name of DAMOD logic module for this DTF. If omitted, IOCS generates standard name
x	x	O	RDONLY=YES	Generates a read-only module. Requires a module save area for each task using the module
x		O	READID=YES	READ filename, ID macro is used for this file
x		O	READKEY=YES	READ filename, KEY macro is used for this file
x	x	O	RECFORM=xxxxxx	(FIXUNB, SPNUNB, VARUNB, or UNDEF). If omitted, FIXUNB is assumed
x	x	O	RECSIZE=(nn)	Register number if RECFORM=UNDEF
x	x	O	RELTYPE=xxx	(DEC or HEX). Indicates decimal or hexadecimal relative addressing
x	x	O	SEPASMB=YES	DTFDA is to be assembled separately
	x	O	SRCHM=YES	Search multiple tracks, if record reference is by key
	x	O	TRLBL=YES	Process trailer labels, LABADDR must be specified
	x	O	VERIFY=YES	Check disk records after they are written. For DEVICE=2321, YES is assumed
x	x	O	WRITEID=YES	WRITE filename, ID macro is used for this file
x	x	O	WRITEKEY=YES	WRITE filename, KEY macro is used for this file
x	x	O	XTNTXIT=xxxxxxxx	Name of your routine to process extent information

M=Mandatory; O=Optional

Figure 3-6 DTFDA macro (part 2 Of 2)

AFTER=YES
This operand must be included if any records (or an additional record) are written in a file by a formating 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 WRITE filename,AFTER; or WRITE filename,RZERO; is used in a program, this operand is required.

BLKSIZE=n
This operand indicates the size of the I/O area by specifying the maximum number of characters that are transferred to or from the area at any one time. When undefined, variable length or spanned records are read or written, the area must be large enough to accommodate the largest record.

For details on how to compute n, see *IOAREA Specification.*

IOCS uses this specification to construct the count field of the CCW for reading or writing records.

CONTROL=YES
Include this operand if a CNTRL macro is issued for this file. The CNTRL macro for seeking on a disk allows you to specify a track address to which access movement should begin for the next READ or WRITE macro. While the arm is moving, you may process data and/or request I/O operations on other devices.

For the 2321, the CNTRL macro enables you 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 symbolic unit is not provided via an EXTENT job control statement. 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 ASSGN job control statement to assign the actual I/O device address to the file.

Note: EXTENT job control statements provided for DAM must be supplied in ascending order, and the symbolic units for multi-volume files must be assigned in consecutive order.

DEVICE={<u>2311</u> | 2314 | 2321 | 3330 | 3340}
This operand specifies the device on which the file is located. Specify 2314 for 2319 and 3330 for 3333. If this operand is omitted, 2311 is assumed.

Note: DAM does not permit the use of different size data modules (on a 3340) for a multivolume file.

DSKXTNT=n
This operand indicates the maximum number of extents (up to 256) that are specified for a file. When RECFORM=FIXUNB, VARUNB, or UNDEF is specified with this operand, 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, 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 your program. The name of a 2-byte field (in which IOCS can store the error-condition or status codes) is entered.

The ERRBYTE codes are available for testing by your program after the attempted transfer of a record is complete. You must issue the WAITF macro before you interrogate the error status information. After testing the ERRBYTE status code, your program can return to IOCS by issuing another macro. One or more of the error status indication bits may be set to 1 by IOCS as in the bits shown in Figure 3-7.

ERREXT=YES
This operand enables irrecoverable I/O errors (occurring before a data transfer takes place) to be indicated to your program. This error information is indicated in the bytes named in the ERRBYTE operand and is available after the WAITF macro has been issued.

FEOVD=YES
This operand is specified if code is generated to handle end-of-volume records. It should be specified only when reading a file which was built using DTFSD and the FEOVD macro.

HOLD=YES
This operand can be specified only if the track hold function is specified

- at system generation time, and

- included in the DAMOD macro, and
- used 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 spanned records are processed, the track containing the first segment is held until you release it. If a formating 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 you want IOCS to supply the ID of a record after each READ or WRITE (ID or KEY) is completed. Specify the name of a record reference field in which IOCS is to store the ID. WAITF should be used before referencing this field.

IOCS supplies the ID in the same form as used in the SEEKADR location. The ID forms, given in Figure 3-4, 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 which includes leading zeros.

Byte	Bit	Error/Status Code Indication	Explanation
0	0	Not applicable	Not applicable
0	1	Wrong-length record	The wrong-length record indication is applicable to fixed-length, undefined length, variable-length, and spanned 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 form the specified length (BLKSIZE minus KEYLEN, or BLKSIZE minus KEYLEN plus 8 if AFTER=YES was specified). • A READ ID is issued, and the length of the record (including key if KEYLEN was specified) 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. **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 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 length of the record (including key if KEYLEN was specified) 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 KEYLEN plus 8 if AFTER=YES was specified). The length of the record written is equal to the maximum data size.

Figure 3-7 **ERRBYTE error status indication bits (part 1 of 4)**

Byte	Bit	Error Status Indication	Explanation
0	1	Wrong-length record (continued)	• 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. **Variable-length records:** This bit is set on under the following conditions: • When a READ is issued and the LL count[1] is greater than the maximum value specified by the BLKSIZE operand. • When a nonformating 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. 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 formating WRITE is issued and the LL count[1] is greater than the maximum specified block size, the record is written with the low-order bytes truncated. **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 by BLKSIZE minus 8. Only the number of bytes specified is read. • When a nonformating WRITE is issued and the record length is not the same as that of the record being processed. 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 formating WRITE is issued and the logical record size is larger than the size specified with BLKSIZE minus 8, the record is truncated to the size specified. • If the first physical record encountered is not an only or first segment. 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.

[1] The LL count is contained in the first two bytes of the block descriptor and counts the length of the physical block including all control information. For more detail see *DOS/VS Data Management Guide*, GC 33-5372.

Figure 3-7 ERRBYTE error status indication bits (part 2 of 4)

Byte	Bit	Error/Status Code Indication	Explanation
0	2	Non-data-transfer error	The block in error was neither read or written. If ERREXT is specified and this bit is off, transfer took place and your program should check for other errors in the ERRBYTE field.
0	3	Not applicable	Not applicable
0	4	No room found	This 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. With spanned records the no-room-found condition is set if not at least one data byte will fit on the specified track in addition to the key, if any, and the 8 byte control field, or if any successive tracks required to transfer the record are not completely empty.
0	5	Not applicable	Not applicable
0	6	Not applicable	Not applicable
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 to Fs for a hexadecimal ID.
1	0	Data check in count area	This is an irrecoverable error.
1	1	Track overrun	The number of bytes on the track exceeds the theoretical capacity.
1	2	End of cylinder	This 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 (see *IDLOC operand*).
1	3	Data check when reading key or data	This is an irrecoverable error.
1	4	No record found	This indication is given when a search ID or key is issued and a record is not found. This applies to both READ commands 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 not the first or only segment, this indicator is set on.

Figure 3-7 **ERRBYTE error status indication bits (part 3 of 4)**

Byte	Bit	Error/Status Code Indication	Explanation
1	5	End of file	This 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 FFFF or, in the case of RELTYPE=DEC, zoned decimal 9's. The bit is set only after all the data records have been processed. For example, in a file having n data records (record n+1 is the end-of-file record), the end-of-file indicator is set on when you read the n+1 record. This bit is also posted when an end-of-volume marker is detected. It is your responsibility to determine if this bit means true EOF or end of volume on a SAM file.

1	6	End of volume	This 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:
			for 2311 cylinder 199, head 9
			for 2314 or 2319 cylinder 199, head 19
			for 3340 or 3333 cylinder 403 head 18
			for 3340 with 3348 model 35 cylinder 347, head 11
			for 3340 with 3348 model 70 cylinder 695, head 11
			for 2321 subcell 19, strip 5, cylinder 4, head 19
			These limits allow for the reserved alternate track area.
			
			If both the end of cylinder and EOV indicators are set on, the ID returned in IDLOC is FFFF or, in the case of RELTYPE=DEC, zoned decimal 9's.

1	7	Not applicable	Not applicable

Figure 3-7 **ERRBYTE error status indication bits (part 4 of 4)**

HOLD=YES
This operand can be specified only if the track hold function is specified

- at system generation time, and
- included in the DAMOD macro, and
- used 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 spanned records are processed, the track containing the first segment is held until you release it. If a formating 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 you want IOCS to supply the ID of a record after each READ or WRITE (ID or KEY) is completed. Specify the name of a record reference field in which IOCS is to store the ID. WAITF should be used before referencing this field.

IOCS supplies the ID in the same form as used in the SEEKADR location. The ID forms, given in Figure 3-4, 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 which includes leading zeros.

IOCS either supplies the ID of the record specified in the READ/WRITE macro, or the ID of the next record location. The following may occur when this option is taken:

- Whenever a READ or WRITE ID (or 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.

Figure 3-8 summarizes the IDLOC ID supplied under the various circumstances.

MACRO	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 3-8 ID supplied after a READ or WRITE macro

If IDLOC is specified and end of cylinder is reached on a disk, the cylinder number is increased by 1, the head number is set to 0, and the record number is set to 1. On the 2321, 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 to be set to 0, and the record number to be 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 your 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 name of the input/output area used for 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 storage.

The input/output area must be large enough to contain the maximum number of bytes required in any READ or WRITE macro issued for a file in your program. This is affected by the length of record data areas, and by the use of the count and key areas and control information as shown in Figures 3-1 and 3-2 and described under *IOAREA Specification.*

- If undefined records are specified in the DTFDA RECFORM operand, the area must provide space for the largest data record that will be processed.

- If variable or spanned records are specified in the DTFDA RECFORM operand, the area must be large enough to contain the largest record in the file, plus an additional eight bytes for control words. You must place the first byte of your record in the ninth byte of the I/O area for all write operations. You must also place the data length plus four in 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 KEYLEN operand is specified and any instructions that read or write the key area of a record are issued in your program, the input/output area (for records other than spanned and variable) must provide room for the key area as well as for 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 your program, eight bytes must be allotted at the beginning of the I/O area to enable IOCS to construct the count field which is to be transferred to disk (for records other than spanned and variable).

Whenever a WRITE macro is issued, IOCS assumes that the input/output area (see Figures 3-1 or 3-2) contains the information implied by the type of macro that is being executed.

KEYARG=name
This operand must be included if records are identified by key, that is, if the macro READ filename,KEY; or WRITE filename,KEY; is used in a program, this entry and the corresponding KEYLEN operand are required. KEYARG specifies the name of the key field in which you supply the record key for the READ/WRITE routines.

The KEYARG operand is required for formating WRITE (WRITE filename, AFTER) operations for files containing keys if RECFORM=VARUNB or SPNUNB. 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 of bytes in each key. All keys must be the same length. If this operand is omitted, IOCS assumes a key length of zero.

If there are keys recorded on DASD and this entry is absent, a WRITE ID or READ ID 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 I/O area is located (see the section *IOAREA Specification).*

LABADDR=name
You may require one or more user labels in addition to the standard file label. If so, you must include your own routine to check, or write, the labels. The name of such a routine is specified in this operand. IOCS branches to this routine after it has processed the standard label. See *Writing User Standard Labels on Disk* and *Checking User Standard Labels on Disk* in the *Label Processing* chapter for a discussion of what the LABADDR routine should do.

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, MODNAME must specify the same name as the DAMOD macro. If this entry is omitted, standard names are generated for calling the logic module. If two DTF macros 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 task should have its own uniquely defined save area. Each time an imperative macro (except OPEN, OPENR, or LBRET) is issued, register 13 must contain the address of the save area associated with the task. 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* in the *Multitasking Macros* chapter.

READID=YES
This operand must be included if any input records are specified by ID (identifier) in your program, that is, whenever READ filename,ID is used.

READKEY=YES
This operand must be included if any input records are specified by key in your program, that is, whenever READ filename,KEY is used.

RECFORM={FIXUNB | SPNUNB | UNDEF | VARUNB}
This operand specifies the type of records in the input or output file. The specifications are as follows:

FIXUNB	For fixed-length records. All records are considered unblocked. If you want blocked records, you must provide your 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 of undefined format.

VARUNB For variable-length records. This specification is for unblocked variable-length records.

For a definition of record formats see *DOS/VS Data Management Guide,* GC33-5372.

RECSIZE=(r)
This operand must be included if undefined records are specified (RECFORM=UNDEF). It specifies the number of the general-purpose register (2-12) that contains the length of each individual input or output record.

Whenever an 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, you must load the length of the data area of the record (in bytes) into this register, before you issue the WRITE macro for the record. IOCS adds the length of the key when required.

When records are written (AFTER specified in the WRITE macro), 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 to be used. When RECFORM=FIXUNB, VARUNB, or UNDEF, 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 name of your track-reference field. In this field, you store 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 3-4 for the types of track reference fields that can be used.

SEPASMB=YES
Include this operand only if the DTFDA 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 problem program and no CATALR card is punched.

SRCHM=YES
If records are identified by key, this operand may be included to cause IOCS to search multiple tracks for each specified record. The macro 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 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* chapter.

TYPEFLE={INPUT | OUTPUT}
This operand must be included to indicate how standard volume and file labels are to be processed. INPUT indicates that standard labels are to be read; OUTPUT indicates that standard labels are to be written.

For DASD files this entry is always required.

VERIFY=YES
This operand is included if you want to check the parity of disk records after they are written. If this operand is omitted, any records written on a disk are not verified. VERIFY is always assumed when 2321 records are written.

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 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 WRITE filename,KEY is used.

XTNTXIT=name
This operand is included if you want to process label extent information. It specifies the name of your extent exit routine. During an OPEN, IOCS branches to your routine after each specified extent is checked and validated. Upon entering your routine, IOCS stores in register 1 the address of a 14-byte field that contains the label extent information (in binary form). If user labels are present, the user label track is returned as a separate extent and the lower limit of the first normal extent is increased by one track. The format of this field is shown in Figure 3-9.

Return to IOCS by use of the LBRET macro. Registers 2 - 13 are available in the XTNTXIT routine. Within the routine you cannot issue a macro that calls a transient routine (such as OPEN, OPENR, CLOSE, CLOSER, DUMP, PDUMP, CANCEL, CHKPT, etc.).

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 binary number
13	Present binary number of the extent (B2)

Figure 3-9 **Label extent information field**

DAMOD Macro

Listed here are the operands you can specify for DAMOD. The first card contains DAMOD in the operation field and may contain a module name in the name field. The parameters are explained here and summarized in Figure 3-10.

AFTER=YES
This operand generates a logic module that can perform a formating 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, and
- included in the DTFDA macro, and
- used when the file is referenced.

For more information see the DTFDA HOLD operand.

ERREXT=YES
Include this operand if irrecoverable I/O errors (occurring before a data transfer takes place) are to be indicated to your program in the bytes named in the DTF ERRBYTE operand.

FEOVD=YES
This operand is specified if coding is to handle end-of-volume records. It should be specified only if you are reading a file built using DTFSD and the FEOVD macro.

IDLOC=YES
This operand generates a logic module that returns record identifier (ID) information to you. The module also processes any files in which the IDLOC operand is not specified in the DTF.

RDONLY=YES
This operand causes a read-only module to be generated. Whenever this operand is specified, any DTF used with the module must have the same operand.

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 non-data-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 3-10 **DAMOD macro**

RECFORM={<u>FIXUNB</u> | SPNUNB | UNDEF | VARUNB}
If UNDEF is specified, the logic module generated can handle both unblocked fixed-length and undefined records. If the operand 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
Include this operand only if the module is assembled separately. This causes a CATALR card with the module name (standard or user-specified) to be punched ahead of the object deck and defines the module name as an ENTRY point in the assembly. If the operand is omitted, the program assumes that the module is being assembled with the problem program and no CATALR card is punched.

Standard DAMOD Names
Each name begins with a 3-character prefix (IJI) and continues with 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 *IOCS Subset/Superset Names* in *The Macro System* chapter.

```
      + + + + +
I J I B A E H X
      F Z I P Z
      +   Z Z +
      S   + + W
      V   E H Y
          R R
          Z Z
```

+ Subsetting/supersetting permitted.

Notes:

1. The module IJIBAEHW will cause assembly error IPK154 TOO MANY ENTRY SYMBOLS. The valid entry points for this module total more than 100, which is the maximum for assembler language. Specify less parameters for DAMOD if you can. Otherwise, you must assemble your own module for your program.

2. Your 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.

IMPERATIVE MACROS

After the DAM files are defined by the declarative macros, the imperative macros can be used to operate on the files. The imperative macros are divided into three groups: those for initialization, processing, and completion.

Initialization Macros

The initialization macros OPENR or OPEN must be used to activate a DAM file for processing. These macros associate the logical file declared in your program with a specific physical file on a DASD. The association by OPENR or OPEN of your program's logical file with a specific physical file remains in effect throughout your processing of the file until you issue a CLOSE or CLOSER macro.

Included here under the category of initialization macros is the LBRET macro, which is connected only with label and extent processing. LBRET is used to return to IOCS from a subroutine of your program which writes or checks labels and extents.

OPEN and OPENR Macros

Op	Operand
for self-relocating programs	
OPENR	$\left\{\begin{matrix}\text{filename1}\\\text{(r1)}\end{matrix}\right\}$ $\left[,\left\{\begin{matrix}\text{filename2}\\\text{(r2)}\end{matrix}\right\}\ldots,\left\{\begin{matrix}\text{filenamen}\\\text{(rn)}\end{matrix}\right\}\right]$
for programs that are not self-relocating	
OPEN	$\left\{\begin{matrix}\text{filename1}\\\text{(r1)}\end{matrix}\right\}$ $\left[,\left\{\begin{matrix}\text{filename2}\\\text{(r2)}\end{matrix}\right\}\ldots,\left\{\begin{matrix}\text{filenamen}\\\text{(rn)}\end{matrix}\right\}\right]$

The OPENR or OPEN macro activates all files.

When OPENR is specified, the symbolic address constants that OPENR generates from the parameter list are self-relocating. When OPEN is specified, the symbolic address constants are not self-relocating.

To write the most efficient code in a multiprogramming environment it is recommended that OPENR be used.

Self-relocating programs using LIOCS must use OPENR to activate all files, including console files. In addition to activating files for processing, OPENR relocates all address constants within the DTF tables (zero constants are relocated only when they constitute the module address).

If OPEN or OPENR attempts to activate a LIOCS file (DTF) whose device is unassigned, the job is terminated. If the device is assigned IGN, OPEN or OPENR does not activate the file but turns on DTF byte 16, bit 2, to indicate the file is not activated. If DTF byte 16 bit 2 is on after issuing an OPEN or OPENR, input/output operations should not be performed for the file.

Enter the symbolic name of the file (DTF filename) 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, you can load the address of the DTF filename into a register and specify the register using ordinary register notation. The high-order 8 bits of this register must be zeros. If symbolic notation is used, you need to establish addressability through a base register. For OPENR, the address of filename may be preloaded into any of the registers 2-15. For OPEN, the address of filename may be preloaded into register 0 or any of the registers 2-15.

Note: If you use register notation, we recommend that you follow the standard practice of using only registers 2-12.

Whenever an input/output DASD file is opened and you plan to process user-standard labels (UHL only), you 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 OPENR or issue a separate OPEN or OPENR preceding the OPEN or OPENR for the file.

If an output file is created using DAM, 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(R) checks the standard VOL1 label and checks the extents specified in the extent cards for the following:

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 or OPENR checks all the labels in the VTOC to ensure that the created file does not destroy an existing unexpired file. OPEN or OPENR then creates the standard label(s) for the file and writes the label(s) in the VTOC.

If you wish to create your own user labels (UHL) for the file, include the DTF LABADDR operand. OPEN or OPENR reserves the first track of the first extent for these header labels and gives control to your label routine.

If the XTNTXIT operand is specified, OPEN or OPENR stores the address of a 14-byte extent information area in register 1. (See Figure 3-9 for the format of this area.) Then, OPEN or OPENR gives control to your extent routine. You 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 for use.

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 or OPENR checks the standard VOL1 label and then checks the file label(s) in the VTOC. OPEN or OPENR checks some of the information specified in the extent cards for that volume. If LABADDR is specified, OPEN or OPENR makes the user standard header labels available one at a time for checking.

If the XTNTXIT operand is specified, OPEN or OPENR stores the address of a 14-byte extent information area in register 1. (See Figure 3-9 for the format of this area.) Control is then given to your extent routine. You 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 use.

LBRET Macro

Name	Operation	Operand
[name]	LBRET	{1 | 2 | 3}

The LBRET macro is issued in your subroutines when you have completed processing labels or extents and wish to return control to IOCS. LBRET applies to subroutines that write or check DASD user standard labels or handle extent information. The operand used depends on the function to be performed. See the *Label Processing* chapter.

Checking User Standard DASD Labels: IOCS passes the labels to you one at a time until the maximum allowable number has been read and updated, or until you signify you want no more. In the label routine, use LBRET 3 if you want IOCS to update (rewrite) the label read and pass you the next label. Use LBRET 2 if you simply want IOCS to read and pass you 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 you want to eliminate the checking of one or more remaining labels, use LBRET 1.

Writing User Standard DASD Labels: Build the labels one at a time and use LBRET to return to IOCS to write the labels. Use LBRET 2 if you want control returned to you 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 you wish to stop writing labels before the maximum number is written.

Checking DASD Extents: When using the direct access method, you can process your extent information. After each extent is processed, you should use LBRET 2 to obtain the next extent.

Processing Macros

Once DAM files have been readied for processing with the initialization macros, the READ, WRITE, WAITF, and CNTRL macros described in this section may be used.

READ Macro

Name	Operation	Operand
[name]	READ	$\left\{\begin{matrix}\text{filename}\\ \text{(1)}\end{matrix}\right\}, \left\{\begin{matrix}\text{KEY}\\ \text{ID}\end{matrix}\right\}$

The READ and WAITF macros transfer a record from DASD to an input area in virtual storage. The input area must be specified in the DTFDA IOAREA1 operand.

The READ macro 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 file if the 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 specified 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 are undefined (RECFORM=UNDEF), DAM supplies the data length of each record in the designated register in the DTF RECSIZE operand.

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 macro must be the word KEY, and the READKEY operand must be specified in the DTFDA.

Whenever this method of reference is used, your program must supply the desired record key to IOCS before the READ macro is issued. For this, the key must be stored in the key field (specified in the DTFDA KEYARG operand). When the READ macro 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 input area.

Only the specified track is searched unless you request that multiple tracks be searched on each READ (by including the SRCHM operand in the DTFDA). 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 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 macro must be the letters ID, and the READID operand must be included in the DTFDA.

Whenever this method of reference is used, your program must supply both the track information and the record number in the track-reference field. When the READ macro 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 the DTFDA KEYLEN operand) and the data area of the record are transferred to key and data portions of the input area.

WRITE Macro

Name	Operation	Operand
[name]	WRITE	$\left\{\begin{matrix}\text{filename}\\ \text{(1)}\end{matrix}\right\}, \left\{\begin{matrix}\text{KEY}\\ \text{ID}\\ \text{AFTER [,EOF]}\\ \text{RZERO}\end{matrix}\right\}$

The KEY, ID, or AFTER forms of the macro transfer an output record from virtual storage to DASD storage. The output area must be specified in the DTFDA IOAREA1 operand, 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. This form writes an end-of-file record (a record with a length of zero) on a specified track after the last record on a track.

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), you must determine the length of each record and load it into a register for IOCS use before you issue the WRITE macro for that record. The register for this purpose must be specified in the DTFDA RECSIZE operand.

If you are creating variable length or spanned unblocked records with WRITE filename,AFTER you must put the data length of the record to be written plus 4 into the 5th and 6th bytes of the control words preceding the data. In the case that you are updating records previously read by a READ macro from the same physical file you should not change the control words. Otherwise, the wrong length record bit will be set in the error information returned to your program.

Record Reference by Key
If the DASD location for writing records is determined by the record key (control information in the key area of the DASD record), the word KEY must be entered as the second parameter of the WRITE macro. Also the WRITEKY operand must be included in the DTFDA.

Whenever this method of reference is used, your program must supply the key of the desired record to IOCS before the WRITE is issued. The key must be stored in the key field (specified by the DTFDA KEYARG operand). When the WRITE 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 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 you request that multiple tracks be searched on each WRITE macro. Searching multiple tracks is specified by including the SRCHM operand in the DTFDA. 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 file.

Record Reference by ID
If the DASD location for writing records is determined by the record ID (identifier in the count area of records), ID must be entered as the second parameter of the WRITE macro and the WRITEID operand must be included in the DTFDA.

Whenever this method of reference is used, your program must supply both the track information and the record number in the track-reference field. When the WRITE 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 output area is transferred to the key area (if present and specified in DTFDA KEYLEN) and the data area of the DASD rerecord. If RECFORM is FIXUNB or UNDEF the key must precede your data in the IOAREA1, otherwise you must load the key into the key field (specified by the KEYARG operand) before you issue the WRITE macro. 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 set set on.

Record Reference by 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 macro must be AFTER and the AFTER operand must be included in the DTFDA.

Whenever this method of reference is used for writing records, your 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 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, or the track is not followed by enough empty tracks in the case of spanned records, IOCS does not write the record and, instead, sets an indication in your error/status byte specified by the DTFDA ERRBYTE operand.

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.

For the 2311 and 2314, the capacity record will take into account a track tolerance of about 5%, to ensure that minor hardware imprecisions on the disk tracks do not interfere with program execution. If a record is close to the maximum record size for a track, the capacity record could thus show a negative value.

Record Reference by RZERO
WRITE filename,RZERO resets the capacity record to reflect an empty track. Your 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 in a two byte field and erases the full track after R0. The maximum track capacities are:

for 2311	3625
for 2314 or 2319	7294
for 3330 or 3333	13030
for 3340	8368
for 2321	2000

This form of the WRITE macro should be issued every time your program reuses a certain portion of a pack or data module. 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 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 macro must be issued before your program attempts to process an input record which has been read or to build another output record for the file concerned. The program does not regain control until the data transfer is complete. Thus, the WAITF macro must be issued after any READ or WRITE macro for a file, and before the succeeding READ or WRITE macro for the same file. The WAITF macro makes error/status information, if any, available to your program in the field specified by the DTFDA ERRBYTE operand.

CNTRL Macro

Name	Operation	Operand
[name]	CNTRL	{filename / (1)}, code

The CNTRL (control) macro can begin DASD access movement (SEEK) or restore a data cell strip (RESTR) for the next READ or WRITE. 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 specified either as a symbol or in register notation.

The second parameter must be the word SEEK (for any DASD) or RESTR (for the 2321 only). The seek address must be provided in the field with the name given in the DTFDA SEEKADR operand before issuing the CNTRL macro.

Completion Macros

CLOSE and CLOSER Macros

Op	Operand
for self-relocating programs	
CLOSER	$\left\{\begin{matrix}\text{filename1}\\ (\text{r1})\end{matrix}\right\}$ $\left[,\left\{\begin{matrix}\text{filename2}\\ (\text{r2})\end{matrix}\right\}\ldots,\left\{\begin{matrix}\text{filenamen}\\ (\text{rn})\end{matrix}\right\}\right]$
for programs that are not self-relocating	
CLOSE	$\left\{\begin{matrix}\text{filename1}\\ (\text{r1})\end{matrix}\right\}$ $\left[\left\{\begin{matrix}\text{filename2}\\ (\text{r2})\end{matrix}\right\}\ldots,\left\{\begin{matrix}\text{filenamen}\\ (\text{rn})\end{matrix}\right\}\right]$

The CLOSER or CLOSE completion macro must be used after the processing of a file is completed. These macros end the association of the logical file declared in your program with a specific physical file on a DASD.

The CLOSER or CLOSE macro deactivates any file which was previously opened. 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. No further commands can be issued for the file unless it is reopened.

When CLOSER is specified, the symbolic address constants that CLOSER generates from the parameter list are self-relocating. When CLOSE is specified, the symbolic address constants are not self-relocating.

To write the most efficient code in a multiprogramming environment it is recommended that CLOSER be used.

Enter the symbolic name of the file (assigned in the DTF header entry) in the operand field. A maximum of 16 files may be closed by one macro by entering additional filename parameters as operands. Alternately, you 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. For CLOSER, the address of filename may be preloaded into any of the registers 2-15. For CLOSE, the address of filename may be preloaded into register 0 or any of the registers 2-15.

Note: If you use register notation, we recommend that you follow the standard practice of using only registers 2-12.

PART 4

INDEXED SEQUENTIAL ACCESS METHOD

Concepts of ISAM

Declarative Macros

DTFIS
ISMOD

Imperative Macros

CLOSE	OPENR
CLOSER	PUT
ENDFL	READ
ERET	SETFL
ESETL	SETL
GET	WAITF
OPEN	WRITE

CONCEPTS OF ISAM

With ISAM you can process DASD records in either random or sequential order. For random processing, you supply the key (control information) of the desired record to ISAM and issue a READ or WRITE macro to transfer the specified record. For sequential processing, you specify the first record to be processed and then issue GET or PUT macros until all desired sequential records are processed. The successive records are made available in sequential order by key. Variations in macros permit:

- Creating a DASD file
- Reading, adding to, or updating a DASD file

Whenever ISAM is used, the file must be defined by the declarative macro DTFIS (Define The File for Indexed Sequential system). The detail entries for this macro are described in the *Declarative Macros* section later in this chapter. In order to understand the use of some of these entries, however, it is first necessary to indicate how ISAM processing uses them.

For processing VSAM files with an ISAM program by means of the ISAM Interface Program (IIP), see Appendix J.

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:

Count
Key
Data

All keys must be the same length, and this length must be specified in the DTFIS KEYLEN operand.

The logical records must be fixed length, and the length must be specified in the DTFIS RECSIZE operand. Logical records may be either blocked or unblocked, and this is specified in the DTFIS RECFORM operand. When blocked records are specified, the key of the highest (last) record 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 NRECDS operand.

Storage Areas

Records of one logical file are transferred to, or from, one or more I/O areas in virtual 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*, below). When an overflow record is brought into the I/O area, you should not alter the sequence-link field. The I/O area requirements are illustrated in Figure 4-1 and described in detail in the discussions of the DTFIS IOAREAL, IOAREAR, IOAREAS, and IOAREA2 operands.

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 IOREG operand. This register is used for indexing, and points to the beginning of each record.

If the records are processed in a work area, the DTFIS WORKL, WORKR, or WORKS operand must be specified (WORKL must be specified in any event when creating or adding records to a file). ISAM moves each individual input record from the I/O area to the work area where it is available to your program for processing. Similarly, on output ISAM moves the completed record from the work area to the I/O area where it is available for transfer to DASD storage. Whenever a work area is used, no register is required.

Organization of Records on DASD

When a logical file of presorted records is loaded onto a DASD, ISAM organizes the file in a way that allows you to access any record. For any type

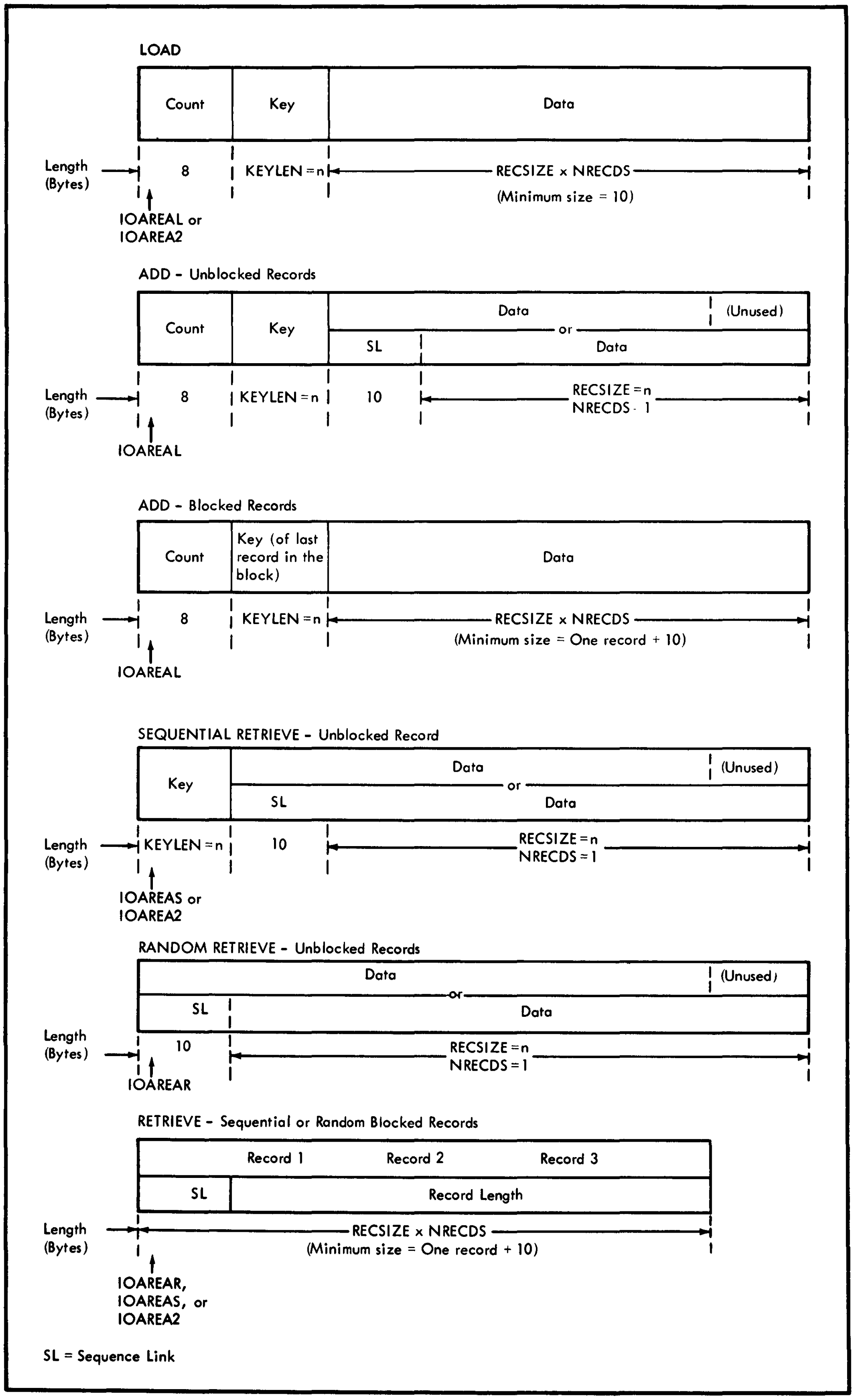


Figure 4-1 **I/O areas resulting from different DTFIS operands**

of processing, the entire ISAM file must be on line. If an ISAM file is assigned ignore by JCL, no processing can be done for that file.

Reference can be made to records at random throughout the file, or to a series of records in the file in their 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 into a specified area of the DASD volume. This area is called the prime area. Both the starting and ending limits of this area are specified by EXTENT job control 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 it. The indexes:

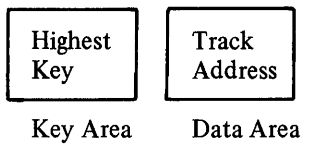
- Permit rapid access to individual records for random processing.
- Supply the records in key order for sequential processing.

Either two or three indexes are built: a track index and a cylinder index are always built, and a master index is also built if you specify the DTFIS MSTIND operand.

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, ISAM searches the master index (if used), and then the cylinder index, and 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 (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 KEYLEN operand. 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. Therefore you should not use a key of all one bits for any of your records.

Examples of a track index, cylinder index, and master index are shown in the *DOS/VS Data Management Guide*, GC33-5372.

Track Index

The track index is the lowest-level index for the logical file. A separate track index is built for each cylinder used, 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.

When the track indexes are originally constructed, they contain two similar entries (normal and overflow) for each track used on the cylinder. 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*, below). 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 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 which you specify in an EXTENT job control statement. You may change the upper extent limit; however, no validity check is performed by the ISMOD and it is therefore your 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. 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 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. 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 the data cell corresponds to the cylinder of a disk drive in ISAM.

Master Index

The optional master index is the highest-level index for a logical file. This index is built only if it is specified by the DTFIS MSTIND operand. A master index is built in the location specified by an EXTENT job control statement. Like the cylinder index, it may be located on the same volume with the data 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. The dummy entry indicates the end of the master index.

Addition of Records and Overflow Areas

After a logical file is organized on a DASD, it may subsequently become necessary to add records. 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 file.

If all records to be added have keys that are higher than the highest key in the file, the upper limit of the prime area of the file can be adjusted (if necessary) with an EXTENT job control 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 in the file, 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 your 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 the DTFIS RECFORM operand for the data records in the file.

An example of the addition of records to an ISAM file using an overflow area is shown in the *DOS/VS Data Management Guide*, GC33-5372.

You may request two types of overflow areas:

- A cylinder overflow area for each cylinder. Specify the number of tracks to be reserved for each cylinder overflow area with the DTFIS CYLOFL operand when a file is loaded or when records are added to an existing file.

- An independent overflow area for the entire file, specified with an EXTENT job control statement. This area may be on the same volume as the file or on a different (on-line) volume of the same device type. An independent overflow area may be added to a file originally created without it when the DTFIS IOROUT operand specifies LOAD, ADD, or ADDRTR.

The independent overflow area may be used in addition to cylinder overflow areas or without them. When used in addition to cylinder overflow areas, it is used whenever one of the cylinder overflow areas is filled.

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 chapter.

Programming Considerations

Recordsize=keylength+(blocking factor x record length). The maximum record size possible for ISAM on the various direct access devices is shown below.

Device	Maximum Record Size (Bytes)
2311	3,605
2314	7,249
2319	7,249
2321	1,984
3330	12,974
3333	12,974
3340	8,293

Formulas to calculate the storage requirements for an ISAM file on the various direct-access devices are given in the *DOS/VS Data Management Guide*, GC33-5372.

When writing ISAM programs, do not forget to include the LBLTYP, DLBL (or DLAB) job control statements. Information about these job control statements will be found in *DOS/VS System Control Statements*, GC33-5376. Examples of complete sets of job control statements for ISAM will also be found there.

DOS/VS does not support a null ISAM file. If an attempt is made to access a null file, an X'10' error indication is placed in the field filenameC (FilenameC is described in the discussion of the DTFIS ERREXT operand, below).

ISAM maintains a helpful set of statistics to assist you in determining when reorganization of an ISAM file is required. These statistics are maintained in the Format 2 DASD label recorded with the file; when the file is processed, the statistics occupy fields within the DTFIS table. You can test these fields as you process the file. The fields, and the names by which you reference them, are described below.

- prime record count (filenameP).
  A four-byte count of the number of records in the prime data area. FilenameP is used for DTFIS ADD, while filenameP+4 is used for DTFIS LOAD.

- overflow record count (filenameO).
  A two-byte count of the number of records in the overflow area(s).

- available independent overflow tracks (filenameI).
  A two-byte count of the number of tracks remaining in the independent overflow area, if used.

- cylinder overflow areas full (filenameA).
  A two-byte count of the number of cylinder areas that are full, necessitating use of the independent overflow area.

- non-first overflow reference (filenameR).
  A four-byte count of the number of times a random reference (READ) is made to records that are the second or higher links in an overflow chain.

In addition to these fields maintained automatically by ISAM, there is another field--filenameT--which you can use to keep a count of records tagged for deletion. This field is kept in the Format 2 DASD label recorded with the file and is available in the DTFIS table when the file is processed. You may tag the records for deletion by any method you desire, so long as the keys of the records are not changed in such a way that the sequence in the file would be altered. For instance, you could overwrite the data portion of a record with zeros; or a special field within a record could indicate that the record is deleted. You can keep a count of such records in filenameT. When reorganizing the file, tagged or deleted records can be eliminated. Additional information on reorganizing an ISAM file appears in the *DOS/VS Data Management Guide*, GC33-5372.

Example of an ISAM File

Figure 4-2 shows schematically a simplified example of a file organized on a DASD by ISAM. This figure illustrates a file on a 3330, 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 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 1-16. Tracks 17 and 18 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	.
.	.
16	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 16.
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 overflow area.

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).

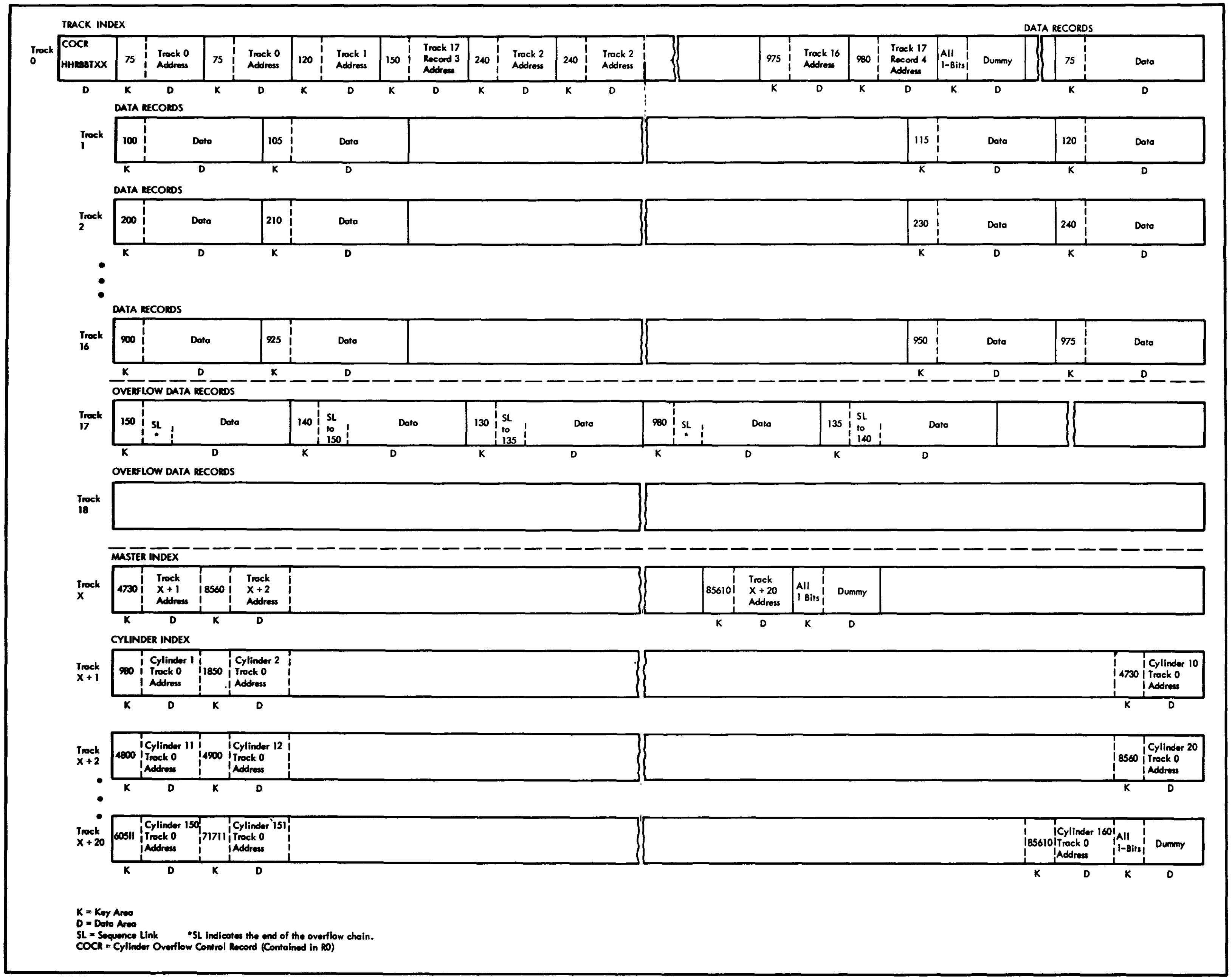


Figure 4-2 Example of a File Organized by ISAM

ISAM files must first be defined by the declarative macros before the imperative macros, described later in this chapter, are used to operate on the files.

There are two related types of declarative macros--DTFIS and ISMOD. These two macros are described below.

DTFIS Macro

Enter the symbolic name of the file (filename) in teh name field and DTFIS in the operation field. The detail entries follow the DTFIS header card in any order. Figure 4-3 lists the keyword operands contained in the operation field.

Applies to						
Ran. Rtvl.	Seq. Rtvl.	Load	Add			
X	X	X	X	M	DSKXTNT=n	Maximum number of extents specified for this file
X	X	X	X	M	IOROUT=xxxxxx	(LOAD, ADD, RETRVE, or ADDRTR
X	X	X	X	M	KEYLEN=nnn	Number of bytes in record key (maximum is 255)
X	X	X	X	M	NRECDS=nnn	Number of records in a block. Required for blocked records only; if unblocked, 1 is assumed.
X	X	X	X	M	RECFORM=xxxxxx	(FIXUNB or FIXBLK)
X	X	X	X	M	RECSIZE=nnnn	Number of characters in logical record
X	X	X	X	O	CYLOFL=nn	Number of tracks for each cylinder overflow area. Maximum = 8 for 2311, 18 for 2314 and 2321, 17 for 3330 and 3333, 10 for 3340
X	X	X	X	O	DEVICE=nnnn	(2311, 2314, 2321, 3330, 3340). If omitted, 2311 is assumed
X	X	X	X	O	ERREXT=YES	Non data-transfer error returns and ERET desired
X	X	X	X	O	HINDEX=nnnn	(2311, 2314, 2321, 3330, 3340). Unit containing highest level index. If omitted, 2311 is assumed
X	X		X	O	HOLD=YES	Track hold function is desired
X			X	O	INDAREA=xxxxxxxx	Symbolic name of cylinder index area
X			X	O	INDSKIP=YES	Index skip feature is to be used
X			X	O	INDSIZE=nnnnn	Number of bytes required for the cylinder index area

M=Mandatory; O=Optional

Figure 4-3 DTFIS macro (part 1 of 2)

Applies to

Ran. Rtvl.	Seq. Rtvl.	Load	Add			
		X	X	O	IOAREAL=xxxxxxxx	Name of I/O area
X				O	IOAREAR=xxxxxxxx	
	X			O	IOAREAS=xxxxxxxx	
	X	X		O	IOAREA2=xxxxxxxx	Name of second I/O area
X	X			O	IOREG=(nn)	Register number. Omit if WORKA or WORKS is specified
			X	O	IOSIZE=nnnn	Bytes alloted to IOAREAL
X	X			O	KEYARG=xxxxxxxx	Number of key field in storage, for random retrieval or sequential retrieval starting by key
X	X	X	X	O	KEYLOC=nnnn	Number of high-order position of key field within record, if RECFORM=FIXBLK
X	X	X	X	O	MODNAME=xxxxxxx	Name of ISMOD logic module for this DTF. If omitted, IOCS generates standard name
X	X	X	X	O	MSTIND=YES	Master index used
X	X	X	X	O	RDONLY=YES	Generates a read-only module. Requires a module save area for each task using the module
X	X	X	X	O	SEPASMB=YES	DTFIS is to be assembled separately.
X	X			O	TYPEFLE=xxxxxx	(RANDOM, SQNTL, or RANSEQ)
X	X	X	X	O	VERIFY=YES	Check disk records after they are written. For DEVICE=2321, YES is assumed.
		X	X	O	WORKL=xxxxxxxx	Name of work area for loading or adding to the file
X				O	WORKR=xxxxxxxx	Name of work area for random retrieval. Omit IOREG
	X			O	WORKS=YES	GET or PUT specifies work area

M=Mandatory; O=Optional

Figure 4-3 **DTFIS macro (part 2 of 2)**

CYLOFL=n

This operand must be included if cylinder overflow areas are reserved for a 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 of tracks to be reserved on each cylinder. The maximum number of tracks that can be reserved on each cylinder is:

for 2311	8
for 2314, 2319, or 2321	18
for 3330 or 3333	17
for 3340	10

If an independent overflow area is also specified (by an EXTENT job control statement), overflow records are written in the independent overflow area after a cylinder overflow area becomes filled.

DEVICE={2311 | 2314 | 2321 | 3330 | 3340}

This operand specifies the unit that contains the prime data area or overflow areas for the logical file. Specify 2314 for 2319 and 3330 for 3333. For ISAM the prime data area must be on the same device type, and for a 3340 on the same model of data module.

DSKXTNT=n

This operand must be included to specify the maximum number 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 job control 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 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 is required for IOCS to supply your program with detailed information about irrecoverable I/O errors occurring before a data transfer takes place, and for your program to be able to use the ERET imperative macro to return to IOCS specifying an action to be taken for an error condition.

Some error information is available for testing by your program after each imperative macro is executed regardless of whether ERREXT=YES is specified or not, by referencing the field *filenameC.* Filename is the same name as that specified in the DTF header entry for the file. One or more of the bits in the filenameC byte may be set to 1 by IOCS. The meaning of the bits varies depending on which parameter was specified in the IOROUT operand; Figure 4-4 shows the meaning if IOROUT=ADD, RETRVE, or ADDRTR was specified; Figure 4-5 shows the meaning if IOROUT=LOAD was specified.

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 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 primefile 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. You 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 4-4 **FilenameC - status or condition code byte if IOROUT=ADD, RETRV, or 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	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 file. You 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 entries needed to index each cyliner specified for the prime data area. This condition can occur during the execution of the SETFL. You 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. You must extend the upper limit, if you are creating the file, by using an extent card. Or, you must reorganize the 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 wite an EOF record. This condition can occur during the execution of the ENDFL macro.

Figure 4-5 **FilenameC - status or condition code byte if IOROUT=LOAD**

If ERREXT=YES is not specified, IOCS returns the address of the DTF table in register 1, as well as any data-transfer error information in filenameC, after each imperative macro is executed; non-data-transfer error information is not given. After testing filenameC return to IOCS by issuing any imperative macro except ERET; no special action is taken by IOCS to correct or check an error.

If ERREXT=YES is specified, IOCS returns the address of an ERREXT parameter list in register 1 after each imperative macro is executed, and information about both data-transfer and non-data-transfer errors in filenameC. The format of the ERREXT parameter list is shown in Figure 4-6. After testing filenameC and finding an error, return to IOCS by using the ERET imperative macro; IOCS takes the action indicated by the ERET operand. If HOLD=YES (and ERREXT=YES), ERET must be used to return to IOCS to free any held track.

Your program is also 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 your program is not equal to the blocksize of the previously built file, this bit will be set to 1.

HINDEX={2311 | 2314 | 2321 | 3330 | 3340}
This entry specifies the unit containing the highest index. Specify 2314 for 2319 and 3330 for 3333.

Bytes	Bits	Contents
0-3	–	DTF address
4-7	–	Virtual 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 above.
16	.	Record identification:
	1	Data record
	2	Track index record
	3	Cylinder index record
		Master index record
	.	Type of operation:
	4	Not used
	5	Not used
	6	Read
	7	Write
17	–	Command code of failing CCW

Figure 4.6 **ERREXT parameter list**

HOLD=YES
This operand provides for the track hold option for both data and index records. If the HOLD operand is omitted, the track hold function is not performed.

Because track hold cannot be performed on a LOAD file, HOLD=YES cannot be specified when IOROUT=LOAD.

If HOLD=YES and ERREXT=YES, your program must issue the ERET macro to return to the ISAM module to free any held tracks.

For further information see *DASD Track Protection Macros* in the *Multitasking Macros* chapter.

INDAREA=name
This operand specifies the name of the area assigned to the cylinder index. If specified, all or part of the cylinder index resides in virtual 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 virtual 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 virtual storage, this operand specifies the index skip feature. This feature allows ISAM to skip any index entries preceding those needed to process a given key. If the index skip operand is omitted, the cylinder indexes are processed sequentially.

This operand may only be specified with the INDAREA and INDSIZE operands and increases throughput only when:

- The records are presorted.
- The allocated virtual storage is insufficient for storing all of the cylinder index.
- A large segment(s) of the file is not referenced.

INDSIZE=n
This operand specifies the length (in bytes) of the index area assigned in virtual storage to the cylinder index by INDAREA. The minimum number must be:

(m+3)(keylength+6)

where m is the number of entries to be read into virtual 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, the entire cylinder index is read into virtual storage at one time.

The resident index facility is suppressed if this operand is omitted or if the minimum requirement is not met at assembly time, or if an irrecoverable 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 a file. It specifies the 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 storage. The ISAM routines construct the contents of this area and transfer records to DASD.

This output area must be large enough to contain the count, key, and data areas 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 (see Figure 4-7).

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, the IOREAL area must be at least as large as the number of bytes specified in the IOSIZE operand.

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 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 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 (see Figure 4-8).

IOAREAS=name
This operand must be included whenever records are processed in sequential order by key. It specifies the name of the input/output area used for sequential

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	—	Record Length x Blocking Factor
			OR*	
	8	Key Length	10	Record Length
* Whichever Is Larger				

Figure 4-7 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
			OR*	
	—	—	10	Record Length
* Whichever is Larger				

Figure 4-8 I/O area requirements for random or sequential retrieval by ISAM

retrieval (and updating). The specified name must be the same as that used in the DS instruction that reserves this area of storage.

This 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 4-8).

IOAREA2=name
This operand permits overlapping of I/O with indexed sequential processing for either the load (creation) or sequential retrieval functions. Specify the name of an I/O area to be 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 operand for the load function or the IOAREAS operand for the sequential retrieval function. If the operand is omitted, one I/O area is assumed. If TYPEFLE=RANSEQ, this operand must not be specified.

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. The parameters have the following meanings:

LOAD	To build a logical file on a DASD or to extent a file beyond the highest record presently in a file.
ADD	To insert new records into a 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 (decimal) number of bytes in the virtual-storage area assigned for the add function using IOAREAL. The number can be computed using the following formula:

m(keylength+blocksize+40)+24

where m is the maximum number of physical records that can be read into virtual storage at one time; 40 is the sum of 8 for the count field and 32 for an ISAM CCW; 24 is another ISAM CCW; and n must also be at least equal to

(keylength+blocksize+74)

This formula accounts for a needed sequence link field for unblocked records or short blocks (see Figure 4-4).

If omitted, or if the minimum requirement is not met, no increase in throughput is realized.

n should not exceed the track capacity because the throughput cannot be increased by enlarging it further.

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 key field in which you must supply the record key to ISAM.

KEYLEN=n
This operand must be included to specify the number of bytes in the record key.

KEYLOC=n
This operand must always be specified if RECFORM=FIXBLK. 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 operand should specify 21.

ISAM uses this specification to locate (by key) a specified record within a block. The key area of a block of records 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 must specify the same name as the ISMOD macro. If this entry is omitted, standard names are generated for calling the logic module. If two DTF macros 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 or is to be built for a file. The location of the master index is specified by an EXTENT job control statement.

NRECDS=n
This operand specifies the number of logical records in a block (called the blocking factor). It is required only if RECFORM=FIXBLK.

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 task should have its own uniquely defined save area. Register 13 must contain the address of the save area associated with the task each time an imperative macro (except OPEN, OPENR, LBRET, SETL, or SETFL) is issued. 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* in the chapter *Multitasking Macros.*

RECFORM={FIXUNB | FIXBLK}
This operand specifies whether records are blocked or unblocked. FIXUNB is used for unblocked records, and 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 onto a DASD 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,* above), but this has no effect on this operand. RECFORM refers to records in the prime data area only.

RECSIZE=n
This operand must be included to specify the number of characters in the data area of each individual record. This operand should specify the same num-

ber for additions and retrieval as indicated when the file was created.

SEPASMB=YES
Include this operand only if the DTFIS 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 problem program and no CATALR card is punched.

TYPEFLE={RANDOM | SEQNTL | RANSEQ}
This operand must be included when IOROUT=RETRVE or ADDRTR. It specifies the type(s) of processing performed by your program for the file.

RANDOM	is used for random processing. Records are retrieved in random order specified by key.
SEQNTL	is used for sequential processing. Your 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* later in this chapter.
RANSEQ	is used if both random and sequential processing are to be performed for the same file. If RANSEQ is specified, the IOAREA2 operand must not be specified.

TYPEFLE is not required for loading or adding functions.

VERIFY=YES
Use this operand if you want 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 a disk are not verified.

WORKL=name
This operand must be included whenever a file is created (loaded) or records are added to a file. It specifies the name of the work area in which you 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 storage. This work area must provide space for one logical 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 operand must be included if the individual records are to be processed in a work area rather than in the I/O area. It specifies the name of the work area. This name must be the same as the name used in the DS instruction that reserves this area of storage. This area must provide space for one logical record (data area). When this entry is included and a READ or WRITE macro is executed, ISAM moves the individual record to, or from, this area.

WORKS=YES
When records are processed in sequential order, this operand must be included if the individual records are processed in work areas rather than in the I/O area. Each GET and PUT macro must specify the name of the work area 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 work area 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		

ISMOD Macro

Listed here are the operands you can supply for ISMOD. The first card contains ISMOD in the operation field and may contain a module name in the name field. The operands are explained below and shown in Figure 4-9.

Note: If an ISMOD module precedes an assembler-language USING statement or follows your program, registers 2-12 remain unrestricted even at assembly time. However, if the ISMOD module lies within your program, you should issue the same USING statement (as that which was issued before the IS-MOD module) directly following the module. This action is necessary because the ISMOD module uses registers 1, 2, and 3 as base registers, and the IS-MOD CORDATA 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 module that can process DTFIS files (add or random retrieve functions) with or without the cylinder index entries resident in virtual storage. If omitted, the module generated cannot process the resident cylinder index entries.

If an irrecoverable I/O error occurs while reading indexes into virtual storage, the program will not use 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 IOSIZE operand is required in the DTF. 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 non-data-transfer error conditions are returned in filenameC.

If HOLD=YES and ERREXT=YES, your program must issue the ERET macro to return to the ISAM module to free any held tracks. See the DTF ER-REXT and HOLD operands.

HOLD=YES
This operand provides for the track hold option for both data and index records. If the HOLD operand is omitted, the track hold function is not performed.

Because track hold cannot be performed on a LOAD file, HOLD=YES cannot be specified when IOROUT=LOAD.

If HOLD=YES and ERREXT=YES, your program must issue the ERET macro to return to the ISAM module to free any held tracks.

For further information see *DASD Track Protection Macros* later in this chapter.

IOAREA2=YES
Include this operand if a second I/O area is to be used--that is, if IOAREA2 is specified in the DTF. The operand is only valid for load or sequential retrieval functions. This module can process DTFs with one or two I/O areas specified. This operand must not be specified if TYPEFLE=RANSEQ is specified.

IOROUT={LOAD | ADD | RETRVE | ADDRTR}
This operand specifies the type of module required to perform a given function.

LOAD generates a module for creating or extending a file.

ADD generates a module for adding new records to an existing file.

RETRVE generates a module to retrieve (randomly/sequentially) records from a file.

ADDRTR generates a 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 of the DTF, and in which the TYPEFLE operand contains the corresponding parameter (or a subset of it).

RDONLY=YES
This operand causes a read-only module to be generated. Whenever this operand is specified, any DTF used with the module must have the same operand.

RECFORM={FIXUNB | FIXBLK | BOTH}
This operand generates a module that creates, adds to, or processes an unblocked (FIXUNB) or blocked (FIXBLK) file. If BOTH is specified, a module is generated to process both unblocked and blocked files, and the DTF may specify either FIXUNB or FIXBLK in the RECFORM operand. The RECFORM operand is required only when IOR-OUT 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
Include this operand only if the module is assembled separately. This causes a CATALR card with the module name (standard or user-specified) to be punched ahead of the object deck and defines the module name as an ENTRY point in the assembly. If the operand is omitted, the program assumes that the

DTF is being assembled with the problem program and no CATALR card is punched.

TYPEFLE={RANDOM | SEQNTL | RANSEQ}
This operand is required when IOROUT specifies RETRVE or ADDRTR. RANDOM generates a module that includes only random retrieval capabilities. SEQNTL generates a module that includes only sequential retrieval capabilities. RANSEQ generates a module that includes random and sequential capabilities. It also processes any file in which the TYPEFLE operand specifies either RANDOM or SEQNTL. If TYPEFLE=RANSEQ, IOAREA2=YES must not be specified.

When all operands are omitted, the ISMOD module can only process files where IOROUT=RETRVE, TYPEFLE=RANSEQ, CORINDX, CORDATA, HOLD, and RDONLY are not specified. In this event, the module name assumed is IJHZRBZZ.

Standard ISMOD Names
Each name begins with a 3-character prefix (IJH) and continues with 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

Name	Operation	Operand	Remarks
[modname]	ISMOD		Must be included.
		ERREXT=YES	Required if non-data-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 virtual 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 RETRV or ADDRTR.

Figure 4-9 **ISMOD macro**

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 *IOCS Subset/Superset Names* in *The Macro System* chapter.

```
      + + + + +
I J H A A B B F
      B I R O O
      Z + + + +
      + A B C S
      A R S Z Y
      U * +   +
      Z L G   G
          S   P
          +   +
          G   T
          Z   Z
```

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

IMPERATIVE MACROS

After the ISAM files are defined by the declarative macros, the imperative macros can be used to operate on the files. The imperative macros are divided into three groups: those for initialization, processing, and completion.

Initialization Macros

OPEN and OPENR Macros

Op	Operand
for self-relocating programs	
OPENR	$\begin{Bmatrix}\text{filename1}\\ \text{(r1)}\end{Bmatrix}$ $\left[,\begin{Bmatrix}\text{filename2}\\ \text{(r2)}\end{Bmatrix}\ldots,\begin{Bmatrix}\text{filenamen}\\ \text{(rn)}\end{Bmatrix}\right]$
for programs that are not self-relocating	
OPEN	$\begin{Bmatrix}\text{filename1}\\ \text{(r1)}\end{Bmatrix}$ $\left[,\begin{Bmatrix}\text{filename2}\\ \text{(r2)}\end{Bmatrix}\ldots,\begin{Bmatrix}\text{filenamen}\\ \text{(rn)}\end{Bmatrix}\right]$

The OPENR or OPEN macro must be used to activate an ISAM file for processing. These macros associate the logical file declared in your program with a specific physical file on a DASD. The association by OPENR or OPEN of your program's logical file with a specific physical file remains in effect throughout your processing of the file until you issue a CLOSE or CLOSER macro.

When OPENR is specified, the symbolic address constants that OPENR generates from the parameter list are self-relocating. When OPEN is specified, the symbolic address constants are not self-relocating.

To write the most efficient code in a multiprogramming environment it is recommended that OPENR be used.

Self-relocating programs using LIOCS must use OPENR to activate all files, including console files.

In addition to activating files for processing, OPENR relocates all address constants (except zero constants) within the DTF tables.

If OPEN or OPENR attempts to activate a LIOCS file (DTF) whose device is unassigned, the job is terminated. If the device is assigned IGN, the OPEN or OPENR does not activate the file but turns on DTF byte 16, bit 2, to indicate the file is not activated. If DTF byte 16 bit 2 is on after issuing an OPEN or OPENR, input/output operations should not be performed for the file.

Enter the symbolic name of the file (DTF filename) 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, you 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. If symbolic notation is used, you must establish addressability through a base register. For OPENR, the address of filename may be preloaded into any of the registers 2-15. For OPEN, the address of filename may be preloaded into register 0 or any of the registers 2-15.

Note: If you use register notation, we recommend that you follow the standard practice of using only registers 2-12.

Whenever a DASD file is opened, you must provide the information for checking or building the labels. (See the *Label Processing* chapter.)

When a file is created or extended, 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 or OPENR checks the standard VOL1 label and performs extensive checks on the extents specified in the EXTENT job control statements for that volume. The extents must meet the following conditions:

1. All prime data extents must be contiguous.
2. The master and cylinder index extents must be contiguous 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 job control statements for the independent overflow tracks can be placed either before or after all the EXTENT job control statements for the prime data extents.

OPEN or OPENR 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 or OPENR 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 job control statements are available for writing. All volumes containing an ISAM file must be on-line and ready when the file is first opened.

For each volume, OPEN or OPENR checks the extents specified in the EXTENT job control statements for that volume (for example, checks that the data extents are contiguous). OPEN or OPENR 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 job control statements are available for use.

Processing Macros

Once ISAM files have been readied for processing with the OPENR or OPEN macro, the processing macros described in this section may be used.

In this section, first the frequently used ERET macro is described, and then the groups of macros used for:

- Loading or extending a file
- Adding records to a file
- Random retrieval of records
- Sequential retrieval of records

ERET Macro

Name	Operation	Operand
[name]	ERET	$\left\{\begin{array}{l}\underline{\text{SKIP}}\\ \text{IGNORE}\\ \text{RETRY}\end{array}\right\}$

At the completion of each imperative macro, filenameC should be checked by your error routine. See the DTFIS ERREXT operand for details and for the format of filenameC. The ERET (error return) macro enables a program specifying the ERREXT operand in the DTF to return to IOCS and specify an action to be taken for each error condition.

After each imperative macro is executed, register 1 contains the address of an 18-byte parameter list. The contents of this parameter list are shown in Figure 4-6. Nondata transfer error conditions are indicated in the DTF data transfer bit (byte 2, bit 2) and your error routine can return to IOCS via the ERET macro. The ERET IGNORE or ERET SKIP operand returns to IOCS to ignore the error condition and to continue processing with the block in error. The ERET RETRY operand returns to IOCS to make another attempt at reading or writing the record which caused the error.

Note: The ERREXT routine does not handle nonrecoverable errors that are posted in filenameC. Examples of nonrecoverable 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 nonrecoverable no record found condition.

Your error 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, the data was not read or written. If it is off, data transfer did take place.

If any IOCS macros other than ERET are issued in the error routine, the contents of registers 14 and 13 (with RDONLY) should be saved before use and restored after use.

If HOLD=YES is specified, you must issue the ERET macro to return to IOCS during an ERREXT to free any held tracks.

Note: If the error occurred on an index record, you should not IGNORE this record unless it is first checked for accuracy. If the record was read inaccurately, you should RETRY to read the record.

Loading or Extending a File

The functions of originally loading a file of presorted records onto a DASD, and of 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 IOROUT=LOAD operand), and use the same macros. However, the type field in the DLAB job control statement must specify ISC for load creation and ISE for load extension.

The areas of the volumes used for the file are specified by EXTENT job control statements. The areas are:

The prime area where the data records are written.

A cylinder index area where your want ISAM to build the cylinder index.

A master index area if a master index is to be built (specified by the DTFIS MSTIND operand).

During a load operation, ISAM builds the track, cylinder, and master indexes.

A combination of three different macros is required in your program to load original or extension records onto a DASD. These macros are SETFL, WRITE, and ENDFL.

SETFL sets the ISAM processing mode for loading or extending a file. WRITE performs the actual loading of new records into the file. ENDFL turns the load mode off. These three macros are described in detail below.

SETFL Macro

Name	Operation	Operand
[name]	SETFL	{filename} {(0)}

The SETFL (set file load mode) macro causes ISAM to set up the file so that the load or extension function can be performed. This macro must be issued whenever the file is loaded or extended. 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 for this macro 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.

WRITE Macro

Name	Operation	Operand
[name]	WRITE	{filename} {(1)},NEWKEY

When a WRITE macro with the parameter NEWKEY is issued in your program between a SETFL macro and an ENDFL macro, ISAM loads a record onto the DASD.

The WRITE macro for loading and extending 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 NEWKEY.

Before issuing the WRITE macro, your program must store the key and data portions of the record in a work area (specified by DTFIS WORKL). The ISAM routines construct the I/O area (see Figure 4-1) 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 record 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 your 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. Using filenameH is required if you plan to retrieve records in sequential order starting with the ID of a particular record (see *SETL Macro*, below).

As records are loaded or extended 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

Name	Operation	Operand
[name]	ENDFL	{filename / (0)}

The ENDFL (end file load mode) macro ends the mode initiated by the SETFL macro. The name of the file to be loaded is the only parameter required, 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 or CLOSER 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.

Adding Records to a File

New records can be added to an existing ISAM file. Each record is inserted in the proper place sequentially by key. To provide this function specify ADD or ADDRTR in the DTFIS IOROUT operand.

The file may contain either blocked or unblocked records, as specified by the DTFIS RECFORM operand. When the file contains blocked records, you must provide ISAM with the location of the key field provided through the DTFIS KEYLOC operand. The records to be inserted 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 ESETL must be issued before a record is added. Two macros--WRITE and WAITF--are used in a program to actually add records to a file.

WRITE Macro

Name	Operation	Operand
[name]	WRITE	{filename / (1)} ,NEWKEY

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 work area specified in the DTFIS WORKL operand. 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 the I/O area specified in the DTFIS IOAREAL operand. The two records are then compared to check for duplicate records. If a duplication is found, this information is posted in the DTF table at filenameC. If none is found, the appropriate record (in your 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 your work area, and the next record on the track is read into the I/O area. Then, the record in the work area 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 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 fit are read from the track and placed in the I/O area (specified in the DTFIS operand IOAREAL). The added record is compared with existing records in the I/O area. If a duplicate key is found, the condition is posted for you in the DTF table at filenameC. If no duplicate is found, the records are shifted in virtual storage, leaving the record with the highest key remaining in the work area. 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. It 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 and the area is not filled.

If the cylinder overflow area is filled, or if only an independent area is specified by an EXTENT job control statement, 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.

If records are to be added to a blocked file, a work area must be specified by the DTFIS WORKL operand. Each added record must contain a key field in the same location as the records already in the file. You must specify the high-order position of the key field (relative to the leftmost position of the logical record). Use the DTFIS KEYLOC operand for this purpose.

When a WRITE macro is issued, 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 WORKL) is low, the record 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 work area. 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 work area. 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 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, this condition is posted in filenameC.

If the new record is higher than all records presently in the file (end-of-file), ISAM checks to 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.

Random Retrieval of Records

Records in an ISAM file can be retrieved in random order for processing and/or updating. Retrieval must be specified in the DTFIS with the operand IOROUT=RETRVE or IOROUT=ADDRTR. Random processing must be specified in the DTFIS with the operand TYPEFLE=RANDOM or TYPEFLE=RANSEQ.

Because random reference to the file is by record key, your program must supply the key of the desired record. To do this, the key must be stored in the key field specified by the DTFIS KEYARG operand. 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 macros for a particular record to be updated) can result in a lost record and a duplicate key.

The DTFIS RECSIZE operand 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

Name	Operation	Operand
[name]	READ	$\left\{\begin{matrix}\text{filename}\\(1)\end{matrix}\right\}$, KEY

The READ macro causes ISAM to retrieve the specified record from the file. This macro requires two parameters. The first parameter specifies the name of the file from which the record is to be transferred to virtual 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 IOAREAR operand. The ISAM routines also move the record from the I/O area to the specified work area if the WORKR operand is included in the DTFIS.

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 in the work area (if specified). For processing in the I/O area, ISAM supplies the address of the record in the register specified by the DTFIS IOREG operand. The ID of the record can be referenced using filenameG. A WAITF macro must follow a READ macro.

WRITE Macro

Name	Operation	Operand
[name]	WRITE	$\left\{\begin{matrix}\text{filename}\\(1)\end{matrix}\right\}$, KEY

The WRITE macro with the parameter KEY is used for random updating. It causes ISAM to transfer the specified record from virtual storage to DASD storage. This macro 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 macro. 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 macro for the same file. The record is updated from the work area (if one is specified) or from the I/O area. The key need not be specified again ahead of the WRITE macro. A WAITF macro must follow a WRITE macro.

WAITF Macro

Name	Operation	Operand
[name]	WAITF	$\left\{\begin{matrix}\text{filename}\\(1)\end{matrix}\right\}$

The WAITF macro is issued to ensure that record transfer is completed. Filenname is the same name as that used in the DTFIS header entry, and can be specified as a symbol or in register notation.

This macro must be issued before your program attempts to process an input record which has been read or to 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 is specified in the DTFIS 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 macro posts any exceptional conditions in the DTFIS table at filenameC. The WAITF macro applies to the functions described in *Adding Records to a File* and *Random Retrieval of Records*, above.

Sequential Retrieval of Records

Records of an ISAM file can be retrieved in sequential order by key for processing and/or updating. The DTFIS IOROUT=RETRVE operand must be specified. Sequential processing must be specified in the DTFIS TYPEFLE=SEQNTL or RANSEQ operand.

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. You must specify, in SETL, the type of reference you use in your program.

Whenever the starting reference is by key and the file contains blocked records (RECFORM=FIXBLK), you must also provide ISAM with the position of the key field within the records. This is specified in the DTFIS KEYLOC operand. 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 file was loaded.

SETL Macro

Name	Operation	Operand
[name]	SETL	$\left\{\begin{array}{c}\text{filename}\\ \text{(r)}\end{array}\right\}, \left\{\begin{array}{c}\text{idname}\\ \text{(r)}\\ \text{KEY}\\ \text{BOF}\\ \text{GKEY}\end{array}\right\}$

The SETL (set limits) macro 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 same name as that used in the DTFIS header entry, as a symbol or in register notation. Register notation is necessary if the macro is to be used in a self-relocating program.

The second operand specifies where processing is to begin.

If you are processing by the record ID, the operand idname or (r) specifies the symbolic name of the 8-byte field in which you supply the starting (or lowest) reference for ISAM use. This field contains the information shown in Figure 4-10.

If processing begins with a key you supply, the second operand is KEY. The key is supplied in the field specified by the DTFIS KEYARG operand. If the specified key is not present in the file, an indication is given at filenameC.

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 (generic key) as the second operand. GKEY allows processing to begin at the first record (or key) within the desired group. You 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 processing to begin at the first record (or key) containing D642xxxx, regardless of the characters represented by the x's. Your 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).

Byte	Identifier	Contents in Hexadecimal	Information
0	m	02-F5	Number of the extent in which the starting record is located
1-2	bb	0000 (disk)	Always zero for disk
		0000-0009 (2321)	Cell number for data cell

3-4	cc	
		0000-00C7 (2311, 2314, 2319)
		0000-0193 (3330, 3333)
		0000-015B (3348 model 35)
		0000-02B7 (3348 model 70)
			Cylinder number for disk:
			for 2311, 2314, 2319 :0-199
			for 3330, 3333: 0-403
			for 3340 with 3348 model 35: 0-347
			for 3340 with 3348 model 70: 0-695

		0000-1309 (2321)	Subcell (byte 3) and strip (byte 4) for data cell **Note:** The last four strips on each cell are reserved for alternate tracks
5-6	hh	0000-0009 (2311) 0000-0013 (2314, 2319) 0000-0012 (3330, 3333) 0000-000B (3340)	Head position for disk
		0000-0413 (2321)	Cylinder (byte 5) and head (byte 6) for data cell
7	r	01-FF	Record location

Figure 4-10 **Field Supplied for SETL Processing by Record ID**

GET Macro

Name	Operation	Operand
[name]	GET	{filename / (1)}

The GET macro 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 the DTFIS IOAREAS operand). 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 your program. The key is located at the beginning of IOAREAS and the register (IOREG) points to the data. If the records are blocked, ISAM makes each record available by supplying its address in the register specified by the DTFIS IOREG operand. The key is contained in the record.

The second form of the GET macro is used if records are to be processed in a work area (specified by the DTFIS WORKS operand). It requires two parameters, both of which can be specified as symbols or in register notation. The first parameter is the name of the file, and the second is the name of the work area. When using register notation, workname should not be preloaded into register 1.

If the records are blocked, each GET that transfers a block of records to virtual storage will also write the preceding block back into the file in its previous location. GET writes the preceding block if a PUT macro is issued for at least one of the records in the block. If no PUT macro was 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

Name	Operation	Operand
[name]	PUT	{filename / (1)}

The PUT macro 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 the DTFIS IOAREAS operand). It requires only the name of the file to which the records are to be transferred. The name 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 is used if records are processed in a work area. 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 work area. When using register notation, workname should not be loaded into register 1. The work area 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 work area specified in the PUT macro to the I/O area specified for the file in the DTFIS IOAREAS operand.

When the records are unblocked, 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 GETs can be issued without intervening PUTs. Therefore, it is not necessary to rewrite unchanged records.

When the records are blocked, PUTs 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 virtual storage, the GET also writes the completed block back into the file in its previous location. If a PUT is not issued for any record in the block, GET does not write the completed block. The ESETL macro writes the last block processed, if necessary, before the end-of-file.

ESETL Macro

Name	Operation	Operand
[name]	ESETL	$\left\{\begin{matrix}\text{filename}\\(1)\end{matrix}\right\}$

The ESETL (end set limit) macro ends the sequential mode initiated by the SETL macro. Filename 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 the records are blocked, ESETL writes the last block back if a PUT was issued. Register notation is necessary if the macro is to be used 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 *DASD Track Protection Macros* section of the *Multitasking Macros* chapter.

Completion Macros

CLOSE and CLOSER Macros

Op	Operand
for self-relocating programs	
CLOSER	$\left\{\begin{matrix}\text{filename1}\\(r1)\end{matrix}\right\}$ $\left[,\left\{\begin{matrix}\text{filename1}\\(r2)\end{matrix}\right\}\ldots,\left\{\begin{matrix}\text{filenamen}\\(rn)\end{matrix}\right\}\right]$
for programs that are not self-relocating	
CLOSE	$\left\{\begin{matrix}\text{filename1}\\(r1)\end{matrix}\right\}$ $\left[,\left\{\begin{matrix}\text{filename2}\\(r2)\end{matrix}\right\}\ldots,\left\{\begin{matrix}\text{filenamen}\\(rn)\end{matrix}\right\}\right]$

The CLOSER or CLOSE completion macro must be used after the processing of a file is completed. These macros end the association of the logical file declared in your program with a specific physical file on a DASD.

The CLOSER or CLOSE macro deactivates any file that was previously opened. A file may be closed at any time by issuing this macro. 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 CLOSER is specified, the symbolic address constants that CLOSER generates from the parameter list are self-relocating. When CLOSE is specified, the symbolic address constants are not self-relocating.

To write the most efficient code in a multiprogramming environment it is recommended that CLOSER be used.

Enter the symbolic name of the file (assigned in the DTF header entry) in the operand field. A maximum of 16 files may be closed by one CLOSE or CLOSER by entering additional filename parameters as operands. Alternately, you 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. For CLOSER, the address of filename may be preloaded into any of the registers 2-15. For CLOSE, the address of filename may be preloaded into register 0 or any of the registers 2-15.

Note: If you use register notation, we recommend that you follow the standard practice of using only registers 2-12.

See the *Label Processing* chapter for information on label processing done by the CLOSE or CLOSER macro.

PART 5

VIRTUAL STORAGE ACCESS METHOD

CONCEPTS of VSAM

VSAM has key-sequenced and entry-sequenced files. The primary difference between the two is the sequence in which data records are stored.

Records are stored in a *key-sequenced file* in the collating sequence of a key field, such as employee number or invoice number. Each record must have a unique value in its key field. Like ISAM, VSAM uses an index to access records in a key-sequenced file. VSAM also allows free space to be distributed throughout the file so records can be inserted physically into the file. Therefore, separate overflow chains are not needed.

Records are stored in an *entry-sequenced file* in the physical sequence in which they are entered (loaded). New records are stored at the end of the file and records cannot be physically deleted or changed in length. An entry-sequenced file does not have an index.

Records in a key-sequenced file are accessed by their key fields, using the index of the file. This is called *keyed access.*

A record in an entry-sequenced file is addressed by its displacement, in bytes, from the beginning of the file. This is called *addressed access.* This displacement is the Relative Byte Address (RBA) of the record. The RBA does not depend on the location (cylinder and track) of the record on a direct-access volume. For relative byte addressing, VSAM considers the control intervals in the file to be contiguous, as though the file were stored in virtual storage beginning at address 0. When a record is loaded or subsequently added to an entry-sequenced file, VSAM indicates its RBA. You must keep track of the RBAs of the records to gain access to them directly.

Addressed access can also be used for a key-sequenced file, but previous keyed insertion, deletion, or update can change the RBAs of records. Therefore, the user may have to keep track of RBA changes if he wants to use addressed access. (VSAM passes back the RBA of each record retrieved, updated, added, or deleted.)

Fixed-length or variable-length records can be processed by VSAM. Record blocking is completely controlled by VSAM, so the user is not concerned with whether records are blocked or unblocked. VSAM also adds and removes control information to the records automatically, so you access only the data itself. You supply an area for VSAM to move records to and from its I/O buffer (move mode) or you supply four bytes in which VSAM will place the address of the records in its I/O buffer (locate mode).

VSAM stores the records of each type of file in a logical unit called a control interval. A *control interval* is a continuous area of direct processing storage in which VSAM stores data records and control information describing them. It is the unit of a file that VSAM transfers to and from direct processing storage and contains one or more physical blocks. Control intervals are grouped together in a logical unit called a control area. Control intervals and control areas and their relationship to the index and distributed free space of a key-sequenced file are explained in detail in the *DOS/VS Data Management Guide.*

Types of Processing

Keyed access allows the following types of processing:

- Sequential
- Skip Sequential
- Direct

With sequential processing, records are retrieved or stored in ascending key sequence starting from the beginning of the file or another position that you select. You do not have to supply a search argument (key) for VSAM to process the records.

With direct processing, records are retrieved or stored by the search argument (key) you supply. Records can be processed in any order, without regard to the key sequence of records processed before or after.

With skip sequential processing, a group of records can be retrieved or stored sequentially (in ascending key sequence) and then you can skip to a different part of the file and process another group of records sequentially. Skip sequential combines features of both sequential and direct processing.

Addressed access allows the following types of processing:

- Sequential
- Direct

With sequential processing, records are retrieved from a key-sequenced file or an entry-sequenced file in ascending RBA sequence or stored at the end of an entry-sequenced file in the order they are entered.

With direct processing, records are retrieved from a key-sequenced file or an entry-sequenced file by the search argument (RBA) you supply. Records can be added only to the end of an entry-sequenced file with direct processing.

Figure 5-1 summarizes the use of keyed and addressed access to retrieve, add, update, or erase records in key-sequenced and entry-sequenced files.

Types of Macros

VSAM must always run virtual. You code macros to:

- Specify an Access-Method Control Block (ACB) for the file you are going to process, specify a list of addresses of your own exit routines (EXLST), and specify a list of parameters for a particular request for access (RPL)
- Generate, modify, display, and test fields in an ACB, EXLST, or RPL
- Open and close an ACB to connect and disconnect the processing program and the file
- Request access to the file

In the VSAM macros, you can code an *address* (addr) as a symbolic name that generates a relocatable A-type address constant or as a register. You can code a *value* (n) as any absolute expression, except for a self-defining character term. You can code a *name* according to the rules of the assembler.

Some operands of the VSAM macros can have more than one parameter. These operands are shown with parentheses around the parameters. For example, the format of the MACRF operand of the ACB macro is shown as folllows:

MACRF=(option[,option...])

This means that you can code the operand, if it has only one parameter, with or without parentheses around the parameter:

MACRF=option

MACRF=(option)

However, if the operand is coded with two or more parameters, enclosing parentheses are required:

MACRF=(option,option,...)

System control programmers can gain addressed access to the index of a key-sequenced file by opening the index without the data. This chapter generally refers to access as access to a file, but access to an index only is also included.

Type of Request \ Macros Used			GET	PUT	GET for Update and PUT for Update	GET for Update and ERASE
Type of File	Type of Access	Type of Processing	Retrieve Records	Add Records	Update Records	Delete Records
key sequenced	keyed	sequential	yes	yes	yes	yes
		skip sequential	yes	yes	yes	yes
		direct	yes	yes	yes	yes
	addr.	sequential	yes	no	yes*	yes
		direct	yes	no	yes*	yes
entry sequenced	addr.	sequential	yes	to end	yes*	no
		direct	yes	to end	yes*	no

* The length of the records cannot be changed.

Figure 5-1 **Types of processing for keyed and addressed access**

CONTROL BLOCK GENERATING MACROS

The macros ACB, EXLST, and RPL produce an Access-Method Control Block (ACB), a user EXLST, and a Request Parameter List (RPL) when the macros are assembled. The GENCB macro can be used in place of these macros to generate an ACB, EXLST, or RPL when the processing program is executed.

ACB Macro

Assembly of the ACB macro produces an Access-Method Control Block (ACB). The ACB identifies the key-sequenced file or the entry-sequenced file that you want to process and indicates the types of requests that you want to make. The ACB is similar to a DTF in that it identifies the file to be processed. However, you specify most information (such as key length or record format) about the file in the DEFINE statement of Access Method Services. That information then resides in the VSAM catalog and is brought into virtual storage when the ACB is opened.

Name	Operation	Operand
name[1]	ACB	[BUFND=n] [,BUFNI=n] [,BUFSP=n] [,DDNAME=filename[1]] [,EXLST=addr] [,MACRF=(option [,option ...])] [,PASSWD=addr] [,STRNO=n]

1 The name of the ACB macro provides the symbolic address of the ACB. If you omit the DDNAME operand, the name is also used as the DLBL filename.

BUFND=n
This operand specifies the number of buffers to be used for control intervals containing data records. Each buffer is the size of a data control interval. The minimum number is one plus the number specified for the STRNO operand. (If you omit STRNO, BUFND must be at least two, because the default for STRNO is one). If the BUFND operand is omitted, the default is two, the smallest number of data buffers allowed.

VSAM will increase or decrease (not below the minimum required) the number of data buffers you specify if the amount of virtual storage available for buffers differs from the storage requirements indicated by the BUFND and BUFNI operands. See the BUFSP operand below for an explanation.

BUFNI=n
This operand specifies the number of buffers for index control intervals (index records). Each buffer is the size of an index control interval. The minimum number is the number specified for the STRNO operand. (If you omit STRNO, BUFNI must be at least one, because the default for STRNO is one). If the BUFNI operand is omitted, the default is one, the smallest number of index buffers allowed.

VSAM will increase or decrease (not below the minimum required) the number of index buffers you specify if the amount of virtual storage available for buffers differs from the storage requirements indicated by the BUFND and BUFNI operands. See the BUFSP operand below for an explanation.

BUFSP=n
This operand specifies the size, in bytes, of an area for data and index buffers. VSAM issues a GETVIS macro to obtain the buffer area in your processing partition. It must be at least as large as the buffer space size recorded in the catalog entry of the file or the ACB will not be opened.

If the BUFSP operand is omitted, the buffer space size will be the larger of (1) the size recorded in the catalog or (2) a size determined from the value specified for BUFND and BUFNI. The size recorded in the catalog was specified by the BUFFERSPACE parameter in the DEFINE statement of Access Method Services. If that parameter was omitted when the file was defined, a default value was set in the catalog by Access Method Services. This default value, the minimum amount of buffer space allowed by VSAM, is enough space for two data control intervals and one index control interval.

If the values you code for BUFND, BUFNI, and BUFSP are not consistent with each other, VSAM increases or decreases the number of buffers to conform to the size of the buffer area.

If BUFSP is greater than the minimum requirements and greater than the BUFND and BUFNI requirements, the extra space will be allocated between data and index buffers as follows:

- The ACB indicates direct processing only: BUFND and BUFNI are allocated as specified. Then, all additional space is allocated to index buffers.

- The ACB indicates sequential processing: BUFND and BUFNI are allocated as specified. Then, one additional buffer is allocated to the index and the remaining space is allocated to data buffers. If there is still space remaining, and it is insufficient for a single data buffer, it will be allocated to an index buffer.

If BUFSP is greater than the minimum requirements, but less than the BUFND and BUFNI requirements, BUFND and BUFNI will be reduced as follows in order to comply with BUFSP:

- The ACB indicates direct processing only: BUFND is reduced first. Then, if required, BUFNI is reduced until space requirements comply with BUFSP.

- The ACB indicates sequential processing: Reduce the number of index buffers to not less than one more than the minimum number. Then, if required, reduce the number of data buffers to not less than the minimum number. If the space required for these buffers still exceeds BUFSP, reduce the number of index buffers to the minimum number.

If you provide your own pool of I/O buffers for control interval (CNV) access (MACRF=UBF), the BUFND, BUFNI, and BUFSP operands have no effect. The AREA and AREALEN parameters in the RPL define the area for user buffers.

DDNAME=filename
This operand specifies a character string of up to eight bytes and is the same as the filename parameter specified in the DLBL statement that defines the key-sequenced file or the entry-sequenced file to be processed. If you omit this operand, you must specify the DLBL filename as the name (label) of the ACB macro.

EXLST=addr
This operand specifies the address of a list of user exit-routine addresses. The list is generated by the EXLST macro (or the GENCB macro) and can be omitted if you have no exit routines. If you use the EXLST macro, this operand will contain the symbol (address) from the label field of the EXLST macro. If you use the GENCB macro, this operand will contain the address of the EXLST returned by GENCB in register 1. Omitting this operand indicates that you have no user exit routines.

MACRF=(option[,option,...])
This operand specifies the types of access to be gained to the file. Options are arranged in four groups, and each group has a default value. You can specify one or more options for each group, in any combination, except as noted in parentheses. However, you cannot specify both NUB and UBF. You must specify all types of access you are going to use, whether you will use them concurrently or by switching from one to another.

Option	**Meaning**
<u>KEY</u>	Keyed access (key-sequenced file with index only).
ADR	Addressed access
CNV	Control-interval access
<u>SEQ</u>	Sequential processing
DIR	Direct processing
SKP	Skip sequential processing (keyed processing only)
<u>IN</u>	Retrieve records only
OUT	Retrieve, insert, add-to-end, or update records (keyed access); retrieve or add-to-end (addressed access)
<u>NUB</u>	No user buffers; VSAM supplies buffers for I/O operations (KEY, ADR, and CNV access).
UBF	User buffers (only CNV access can be specified). VSAM will read and write control intervals in a buffer you supply. It is pointed to by the AREA parameter of the RPL.

PASSWD=addr
This operand specifies a pointer to a length byte followed by the 8-byte password required to authorize the type of access you have specified to the file. If you omit the operand and a password is required, the console operator can supply it. If the length byte is zero, the operand is considered omitted. If the length of the password is less than eight bytes, OPEN pads it with blanks to eight bytes. If the length of the password is greater than eight bytes, OPEN uses only the first eight bytes.

STRNO=n
This operand indicates how many requests requiring concurrent data-set positioning VSAM is to be prepared to handle. A request is defined by a given

request parameter list or chain of request parameter lists. For convenience, you could specify for STRNO the total number of request parameter lists or chains of parameter lists that you are using to define requests. (VSAM needs to remember only one position for a chain of request parameter lists). However, each position beyond the minimum number that VSAM needs to be able to remember requires additional virtual-storage space for:

- A minimum of one data I/O buffer and, for keyed access, one index I/O buffer (the size of an I/O buffer is the control interval size of a data set)
- Internal control blocks and other areas (about 600 bytes for addressed access: about 1200 bytes for keyed access)

The internal control blocks and other areas are fixed in real storage, so 600 or 1200 bytes of real storage is tied up for each position beyond the minimum. To save this space, you should specify only the number of requests that require concurrent positioning.

VSAM remembers its position in the data set for any sequential or update request. For example, sequential access depends on VSAM's being able to determine the location of the next record from the location of the present record. Updating or deleting a record depends on VSAM's remembering its location after you retrieve it. Also, processing a record in line I/O buffer requires VSAM to remember its location in the buffer.

Even read-only direct retrieval into a work area can make use of positioning because you may modify the type of access from direct to sequential. (You specify for VSAM to remember its position for a direct request by OPTCD=NSP in the RPL macro.

The ENDREQ macro enables you to cause VSAM to forget the position for a request so it can remember the position for another request. If the number specifed for STRNO isn't as large as the number of request parameter lists or chains of request parameter lists, you may have to issue the ENDREQ macro.

EXLST Macro

Assembly of the EXLST macro produces an optional list of addresses of user exit routines. An exit routine is entered when VSAM detects the condition (such as I/O error) that the routine is supposed to handle.

The Exit List (EXLST) is associated with an ACB by the EXLST operand of the ACB macro. Two or more ACBs can refer to the same EXLST.

The number of exit addresses in a list is variable and depends on the number of operands you code. You cannot add addresses to the list after it is generated, but you can change an address or the indication of whether an exit is active (with the MODCB macro).

Name	**Operation**	**Operand**
[name]	EXLST	[EODAD=(addr [, {A | N}][,L])] [,EXCPAD=(addr [, {A | N}][,L])] [, LERAD=(addr [,{A | N}][,L])] [,SYNAD=(addr [,{A | N}][,L])][1]

1 The address must always be specified first, but L can be specified before A or N.

The common optional subparameter {A | N},of which A is the default, indicates whether an exit is active (A) or not active (N). VSAM does not enter a routine whose exit is marked not active. The subparameter L indicates that the address specified gives the location of an 8-byte field containing the name of an exit module in the core image library that VSAM is to load for exit processing. If L is omitted, the address gives the entry point of the exit routine in virtual storage.

When VSAM enters an exit routine, register 13 contains the address of a 72-byte save area that VSAM is already using. If the exit routine returns to VSAM it must not have altered the contents of the save area or of register 13. It can, however, use all other registers without saving their contents (as long as the return address in register 14 is saved).

EODAD=(address[,{A | N}][,L])
This operand specifies the address of a routine that finishes the processing of a file when VSAM reaches the end of the file. VSAM exits to this routine when there is no more data after a GET request or a PUT for update request when you have specified control-interval access and user buffers. With addressed access, there are no more records in entry sequence; with keyed access, there are no more records in key sequence.

If you do not have this exit routine, VSAM exits to the routine for analyzing logical errors (see the LERAD operand). If you do not have the LERAD exit routine, VSAM returns to your processing program at the instruction following the last executed instruction: register 15 contains X'08', and register 1 contains the address of the RPL. Your program can examine the feedback field in the RPL with the SHOWCB or TESTCB macro to discover that VSAM reached the end of the file. See "Requesting Access to Files", later in this chapter, for the return codes for logical errors.

When VSAM exits to the EODAD routine, the contents of the registers are:

Register	**Contents**
0	Unpredictable
1	Address of the RPL
2-12	Same as when the request macro was issued
13	Address of user's 72 byte save area. (It must be preserved for return to VSAM).
14	Address of return to VSAM
15	Address of the exit routine

If the EODAD exit routine returns to VSAM and you issue another GET macro, VSAM enters the EODAD exit routine again. This can cause your program to loop. If, however, you reach end-of-file during keyed access and then change to addressed access, additional records may be retrieved if they are physically after the last record in key sequence (because of a control interval or control area split).

EXCPAD=(address[,{A̲ | N}][,L])
This operand specifies the address of a routine that will receive control from VSAM when an I/O operation is started. It is intended for use by programmers of utilities and systems. By supplying an EXCPAD exit routine, you can overlap VSAM I/O operations with execution of your processing program.

The exit routine must return to VSAM, so that VSAM can return to you mainline program at the instruction following the I/O request macro.

When VSAM exits to the EXCPAD routine, the contents of the registers are:

Register	**Contents**
0	Unpredictable
1	Address of a parameter list with the following contents: X'0' address of the RPL X'4' address of the CCB X'8' EXCPAD lock word
2-13	Same as when the request macro was issued. (Register 13 points to the problem program's save area.)
14	Address of return to VSAM
15	Address of the exit routine

The EXCPAD routine can test the traffic bit of the CCB to determine whether the VSAM I/O operation has been completed. However, the contents of the CCB cannot be changed because it will be used by VSAM.

The EXCPAD lock word may be zero or it may contain an address. If the lock word is zero, the EXCPAD routine may **not** issue a request to insert a record or update a record when the length will change. When the lock word is not zero, the request being executed may cause a control-interval or control-area split, and a system deadlock could result if the EXCPAD routine inserted or changed the length of a record. The lock word will contain the address of the RPL for the active request.

The EXCPAD routine may be entered more than once for a VSAM request, because a request may require more than one I/O operation.

LERAD=(address[,{A̲ | N}][,L])
This operand specifies the address of a routine that analyzes logical errors encountered by VSAM during execution of a GET, PUT, POINT, or ERASE macro. The routine determines what error has occurred by issuing a SHOWCB or TESTCB macro to examine the feedback field (FDBK) in the RPL. The contents of FDBK will be 000000xx, where xx is the error code which indicates the type of error. See *Requesting Access to Files*, later in this chapter, for the error codes for logical errors.

If the routine cannot correct the error, it should either:

- Close the file or index, or
- Return to VSAM (which will return to your processing program at the instruction following the last executed instruction)

If you do not have the LERAD exit routine and VSAM encounters a logical error, VSAM returns to your processing program at the instruction following the last executed instruction: register 15 contains X'08', and register 1 contains the address of the RPL. Your program can examine the feedback field in the RPL with the SHOWCB or TESTCB macro to identify the logical error. (See *Requesting Access to Files*, later in this chapter, for the error codes for logical errors.)

When VSAM exits to the LERAD routine, the contents of the registers are:

Register	Contents
0	Unpredictable
1	Address of the RPL
2-12	Same as when the request macro was issued
13	Address of user's 72-byte save area. (It must be preserved for return to VSAM).
14	Address to return to VSAM
15	Address to enter the exit routine

SYNAD=(address[,{A̲ | N}][,L])
This operand specifies the address of a routine that can anlyze physical I/O errors, detected by VSAM during execution of a GET, PUT, POINT, ERASE, or CLOSE macro, that the system error routine was unable to correct. The exit routine determines what error has occurred (reading or writing data or index) by issuing a SHOWCB or TESTCB macro to examine the feedback field (FDBK) in the RPL. The contents of FDBK will be 000000xx, where xx is the error code which indicates the type of error. See "Requesting Access to Files", later in this chapter, for the error codes for physical errors.

If the routine cannot correct the error, it should close the file and end the job or do other processing. If the error occurred while VSAM was closing the file or index, and if another error occurs after the exit routine issues a CLOSE macro, VSAM does not exit to the routine a second time.

If the exit routine returns to VSAM, whether the error was corrected or the file closed, VSAM drops the request and returns to your processing program at the instruction following the last executed instruction.

If you do not have this exit routine and VSAM detects a physical error, VSAM returns to your processing program at the instruction following the last executed instruction: register 15 contains X'0C', and register 1 contains the address of the RPL. Your program can examine the feedback field in the RPL with the SHOWCB or TESTCB macro to identify the physical error. (See "Requesting Access to Files" for the error codes for physical errors.)

An error in transmitting the contents of a control interval to an I/O buffer (read error) during the execution of a sequential GET request positions VSAM at the next control interval in key sequence for keyed access or in entry sequence for addressed access. The next GET after the error will return the first record from the control interval following the one with the error. For processing an index, VSAM is not positioned at the next index control interval, and you should not attempt to process further.

Errors occurring during writing of a control interval do not change positioning. When VSAM exits to the SYNAD routine, the contents of the registers are:

Register	Contents
0	Unpredictable
1	Address of the RPL
2-12	Same as when the request macro was issued
13	Address of the user's 72-byte save area. (It must be preserved for return to VSAM).
14	Address to return to VSAM
15	Address to enter the exit routine

RPL Macro

Assembly of the RPL macro produces a Request Parameter List (RPL). Each request must be defined by an RPL or an chain of RPLs. Each request is associated with a position in the file. There is one position for a single RPL or a chain of RPLs.

Name	Operation	Operand
[name]	RPL	[ACB=addr][,AREA=addr] [,AREALEN=n] [,ARG=addr] [,KEYLEN=n] [,NXTRPL=addr] [,OPTCD=(option [,option...])][,RECLEN=n

If the ACB specifies concurrent requests, either (1) a single RPL or a chain of RPLs is supplied for each concurrent request or (2) one request must be disconnected from its RPLs (by an ENDREQ macro) before another concurrent request is issued. The parameters required to access a record (such as the address of the work area VSAM will move your records to) are in the RPL instead of in each action macro (GET, PUT, etc.). The list does not indicate a specific action (GET or PUT , for example); a single RPL can be used, without modification, for different actions. However, if you want to use the same RPL for a different type of request (for both direct and sequential processing, for example), you must modify the RPL (with the MODCB macro) each time you change from one type of request to another.

ACB=addr
This operand specifies the Acccess-Method Control Block (ACB) associated with the file you are gaining access to. This operand is required, but you can specify it through a MODCB macro when the program is executed. The operand must be specified before a request is issued against the RPL.

AREA=addr
This operand specifies the address of your I/O work area to which VSAM moves the record (OPTCD=MVE) for GET and PUT requests. You process the record in this work area. If you process the records in VSAM's I/O buffer (OPTCD=LOC), this option contains the address of the record in the I/O buffer (GET only).

When you specify user buffers (MACRF=UBF in the ACB) for control-interval (CNV) access, AREA specifies the address of a single I/O buffer. VSAM uses the buffer to read and write control intervals.

AREALEN=n
This operand specifies the length of the work area. For OPTCD=MVE, the work area must be large enough to contain the largest record retrieved. For OPTCD=LOC, the work area must be at least 4 bytes long to contain the address of the record in the I/O buffer. For control interval access (OPTCD=CNV), the work area must be at least the size of a control interval.

ARG=addr
This operand specifies the address of a field that contains the search argument (key for OPTCD=KEY or RBA for OPTCD=ADR) for:

- Direct retrieval (GET)
- Skip sequential retrieval (GET)
- Sequential positioning (POINT)

With keyed access, the search argument may be a full key or a generic key. If it is a generic key, you must also specify its size in the KEYLEN operand. To learn the RBA of a record to which you have gained access sequentially or directly by key, you can use the SHOWCB macro to display the RBA of the last record processed. (See the section "SHOWCB Macro" later in this chapter.)

When recors are inserted (PUT), VSAM obtains the key from the record itself.

KEYLEN=n
When a generic key is used as a search argument (OPTCD=GEN), this operand specifies the length of the generic key in number of bytes. KEYLEN can be any value from 1 to 255.

If, for example, the full key is 50 bytes long and KEYLEN=10 is specified, VSAM uses the leftmost 10 bytes of the 50-byte key field for comparison with the search argument. The length of the full key is in the catalog. It can be obtained through the KEYLEN parameter of the SHOWCB macro. You place the key (full or generic) in a field pointed to by the ARG parameter.

NXTRPL=address
(Required for chaining request parameter lists.) You can optionally define a single GET or PUT request with two or more request parameter lists chained together. Each list indicates a separate data record, so that when you issue GET, for example, you cause VSAM to retrieve two or more records.

The NXTRPL operand gives the address of the next request parameter list in a chain. Omit this operand from the macro that generates the last RPL in the chain. When you issue a request that is defined by a chain of request parameter lists, indicate in the request macro the address of the first parameter list in the chain.

Each request parameter list in a chain must have the same OPTCD options. Having different options may cause logical errors. You can't chain request parameter lists for updating or deleting records - only for retrieving records or storing new records. You can't process records in the I/O buffer with chained request parameter lists. (OPTCD=UPD, LOC is invalid for a chained request parameter list.)

With chained request parameter lists, a POINT, a sequential or skip sequential GET, or a direct GET with positioning requested (OPTCD=NSP) causes VSAM to position itself at the record following the record identified by the last request parameter list in the chain.

OPTCD=(option[,option...])
This operand specifies the type of access to be gained to the file through the requests defined by this RPL. Options are arranged in six groups, and each group has a default value. You can specify only one option in each group--if your ACB indicates both sequential and direct processing, for example, you must modify the RPL when you switch from one to the other. However, the groups that are not re-

quired are ignored. Thus you can use the same RPL for a combination of requests (GET, PUT, POINT, for example) without zeroing out the in applicable options each time you go from one request to another.

Option	Meaning
<u>KEY</u>	Keyed access (key-sequenced file with index only). You can change from keyed to addressed access at any time without positioning. If you change from keyed to control-interval access, the results are unpredictable. No error code will be issued.
ADR	Addressed access (key-sequenced and entry sequenced files). If you change from addressed to keyed access, you must reestablish positioning or the request will terminate with an error. If you change from addressed to control interval access, the results are unpredictable. No error code will be issued.
CNV	Control-interval access (provided for special applications such as utilities). OPTCD=MVE is required. If you change from control-interval to keyed access, you must reestablish positioning or the request will terminate with an error. If you change from control interval to addressed access, the results are unpredictable. No error code is issued.
<u>SEQ</u>	Sequential processing.
DIR	Direct processing
SKP	Skip sequential processing (keyed access only).
<u>NUP</u>	Request is not for update (you will not update or delete a record you are retrieving; a record you are storing is new). For a direct request, positioning will be released.
NSP	For direct processing only, request is not for update, and VSAM will be positioned at the next record for subsequent sequential processing.
UPD	Request is for update; you must issue a GET for update before you issue a PUT for update or an ERASE. However, if you supply your own buffers for control-interval access, you can issue a PUT for update without a preceding GET.
<u>KEQ</u>	The search argument must equal the key of the data record (keyed direct or skip sequential retrieval or keyed sequential pointing).
KGE	If the search argument does not equal the key of a record the request applies to the record with the next greater key (key direct or skip sequential retrieval or keyed sequential pointing).
<u>FKS</u>	The entire key is to be used for a search argument (keyed direct or skip sequential retrieval or keyed sequential pointing).
GEN	A generic key is to be used for a search argument (keyed direct or skip sequential retrieval or keyed sequential pointing: you specify the length of the generic key with the KEYLEN operand).
<u>MVE</u>	For retrieval and storage, VSAM moves a data record between the I/O buffer and your work area. MVE must also be specified when you supply your own buffers for control-interval access.
LOC	For retrieval, you can process the record in VSAM's I/O buffer. VSAM will pass you a pointer to the record in the buffer. If you want to update the record, you will have to move it to your work area before issuing a PUT macro (OPTCD=MVE).

RECLEN=n
This operand specifies the length of a data record stored by a PUT request. For fixed length records, the length need only be set once. For GET requests, VSAM indicates the length of the record in this field. To process a file with records of different lengths you can examine the field with the SHOWCB or TESTCB macro and modify it with the MODCB macro.

Examples of ACB, EXLST, and RPL Macros

The following examples show specification of the VSAM control blocks by using the ACB, EXLST, and RPL macros. The control blocks are created at assembly time. Default values will be supplied for the parameters that are omitted. Note that the Filename of the file will be ACBAD, the label of the ACB macro.

```
ACBAD   ABC     EXLST=EXITS,PASSWD=PASS
                BUFND=4,BUFNI=3,BUFSP=110
                MACRF=(KEY,SEQ,DIR,OUT)
EXITS   EXLST   EODAD=(ENDUP,N),
                LERAD=LOGERR,
                SYNAD=(IOERR,L)
         RPL    ACB=ACBAD,AREA=WORK,
                ARG=SEARCH,AREALEN=125,
                OPTCD=(DIR,NSP)
          .
          .
          .
PASS      DC    L1'6',C'CHANGE'
WORK      DC    125F
SEARCH    DS    3F
```

CONTROL BLOCK MANIPULATING MACROS

An Access-Method Control Block (ACB), Exit List (EXLST), and a Request Parameter List (RPL) can be dynamically generated, modified, displayed, or tested when your program is executed. You can use the GENCB macro to create a control block or list. You can use the MODCB macro to modify blocks or lists dynamically, the SHOWCB macro to display selected fields, and the TESTCB macro to test the value of selected fields. The advantage of these macros, called the control block manipulation macros, is that you need not be concerned with changes in the formats of the ACB, RPL, and EXLST or with the displacements of the fields in them.

With GENCB and SHOWCB, you can specify an area for VSAM to build the block or list or you can let VSAM supply an area. If you want your program to be reentrant, you should either let VSAM supply the area or you should use a GETVIS macro to supply the area when your program is executed.

Each control block manipulation macro has a parameter list (not the same as an RPL) which is built when the macro is assembled. This parameter list is used by VSAM routines during program execution to generate, modify, display, or test the ACB, EXLST, or RPL. Parameter lists can be built without using the control block manipulation macros, as described in Appendix I. To allow you make your program reentrant or to share macro parameter lists, the GENCB, MODCB, SHOWCB, and TESTCB macros have four forms:

- The standard form builds a parameter list inline along with the instructions produced by the macro. The standard form does not allow reentrant code or shared parameter lists.

- The list form builds a parameter list alone; it does not include the instructions produced by the macro. The list form is used together with the execute form to allow you to share parameter lists between macros and, if the list is built outside your code, to make your program reentrant. If the list is built outside your code, you must obtain the area for it during program execution by a GETVIS macro.

- The execute form contains the instructions produced by the macro to take action on a parameter list; you can also use it to change the parameter list before VSAM acts on it. The execute form is used together with the list form to allow you to share parameter lists between macros.

- The generate form produces both a parameter list and executable instructions. It builds the parameter list in an area you specify. You must obtain the area during program execution by a GETVIS macro.

The following descriptions of each macro show the standard form. The list, execute, and generate forms of the macros are nearly identical to the standard forms. They are discussed together at the end of this section.

When you issue a GENCB, MODCB, SHOWCB, or TESTCB macro, register 13 must contain the address of a 72-byte save area that you are providing.

GENCB Macro

The GENCB macro generates an ACB, an EXLST, or an RPL when it is executed. You can use it in place of the ACB, EXLST, and RPL macros to avoid reassembling your programs should the format of the control block or lists change, and to generate more than one copy of a control block or list.

GENCB generates the control block(s) or list(s) either in an area you specify or, if you do not specify an area, in an area obtained by VSAM (by the GETVIS macro) in your partition.

The standard form of the GENCB macro is:

Name	Op	Operand
[name]	GENCB	BLK={ACB | EXLST | RPL} [,COPIES=n][,keyword= {addr | n | option}...] [,LENGTH=n] [WAREA=addr]

BLK={ACB | EXLST | RPL}
This operand specifies whether you want to generate an ACB, an EXLST, or an RPL.

COPIES=n
This operand specifies the number of blocks or lists you want to generate. The default is 1. If you generate two or more, they are generated next to each other. They are identical, so you must use MODCB to tailor them for a particular file or request. Register 0 contains the total length of the block(s) or list(s) that were generated.

keyword={addr | n | option} ...
The operands you code are identical to those of the ACB, EXLST, and RPL macros, except that you can code the operands in more ways as described in Appendix H. If you do not code any operands, VSAM builds:

- For ACB, an ACB with default values provided by VSAM when you open the file. You must supply the DDNAME=filename operand before the file is opened.
- For EXLST, a complete EXLST with zeros for addresses and all entries flagged inactive
- For RPL, an RPL with default values

LENGTH=n
This operand specifies the length of the area, if any, you provided by the WAREA operand. You can determine the length of an existing block or list by using the SHOWCB macro.

WAREA=addr
This operand specifies the address of an optional area in which you want VSAM to generate the block(s) or list(s). The area must begin on a fullword boundary. If WAREA is specified, the LENGTH operand must also be specified. If you do not specify an area, VSAM obtains an area in your processing partition (with a GETVIS macro) and gives you its address in register 1 and its length in register 0.

Examples of the GENCB Macro The following examples show specification of the VSAM control blocks by using the GENCB macro. With GENCB, the control blocks are created dynamically during execution of the program. The same parameters are specified in this example as were specified in the previous example of ACB, EXLST, and RPL macros. VSAM issues a GETVIS macro to obtain space for each control block in your partition. The address of each block is set in register 1 after the GENCB is executed.

```
*    GENERATE VSAM CONTROL BLOCKS
*
         GENCB    BLK=ACB,EXLST=(3),PASSWD=PASS,BUFND=4,BUFNI=3,
                  BUFSP=11064,MACRF=(KEY,SEQ,DIR,OUT),DDNAME=VFILENM
         LTR      15,15                 (GENCB successful?)
         BNZ      GENERR                (No, go to error routine)
         LP       2,1                   (Yes, save ACB address)
*
         GENCB    BLK=EXLST,EODAD=(ENDUP,N),LERAD=LOGERR,SYNAD=(IOERR,L)
         LTR      15,15                 (GENCB successful?)
         BNZ      GENERR                (No, go to error routine)
         LR       3,1                   (Yes, save EXLST address)
*
         GENCB    BLK=RPL,AREA=WORK,AREALEN=125,OPTCD=(DIR,NSP),
                  ARG=SEARCH,ACB=(2)
         LTR      15,15                 (GENCB successful?)
         BNZ      GENERR                (No, go to error routine)
         LR       4,1                   (Yes, save RPI address)
*
*    PROCESSING ROUTINES

         GET      RPL=(4)
*
*    CONSTANTS AND WORK AREAS
*
PASS     DC       FL1'6',C'CHANGE'
WORK     DS       125F
SEARCH   DS       3F
```

MODCB Macro

The MODCB macro modifies the addresses, values, options, and names that you can establish with the ACB, EXLST, RPL, and GENCB macros in an ACB, EXLST, or RPL.

The standard form of the MODCB macro is:

Name	OP	Operand
[name]	MODCB	{ACB | EXLST | RPL}=addr ,keyword={addr | n | option}...

{ACB | EXLST | RPL}=addr
This operand specifies whether you want to modify an ACB, an EXLST, or an RPL and specifies its address. You cannot modify an open ACB. You can modify a field in an EXLST at any time, but you cannot add entries to or delete entries from it. You cannot modify an active RPL: that is, one that defines a request that has been issued but not completed.

With the execute form of MODCB, you can change the address of the block or list to be modified, but not the type.

keyword={addr | n | option}...
The operands you code are identical to those for the ACB, EXLST, and RPL macros, except that:

- You can code the operands in more ways, as shown in Appendix H.
- There are no defaults for the options of the ACB MACRF operand or the RPL OPTCD operand. With OPTCD, when you set on a new option with the MODCB macro, the old option is automatically turned off, since you can specify only one option in each of its seven groups (see *RPL Macro*).
- You can make an address in an EXLST active or not active without specifying the address (keyword={A | N}).
- When you specify an address for an entry in an EXLST that previously contained zeros (possible if you generated a default list with the GENCB macro), you must code keyword=(addr,A) to make the address active, because A is not a default for the MODCB macro.

Examples of the MODCB Macro The following examples show modification of VSAM control blocks by using the MODCB macro.

The first example shows the use of MODCB to place the length of a record in the RPL when variable-length records are being added to a file:

```
MODCB    RPL=(4),RECLEN=(7)  (Current length in reg.7)
LTR      15,15               (MODCB successful?)
BNZ      MODERR              (No, go to error routine)
PUT      RPL=(4)             (Yes, write record)
```

The second example shows the use of MODCB to activate the EODAD exit specified in the previous GENCB example:

```
MODCB    EXLST=(3),EODAD=A
LTR      15,15               (MODCB successful?)
BNZ      MODERR              (no go to error routine)
```

SHOWCB Macro

The SHOWCB macro displays fields in an ACB, EXLST or RPL. VSAM places these fields in an area that you provide. They are independent of the format of the block or list you are displaying: the fields are displayed in the order that you specify the keywords for them.

The standard form of the SHOWCB macro is:

Name	Op	Operand
[name]	SHOWCB	[{ACB | EXLST | RPL)=addr] ,AREA=addr,FIELDS= (keyword[,keyword...]) ,LENGTH=n [OBJECT={DATA | INDEX}

{ACB | EXLST | RPL}=addr
This operand specifies whether you want to display an ACB, an EXLST, or an RPL and specifies its address.

In the standard and list forms of SHOWCB, you can omit this operand if you are displaying only the standard length of a control block or list (see "Length of a Control Block or List" under the keyword operand). With the execute form of SHOWCB, you can change the address of the block or list to be displayed, but not the type.

AREA=addr
This operand specifies the address of the area in virtual storage that you are providing for VSAM to display the items you specify in the FIELDS operand. The items are in the area in the order you specify the keywords. The area must begin on a fullword boundary.

FIELDS=(keyword[,keyword...])
There are three groups of keywords that you can code for the FIELDS operand of the SHOWCB macro:

- The keywords that you can code with the ACB, EXLST, RPL, and GENCB macros
- The length of an ACB, RPL, or EXLST
- The attributes of an open file or index indicated by the ACB

Keywords of the ACB, EXLST, and RPL Macros
The keywords in this group require one fullword each for display, except DDNAME which requires two fullwords. The keywords are identical to those of the ACB, EXLST, and RPL macros, except that:

- You can code the operands in more ways, as shown in Appendix H.
- You do not code the address, value, option, or name to which the keyword is equal.
- In relation to the ACB macro, you cannot display the MACRF options, but you can display, with the keyword ERROR, the error code (in the rightmost byte of the display word) from the Open or Close routine (see "Opening and Closing Files"). (You can test the MACRF options with the TESTCB macro.)
- In relation to the EXLST macro, you cannot display the codes that indicate whether an exit address is active or not active or is the address of the name of a routine to be loaded (you can test them with the TESTCB macro).
- In relation to the RPL macro, you cannot display the OPTCD options, but you can code the keyword FDBK to display error codes from the request macros and the keyword RBA to display the relative byte address of the last record processed. (You can test the OPTCD options with the TESTCB macro.)

Length of a Control Block or List
You can code the keyword ACBLEN, EXLLEN, or RPLLEN to display either the standard length of an ACB, EXLST, or RPL, or the actual length of a particular block or list. You display a standard length by omitting the {ACB | EXLST | RPL} operand and coding only one (or more) of these length keywords and no other keywords. You display the actual length of a block or list by specifying the {ACB | EXLST | RPL} operand and the corresponding length keyword.

Attributes of an Open File
After a file is opened, the ACB contains information that it does not contain before it is opened or after it is closed. Whether you are displaying the attributes of the data or the index of a key-sequenced file is determined by the OBJECT operand. Each item displayed requires one fullword in your work area except STMST which requires two fullwords. You can display the following items:

Operand	Meaning
AVSPAC	Number of bytes of available space in the data or the index.
BUFNO	Number of buffers being used for the data or the index.

CINV	Size of the control interval in the data or the index.
FS	Percent of the free control intervals in each data control area of a key-sequenced file.
KEYLEN	Full length of the key field in each logical record.
LRECL	Maximum length of a logical record or, for an index, the index control interval size minus seven bytes.
NCIS	Number of control-interval splits in the file.
NDELR	Number of data records deleted from a key-sequenced file.
NEXCP	Number of EXCP commands issued since the data or the index was opened.
NEXT	Number of logical extents, data spaces or portions of data spaces, of the data or of the index.
NINSR	Number of data records inserted into the file.
NIXL	Number of levels in the index of a key-sequenced file.
NLOGR	Number of data records in the file.
NRETR	Number of data records retrieved from the file.
NSSS	Number of data control area splits in the key-sequenced file.
NIPDR	Number of data records updated in the file.
RKP	Displacement of the key-field from the beginning of the data record: the same value is displayed whether the object is index or data.
STMST	System time stamp; the time and day (in microseconds) when the data or index was last closed. Bits 52-63 of the field are unused.

LENGTH=n

This operand specifies the length of the display area you are providing (by way of the AREA operand). Each field in the ACB and RPL takes a fullword, except for DDNAME and STMST in the ACB, which take two fullwords. Each EXLST operand takes only one fullword, since you cannot display the codes A, N, and L.

OBJECT={DATA | INDEX}

This operand specifies, for the open ACB of a key-sequenced file, whether the fields being displayed are for the data or the index. KEYLEN and RKP will contain the same value if the data or the index is being displayed. FS, NCIS, NDELR, NINSR, NIXL, NLOGR, NRETR, NSSS and NUPDR will contain zeros if the index is being displayed.

Examples of the SHOWCB Macro The following examples show the use of the SHOWCB macro to display information from VSAM control blocks.

The first example shows the use of SHOWCB to display statistics about an open file:

```
          SHOWCB   ACB=(2),AREA=DISPLAY,LENGTH=12,
                   FIELDS=(KEYLEN,LRECL,RKP)
          LTR      15,15               (SHOWCB successful?)
          BNZ      SHOWERR             (No, go to error routine)
          .
          .
DISPLAY   DC       0F                  (Align on fullword boundary)
KEYLEN    DS       F
LRECL     DS       F
RKP       DS       F
```

The second example shows the use of SHOWCB to display the length and REA of a record that has been retrieved:

```
          GET       RPL=(4)
          SHOWCB    RPL=(4),AREA=DISPLAY,LENGTH=8,
                    FIELDS=(RECLEN,RBA)
          LTR       15,15               (SHOWCB successful?)
          BNZ       SHOWERR             (No go to error routine)
          .
          .
DISPLAY   DC        0F                  (Align on fullword boundary)
LENGTH    DC        F
RBA       DC        F
```

TESTCB Macro

The TESTCB macro tests values in an ACB, EXLST, or RPL against values that you specify in the macro.

You examine the condition code after issuing a TESTCB macro and examining the return code in register 15. For keywords specified as an option (such as A for an operand of the EXLST macro), a test is for an equal or unequal comparison; for keywords specified as an address or value, a test is for an equal, unequal, high, low, not-high, or not-low comparison. In the comparison, A to B, B is the address, value, or option that you specify in the TESTCB macro. For example, if you test for a value in an ACB, a high comparison means the value in the block is higher than the value you specified in the TESTCB macro.

The standard form of the TESTCB macro is:

Name	Op	Operand
[name]	TESTCB	[{ACB | EXLST | RPL}=addr] [,ERET=addr] ,keyword={addr | n | option} [,OBJECT={DATA | INDEX}]

{ACB | EXLST | RPL}=addr
This operand specifies whether you want to test an ACB, an EXLST, or an RPL and specifies its address.

In the standard and list forms of TESTCB, you can omit this operand if you are testing only the standard length of a control block or list (see "Length of a Control Block or List" under the keyword operand). With the execute form of TESTCB, you can change the address of the block or list to be tested, but not the type.

ERET=addr
This operand specifies the address of a user-written routine that VSAM gives control if, because of an error, it is unable to test for the condition you specified (return code in register 15 is not X'00'). When the ERET routine receives control, it should inspect the return code. If the return code is X'04', an error code will be set in register 0. The section "Return Codes for the GENCB, MODCB, SHOWCB, and TESTCB Macros" describes the return codes and error codes which can be set by TESTCB.

After completing its processing, the ERET rotine can terminate the job or pass control to a point in the processing program that it determines. It cannot return to VSAM.

keyword={addr | n | option}...
This operand specifies a field and a value. The contents of the field are compared with the value and the condition code is set. You can specify only one keyword at a time. There are three groups of operands that you can code with the TESTCB macro:

- The addresses, values, options, and names that you can code with the ACB, EXLST, RPL, and GENCB macros
- The length of a control block or list
- The attributes of an open file or index indicated by the access-method control block

If you code more than one operand, each of them must be compared equal to the corresponding value in the block or list for you to get an equal condition.

Operands of the ACB, EXLST, and RPL Macros The operands in this group are identical to those of the ACB, EXLST, and RPL macros, except that:

- You can code the operands in more ways, as shown in Appendix H.
- In relation to the ACB macro, you can test for return codes from the Open and Close routines (see "Opening and Closing Files") by coding ERROR=code (as any absolute expression, except for a self-defining character term).
- In relation to the EXLST macro, you can test whether an EXLST has an exit of a certain type by coding keyword=0.
- In relation to the EXLST macro, you can test whether an address in an EXLST is active or not active or is the address of the name of a routine to be loaded, without specifying the address (keyword=[{A | N}][,L]).
- In relation to the RPL macro, you can code the operand FDBK=code (as any absolute expression, except for a self-defining character term) to test for return codes from the request macros. (See *Return Codes for the Request Macros* under *Requesting Access to Files.*) You can code the operand RBA=n to test the relative byte address of the last record processed.

Length of a Control Block or List
You can code the operand EXLLEN=n, ACBLEN=n, or RPLLEN=n to test either the standard length of an EXLST, ACB, or RPL; or the

actual length of a particular ACB, RPL, or EXLST. You test for a standard length by omitting the {ACB | EXLST | RPL} operand and coding only one (or more) of these length operands and no other operands. You test the actual length of a block or list by specifying the {ACB | EXLST | RPL} operand and the corresponding length operand.

Attributes of an Open File or Index

After a file is opened, the ACB contains information that it does not contain before it is opened or after it is closed. Whether you are testing for the attributes of the data or the index of a key-sequenced file is determined by the OBJECT operand. You can test whether the file is open by coding: OFLAGS=OPEN. You can test the following fields:

Operand	**Meaning**
AVSPAC	Number of bytes of available space in the data or the index.
BUFNO	Number of buffers being used for the data or the index.
CINV	Size of a control interval in the data or the index.
FS	Percent of free control intervals in each data control area of a key-sequenced file.
KEYLEN	Full length of the key field in each logical record.
LRECL	Maximum length of a logical record or, for an index, the index control interval size minus seven bytes.
NCIS	Number of control-interval splits in the file.
NDELR	Number of data records deleted from a key-sequenced file.
NEXCP	Number of EXCP commands issued since the data or the index was opened.
NEXT	Number of logical extents, data spaces or portions of data spaces, of the data or of the index.
NINSR	Number of records inserted into the file.
NIXL	Number of levels in the index of a key-sequenced file.
NLOGR	Number of data records in the file.
NRETR	Number of data records retrieved from the file.
NSSS	Number of control area splits in a key-sequenced file.
NUPDR	Number of data records updated in the file.
RKP	Displacement of the key field from the beginning of a data record; the same value is displayed whether the object is index or data.
STMST	System time stamp; the time and day (in microseconds) when the data or index was last closed. Bits 52-63 of the fields are unused.

You can also test for these attributes:

Operand	**Meaning**
ATRB=ESDS	Entry-sequenced file
,KSDS	Key-sequenced file
,WCK	VSAM is verifying write operations
,SSWD	Sequence set of the index is adjacent to the file
,REPL	Index records are replicated
,MACRF	ACB options specified
,OPTCD	RPL options specified

OBJECT={DATA | INDEX}

This operand specifies, for the open ACB of a key-sequenced file, whether the field being tested is for the data or the index. KEYLEN and RKP will contain the same value if the data or the index is being tested. FS, NCIS, NDELR, NINSR, NIXL, NLOGR, NRETR, NSSS, and NUPDR will contain zero if the index is being tested.

Examples of the TESTCB Macro

The examples show how the TESTCB macro is used to test values in a VSAM control block.

In the first example, TESTCB is used to determine whether or not a file is open:

```
          TESTCB    ACB=(2),OFLAGS=OPEN,      (File open?)
                    ERET=TESTERR
          BE        OPEN                      (Yes)
          B         UNOPEN                    (No)
          .
          .
TESTERR   ....                (Routine executed if TESTCB unsuccessful)
```

In the second example, an EODAF exit routine is not supplied and TESTCB is used to determine whether the LERAD exit routine was entered because of an end-of-file condition or a processing error:

```
LOGERR    TESTCB    RPL=(4),FDBK=4,           (End-of-file?)
                    ERET=TESTERR
          BE        EODATA                    (YES, go to EOF routine)
          B         ERROR                     (No, go to error routine)
          .
          .
TESTERR   ......              (Routine executed if TESTCB unsuccessful)
```

Return Codes for the GENCB, MODCB, SHOWCB, and TESTCB Macros
When VSAM returns to your processing program after a GENCB, MODCB, SHOWCB, or TESTCB request, register 15 indicates the success or failure of the request. X'00' means the operation has completed successfully. X'04' means an error was detected; register 0 contains an error code which indicates the type of error:

Error Code	Applicable Macros	Meaning
X'01'	G M S T	The request you indicated is invalid.
X'02'	G M S T	You have not indicated an ACB, RPL, or EXLST.
X'03'	G M S T	You have indicated an invalid keyword.
X'04'	M S T	The block or list at the indicated address is not the type indicated.
X'05'	S T	The ACB is closed - it must be opened.
X'06'	S T	You have indicated a nonexistent index in the OBJECT operand
X'07'	M S	There is no entry for the exit you have specified in the EXLST.
X'08'	G	There id not enough virtual storage in your partitions to generate the requested blocks or lists.
X'09'	G S	Your work area is too small to generate the requested blocks or lists or to display the requested fields.
X'0A'	G M	You did not specify a new address for one of your EXLST operands when you specify the L subparameter, or you specified neither an address nor the subparameters A or N.
X'0B'	M	The RPL is active - it must be inactive.
X'0C'	M	The ACB is open - it must be closed.
X'0D'	M	There is no EXLST address indicated in the ACB.
X'0E'	G M T	You have specified inconsistent parameters.
X'0F'	G S	Your work area (WAREA parameter) does not begin on a fullword boundary.

X'08' in register 15 means that you have tried to use the execute form of a macro to change a nonexistent entry in the parameter list.

X'0C' in register 15 means that the request was not executed because an error was encountered while VSAM routines were being loaded.

List, Execute, and Generate Forms of the Control Block Macros
The list and execute forms of the control block manipulation macros (GENCB, MODCB, SHOWCB, and TESTCB) allow you to save virtual storage by using one parameter list for two or more

macros. You can also make your program reentrant; executable by more than one task at a time. The generate form of the macros enables you to make programs reentrant but it does not allow shared parameter lists.

List and Execute Forms The list form of GENCB, MODCB, SHOWCB, and TESTCB has the same parameters as the standard form; except that it includes the parameter MF={L | (L,addr[,label])}.

The parameter list of the macro is created inline when MF=L is coded. This version is not reentrant and register notation cannot be used for macro parameter addresses.

When MF=(L,addr[,label]) is coded, the parameter list of the macro is created in the area specified by "addr". This form is reentrant. You must supply the area by a GETVIS macro when your program is executed. You can determine the size of the parameter list by coding the third operand "label". VSAM equates label to the length of the list.

The execute form produces the executable code of the macros. The execute form is also identical to the standard form, except that it includes the operand MF=(E,addr), where "addr" points to the parameter list created by the list form of the macro. All of the other operands of the macro are optional and are coded only to change entries in the parameter list before the list is used. However, you cannot use the execute form to add or delete entries from the parameter list or to change the type of list.

Generate Form The generate form of the macros allows you to make your program reentrant, but it does not create shared parameter lists. The generate form is the same as the standard form, except that you code MF=(G,addr[,label]). The parameter list is created in an area pointed to by "addr". To ensure that the parameter list is reentrant, "addr" should be coded in register notation. You must obtain this area by a GETVIS macro when the program is executed. You can determine the size of the parameter list by coding the third operand "label". VSAM equates "label" to the length of the list.

Examples of the List, Execute, and Generate Forms The following examples show use of the list, execute, and generate forms of the control block manipulation macros.

In the first example, MODCB is used to place the length of a record in the RPL before the record is written. The list and execute forms are used so that only one parameter list is created even though the macro is issued several times. Thos list from is not reentrant.

```
          MODCB     MF=(E,LENMOD),RECLEN=(7) (Current length in reg.7)
          LTR       15,15                    (MODCB successful?)
          BNZ       MODERR                   (No go to error routine)
          PUT       RPL=(4)                  (Yes, write record)
          .
          .
          .
          MODCB     MF=(E,LENMOD)            (Length is 100 bytes)
          LTR       15,15                    (MODCB successful?)
          BNZ       MODERR                   (No, go to error routine)
          PUT       RPL=(4)                  (Yes, write record)
          .
          .
          .
LENMOD    MODCB     RPL=(4),RECLEN=100,MF=L  (List forms has default length - 100
bytes)
```

In the second example, the generate form is used to create an ACB. It is reentrant because both the ACB itself anf the parameter list of the GENCB macro are created in areas obtained through a GETVIS macro.

```
          LA        10,PARMLEN                 (Load length for GETVIS)
          GETVIS    ADDRESS=(8),LENGTH=(10)    (Get area for parm. list)
          LTR       15,15                      (GETVIS successful?)
          BNZ        VISERR                    (No, go to error routine)
          GENCB     BLK=ACB,MF=(G,(8),PARMLEN,
                    EXLST=(3),BUFND=4,BUFNI=3,
                    BUFSP=11064,DDNAME=VFILENM,
                    MACRF=(KEY,SEQ,DIR,OUT),
                    PASSWD=PASS
          LTR       15,15                      (GENCB successful?)
          BNZ       GENERR                     (No, go to error routine)
          LR        2,1                        (Yes, save ACB address)
          .
          .
          .
PASS      DC        FL'6',C'CHANGE'
```

OPENING AND CLOSING FILES

The OPEN macro connects a processing program to a file, so the program can gain access to data. The CLOSE macro disconnects the program and the file. The TCLOSE macro performs some of the functions of CLOSE but leaves the program and the file connected so processing can continue without reopening the file.

OPEN Macro

Name	Operation	Operand
[name]	OPEN	addr[,addr...]

This operand specifies up to 16 addresses of ACBs and DTFs that specify the files to be opened.

A return code is set in register 15 to indicate whether the ACBs were opened or closed successfully. ACBs should be coded together to ensure that the return code will apply to all of them. If, for example, you coded:

```
OPEN ACB1,DTF1,ACB2
```

the return code will apply to ACB2 only. If ACB2 opened successfully and ACB1 did not, the return code will still be X'00'. (The VSAM Open routine sets register 15 to zero when it receives control after a DTF has been opened.) To ensure that the return code applies to both ACBs, the macro should be coded in one of the following ways:

```
OPEN DTF1,ACB1,ACB2

OPEN ACB1,ACB2,DTF1
```

The OPEN macro calls the Open routine, which verifies that the processing program has authority to process the file. Open constructs VSAM control blocks and loads VSAM routines into your partition. (VSAM routines, unlike those of other access methods, are not link-edited with the processing program.) By examining the DLBL statement indicated by the DDNAME operand in the ACB macro and the volume information in the catalog, Open verifies that the necessary volumes have been mounted. If a key-sequenced file is being opened, VSAM issues an error code to warn you if the data has been updated separately from its index.

Open sets one of the following return codes in register 15. The return code will apply to all ACBs only if they are coded together (see example above):

Return Code	Meaning
X'00'	All ACBs have been opened successfully.
X'04'	All ACBs opened successfully, but one or more ACBs had a warning message.
X'08'	One or more ACBs were not opened successfully. The entries with errors are restored to their pre-Open status.

If register 15 contains X'04', an error code is set in one or more ACBs to indicate a warning message. All ACBs are open and, unless you prevent it, processing will continue on the file that the message applies to. You can use the ERROR keyword of a SHOWCB or TESTCB macro to examine the code.

Error Code	Meaning
X'6C'	The system time stamps of the data of a file and its index do not match; this indicates that either the data or the index has been updated separately and data integrity problems may result if the file is processed now.
X'74'	The file was not successfully closed the last time it was processed because (1) an error caused the job to terminate during CLOSE or before the CLOSE macro was issued or (2) the processing program did not issue a CLOSE macro. If records were added, deleted, or updated during previous processing, do not process the file now because of possible data integrity problems. Consult message 4n25I in *DOS/VS Messages*, GC33-5379. If records were only retrieved, the file will not have data integrity problems and can now be processed as intended.

If register 15 contains X'08', an error code is set in one or more ACBs. You use the ERROR keyword of the SHOWCB or TESTCB macro to examine the code.

Error Code	Meaning
X'00'	No error (set when register 15 contains X'00').
X'04'	This ACB is already open.
X'0E'	The symbolic unit in the DLBL statement is invalid.
X'0F'	No job information block (JIBs) are available from the supervisor.
X'11'	The address in an ASSGN statement for a VSAM module was set to IGN.
X'12'	The address in an ASSGN statement for a VSAM volume was set to UA.
X'22'	The volume serial numbers specified in the EXTENT statement do not match those specified in the catalog entry.
X'24'	More than 16 EXTENT statements were specified for the file.
X'32'	One or more VSAM processing modules cannot be loaded because the user's virtual partition is too small.
X'50'	1. Attempt made to mount two volumes on the same unit when direct or keyed processing specified in the ACB. 2. Operator unable to mount required volume.
X'68'	The time stamp of the volume on which the file is stored does not match the system time stamp in the file's catalog entry. The extent information in the catalog entry may not agree with the extent information in the VTOC.
X'6E'	Attempt made to open an empty file (no records in it) for input only (MACRF=IN in the ACB).
X'75'	The symbolic unit specified in the EXTENT statement is not a valid device type.
X'80'	The DLBL statement is missing or the filename in the DLBL does not match the ACB.
X'84'	A permanent I/O error occurred while VSAM was reading label information from the label information cylinder.
X'88'	Not enough virutual-storage space is available in your partition for work areas, control blocks, or buffers.
X'90'	A permanent I/O error occurred while VSAM was reading or writing a catalog entry.
X'94'	No entry was found in the catalog for this ACB (file not found).
X'98'	Security verification failed; the password specified in the ACB or supplied by the operator for a specific level of access does not match the password in the catalog for that level of access.
X'A0'	Keyed access was specified in the ACB (ACB macro or GENCB macro) but the file is entry-sequenced.
X'A1'	User buffers (MACRF=UBF) has been specified with keyed or addressed access; it can only be specified with control interval access.
X'A4'	A permanent I/O error occurred while VSAM was reading the volume label of the volume the file is on.
X'A8'	The file is not available because it is being (1) updated by (under the exclusive control of) another ACB; or (2) exported by access method.
X'B4'	The VSAM catalog is not connected to the system on logical unit SYSCAT.
X'FF'	The unexpected error occurred during catalog processing; rerun the job and call your IBM Programming Systems Representative if the problem persists.

CLOSE Macro

Name	Operation	Operand
[name]	CLOSE	addr[,addr...]

This operand specifies up to 16 addresses of ACBs and DTFs that specify the files to be closed.

A return code is set in register 15 to indicate whether the ACBs were closed or closed successfully. ACBs should be coded together to ensure that the return code will apply to all of them. If, for example, you coded:

```
CLOSE ACB1,DTF1,ACB2
```

the return code will apply to ACB2 only. If ACB2 closed successfully and ACB1 did not, the return code will still be X'00'. (The VSAM Close routine sets register 15 to zero when it receives control after a DTF has been closed.) To ensure that the return code applies to both ACBs, the macro should be coded in one of write the following ways:

```
CLOSE DTF1,ACB1,ACB2
CLOSE ACB1,ACB2,DTF1
```

The Close routine completes any I/O operations that are outstanding when a processing program issues a CLOSE macro for a file. It writes any output buffers that have not been stored.

Close updates the catalog entries of the file, including pointers to the end of the file and statistics on file processing (such as number of records inserted). If the file was being loaded and the SPEED option was specified (in the catalog), Close formats the last control area in the file to ensure that the entire file is accessible.

Close restores the ACB to the status that it had before the file was opened and frees the virtual storage that Open used to construct VSAM control blocks.

Close sets one of the following return codes in register 15. The return code will apply to all ACBs only if they are coded together (see example above):

Error Code	Meaning
X'00'	All ACBs were closed successfully.
X'04	One or more ACBs were not closed successfully.

If register 15 contains X'04', an error code is set in one or more ACBs. You use the ERROR keyword of the SHOWCB or TESTCB macros to examine the error code.

Error Code	Meaning
X'00'	No error (set when register 15 contains X'00').
X'04'	The ACB was already closed.
X'08'	One or more close routines could not be loaded because there was not enough virtual storage space, or the modules could not be found. Processing cannot continue.
X'88'	Not enough virtual-storage space was available in your partition for a work area for the close routine.
X'90'	A permanent I/O error occurred while VSAM was reading or writing a catalog entry.
X'B8'	A permanent I/O error or an internal error in a VSAM routine occurred while VSAM was completing I/O requests.
X'BC'	The ACB is being used by either a SHOWCB macro or a TESTCB macro.

TCLOSE Macro

Name	Operation	Operand
[name]	TCLOSE	addr[,addr...]

This operand specifies the addresses of up to 16 ACBs. You cannot specify the addresses of DTFs with TCLOSE.

A TCLOSE macro completes outstanding I/O operations and updates the catalog. Processing can continue without reopening the file. You use the TCLOSE macro to protect data while the file is being loaded or extended and the SPEED option was specified when the file was defined. When TCLOSE is issued, the Close routine formats the last control area in the file to ensure that all of the data that has been loaded is accessible.

TCLOSE sets one of the following return codes in register 15:

Return Code	Meaning
X'00'	All ACBs were closed successfully.
X'04'	One or more ACBs were not closed successfully.

If register 15 contains X'04', an error code is set in one or more ACBs. You use the ERROR keyword of the SHOWCB or TESTCB macros to examine the error code.

Error Code	Meaning
X'00'	No error (set when register 15 contains X'00').
X'04'	The ACB was already closed.
X'08'	One or more close routines could not be loaded because there was not enough virtual storage space, or the modules could not be found. Processing cannot continue.
X'88'	Not enough virtual-storage space was available in your partition for a work area for the Close routine.
X'90'	A permanent I/O error occurred while VSAM was reading or writing a catalog entry.
X'B8'	A permanent I/O error or an internal error in a VSAM routine occurred while VSAM was completing I/O requests.
X'BC'	The ACB is being used by either a SHOWCB macro or a TESTCB macro.

REQUESTING ACCESS TO FILES

The macros GET, PUT, POINT, and ERASE initiate all requests for access to data records.

When your program issues a request macro, its processing does not continue until VSAM completes the request. At that time, VSAM sets a return code in register 15. If end-of-file is reached or an error or other special condition occurs during the request, VSAM sets a code containing additional information in the feedback field of the RPL, and takes any required exit. The return codes and codes set in the feedback field of the RPL are described later in this section.

GET Macro

Name	Operation	Operand
[name]	GET	RPL=addr

The operand contains the address of the RPL (or first RPL in a chain of them) that specifies the GET request. When you issue a GET macro, register 13 must contain the address of a 72-byte save area that you are providing. This macro retrieves the next record in key sequence with RPL operand OPTCD=(KEY,SEQ), and the next record in entry sequence with OPTCD=(ADR,SEQ). It retrieves the record specified by the key in the search-argument field with OPTCD=(KEY,SKP) or OPTCD=(KEY,DIR), and by the RBA in the search-argument field with OPTCD=(ADR,DIR). With skip sequential retrieval, each key that you specify must be greater in collating sequence than the key of the previous record retrieved.

Get retrieves the next control interval with OPTCD=(CNV,SEQ) and the control interval specified by the RBA in the search-argument field with OPTCD=(CNV,DIR).

You must issue a GET with OPTCD=UPD to update (PUT with OPTCD=UPD) or to delete (ERASE) a record. You can have the record moved to your work area (OPTCD=MVE) or you can have VSAM leave the record in its I/O buffer and pass you the address of the record (OPTCD=LOC). The AREA parameter points to your work area or to a field in which VSAM will place a record address.

You can also keep VSAM positioned for subsequent sequential or skip sequential processing when you issue a direct GET request with OPTCD=(DIR,NSP) or OPTCD=(DIR,UPD). With OPTCD=(DIR,UPD) however, positioning is cancelled when you issue a PUT for update or an ERASE following the GET for update.

PUT Macro

Name	Operation	Operand
[name]	PUT	RPL=addr

The operand contains the address of the RPL (or the first RPL in a chain of them) that specifies the PUT request. When you issue a PUT macro, register 13 must contain the address of a 72-byte save area that you are providing. This macro stores a new record in key sequence with OPTCD=(KEY,SKP,NUP), OPTCD=(KEY,DIR,NUP), OPTCD=(KEY,SEQ,NUP), or OPTCD=(KEY,DIR,NSP). When OPTCD=(KEY,DIR,NSP), VSAM is kept positioned at the next record in key sequence for subsequent sequential processing.

PUT stores a new record at the end of an entry-sequenced file with OPTCD=ADR (you cannot store a new record in a key-sequenced file with addressed access). With skip sequential storage, OPTCD=(KEY,SKP), the key of each record that you store must be greater in collating sequence than the key of the previous record stored. With control-interval access, OPTCD=(CNV,NUP), PUT stores a new control interval at the end of an entry-sequenced file.

When loading or extending a file with the PUT macro, you must specify sequential or skip sequential processing (OPTCD=SEQ or OPTCD=SKP).

To store a changed record or control interval, you must have previously retrieved it with OPTCD=UPD and also specify OPTCD=UPD when you issue a PUT. You cannot change the key of a record in a key-sequenced file. With OPTCD=ADR, you cannot change the length of a record in either a key-sequenced or an entry-sequenced file.

The record to be added or updated with a PUT macro must be in your work area (OPTCD=MVE); you cannot use OPTCD=LOC with the PUT macro. The AREA parameter of the ACB points to your work area.

POINT Macro

Name	Operation	Operand
[name]	POINT	RPL=addr

The operand contains the address of the RPL (or the first RPL in a chain of them) that specifies the POINT request. When you issue a POINT macro, register 13 must contain the address of a 72-byte save area that you are providing. This macro positions VSAM at the record whose key (with OPTCD=KEY) you specify in the search-argument field. It can be used to position either forward or backward in the file for subsequent sequential or skip sequential processing.

POINT positions VSAM at the record or control interval whose RBA (with OPTCD=ADR or OPTCD=CNV) you specify in the search-argument field. It can be used to position either forward or backward in the file for subsequent sequential processing.

VSAM can also be positioned for sequential processing by either a direct GET or a direct PUT as described in the preceeding sections on the GET and PUT macros.

ERASE Macro

Name	Operation	Operand
[name]	ERASE	RPL=addr

The operand contains the address of the RPL (or the first RPL in a chain of them) that specifies the ERASE request. When you issue an ERASE macro, register 13 must contain the address of a 72-byte save area that you are providing.

This macro deletes the record previously retrieved for update (with the GET macro, OPTCD=UPD). You can delete records in a key-sequenced file by keyed or addressed access, but you cannot delete records in an entry-sequenced file. You cannot delete control intervals (OPTCD=CNV).

ENDREQ Macro

Name	Operation	Operand
[name]	ENDREQ	RPL=addr

The operand contains the address of the RPL (or first RPL in a chain of them) that specifies the request to be terminated. When you issue an ENDREQ macro, register 13 must contain the address of a 72-byte save area that you are providing.

This macro caused VSAM to end a request. If an I/O operation was started, it will be allowed to complete. Also, I/O operations required to maintain the integrity of the file will be performed.

If the request involves a chain of RPLs, all records specified by the request may not be processed. For example, two RPLs are chained in a PUT request to add two new records to the file and an ENDREQ is issued after VSAM started the I/O to add the first record. That I/O operation will be completed and, if it causes a control-interval split, subsequent I/O operations will be performed to complete the split and update the index. However, VSAM will then return control to the processing program without adding the second record.

The ENDREQ macro also causes VSAM to cancel the position in the file established for that request.

Return Codes for the Request Macros

When VSAM returns to your processing program, a return code in register 15 indicates what happened. If an error occurred, additional information on the request will be in the RPL. Your processing program can examine the feedback field of the RPL with the FDBK keyword of the SHOWCB or TESTCB macro: register 1 contains the address of the RPL that defines the request that caused the error.

Control is returned to the instruction following your action macro when (1) the request is completed, (2) the request was not accepted because another request was using the RPL, or (3) an error occurred and you did not have an active exit routine. If you do have an active exit routine and it returns control to VSAM after processing the error, VSAM then returns control to the instruction following the action macro.

When you gain control after a request, register 15 will contain one of the following return codes:

X'00'	Request completed successully; the RPL might contain additional (non-error) information about the request.
X'04'	This request was not accepted because a request from another task is active on the same RPL; no additional information in the RPL.
X'08'	Logical error; the error code in the RPL identifies the specific error.
X'0C'	Uncorrectable I/O error; the error code in the RPL identifies the specific error.

If the request completed with a logical error (X'08'), your LERAD exit routine is entered if you specified the LERAD exit in the EXLST and it is active. When you reach end-of-file, error code X'04' is set. Your EODAD exit routine will be entered. If you have no EODAD exit routine or it is inactive, your LERAD exit routine is entered.

If the request completed with an I/O (physical) error (X'0C'), your SYNAD exit routine is entered if you specified the SYNAD exit in the EXLST and it is active. The return code is not set in register 15 when the exit routine is entered. VSAM sets the return code in register 15 and returns control to the instruction following the action macro after you complete the exit routine.

The feedback field in the RPL (FDBK parameter in SHOWCB and TESTCB) is a fullword with the following format:

000000xx

where
xx is an error code which describes the error or, if the return code is zero, additional information about the request.

Return Code	Error Code	
X'00'	Request completed successfully.	
	X'00'	Request completed successfully; no additional information.
	X'04'	Request completed successfully; end-of-volume calling during request.
X'04'	Request not accepted; a request from another task is active on this RPL.	
X'08'	Logical error.	
	X'04'	End-of-file encountered, EODAD exit routine entered if one supplied.
	X'08'	Duplicate record.
	X'0C'	Record out sequence (record might be a duplicate).
	X'10'	No record found.
	X'14'	Record already held in exclusive control by another requester.
	X'18'	Record is on a volume which is not mounted.
	X'1C'	All extents of file are full and VSAM cannot sub-allocate new extents because there is no available data space on (1) a volume which already contains extents of the file or (2) a volume that is candidate for extension of the file.
	X'20'	Invalid RBA specified.
	X'24'	The key of the record to be inserted does not fall in an existing key range in the file.
	X'28'	Insufficient virtual storage available to finish request.
	X'2C'	The work area you have supplied (AREA parameter) is not large enough.
	X'30'	One or more VSAM processing modules cannot be loaded because the user's virtual partition is not large enough.
	X'34'	An internal error occurred in a VSAM routine. Save the program listing, console log, and dump, and contact your IBM Programming Support Representative.

Return Code	Error Code	
X'08'	X'38'	The VSAM catalog was accessed during processing of a request. An I/O error or an internal processing error occurred during catalog access. Use the LISTCAT command of Access Method Services to examine the catalog entry for this cluster. You can then attempt to rerun the job if the catalog entry is valid. If the rerun is not successful, save the program listing, console log, LISTCAT listing, and dump and contact your IBM Programming Support Representative.
	X'40'	As many requests are active as the number specified in the STRNO operand of the ACB; therefore, another request cannot be started.
	X'44'	Type of accessing for the request not in ACB when file was opened: • Keyed access not specified • Output not specified • CNV processing not specified
	X'48'	Keyed access requested for an entry-sequenced file.
	X'4C'	Addressed or control-interval insertion requested for a key-sequenced file.
	X'50'	ERASE macro specified for an entry-sequenced file or for control-interval processing.
	X'54'	Locate mode specified for PUT macro or for user buffer processing.
	X'58'	VSAM not positioned (POINT) for sequential GET or you changed to keyed access after establishing position by addressed access.
	X'5C'	PUT update or ERASE issued without a preceding GET update.
	X'60'	Attempt to change key when updating record.
	X'64'	Attempt to change record length during update with addressed access.
	X'68'	Invalid or conflicting RPL options.
	X'6C'	Record length specified that is larger than allowed maximum, equal or zero, or smaller than key length plus relative key position.
	X'70'	Length of generic key is too large or equal to zero.
	X'74'	Request other than sequential or skip sequential PUT to insert records specified during loading of the file.
X'0C'	Physical (I/O) Errors	
	X'04'	I/O error while reading data.
	X'08'	I/O error while reading index set of index.
	X'0C'	I/O error while reading sequence set of index.
	X'10'	I/O error while writing data.
	X'14'	I/O error while writing index set of index.
	X'18'	I/O error while writing sequence set of index.

PART 6

PHYSICAL IOCS

CONCEPTS OF PHYSICAL IOCS

Records can be transferred to or from an input/output device by issuing physical IOCS macros. These macros relate directly to the physical IOCS routines and are distinct from the logical IOCS macros described in the SAM, DAM, ISAM, and VSAM chapters of this book. For more information on the distinction between physical and logical IOCS see the *DOS/VS Data Management Guide*, GC33-5372.

When using physical IOCS macros, you must provide for such functions as the blocking or deblocking of 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 CCWs. You 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 perform the following:

- Starting I/O operations
- I/O interrupt handling
- Channel scheduling
- Device error handling

Thus physical IOCS macros provide you with the capability of obtaining data and performing nondata operations with I/O devices using exactly the CCWs you request. For example, if your program handles only physical records, you do not need the logical IOCS routines for blocking and deblocking logical records.

Three macros are available for direct communication with physical IOCS: CCB (command control block), EXCP (execute channel program), and WAIT. Whenever physical IOCS macros are used, you must construct the CCWs for input/output operations. Use the assembler instruction CCW statement to do this. A detailed technical description of the CCW can be found in *IBM System/370 Principles of Operation*, GA22-7000. Considerations for CCW programming are given in the *DOS/VS Data Management Guide*, GC33-5372.

Macros normally used with files processed by logical IOCS are necessary in addition to the macros provided by PIOCS when standard DASD or magnetic tape labels are processed, or when DASD file protect is present. The DTFPH, OPEN or OPENR, CLOSE or CLOSER, LBRET, FEOV, and SEOV macros can be used in this processing. The OPEN and the DTFPH macros are also necessary when a disk is used for a checkpoint file. PIOCS considerations for alternate tape switching and bypassing embedded checkpoint records on tape are given in the *DOS/VS Data Management Guide*, GC33-5372.

The CCB, EXCP, and WAIT macros used for direct communication with PIOCS; the PIOCS DTFPH declarative macro; and the OPEN or OPENR, LBRET, FEOV, SEOV, and CLOSE or CLOSER imperative macros used with PIOCS, are described in the remainder of this chapter.

Physical IOCS Macros

CCB Macro

Name	Op	Operand
blockname	CCB	SYSnnn,command-list-name [,X'nnnn'][,sense address]

A CCB (command control block) macro must be specified in your program for each I/O device controlled by physical IOCS macros. The first 16 bytes of all the generated DTF tables--except DTFOR and DTFSR (optical reader)--contain the CCB. For DTFOR and DTFSR (optical reader), the CCB begins at filename+24 which includes the DTFPH. The CCB (see Figure 6-1) is necessary to communicate information to physical IOCS so that it can perform desired operations (for example, notifying your program of printer channel 9). The CCB also receives status information after an operation and makes this available to your program.

blockname: The CCB macro must be given a symbolic name (blockname). This name can be used as the operand in the EXCP and WAIT macros which refer to the CCB.

SYSnnn: This operand specifies the symbolic unit for the actual I/O unit with which this CCB is associated. A list of symbolic units applying to CCB can be found in the *Symbolic Unit Addresses* section of *The Macro System* chapter. The actual I/O unit can be assigned to the symbolic unit by an ASSGN job control statement.

command-list-name: This operand 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 CCW.

X'nnnn': A hexadecimal value used to set the CCB user option bits. Column 5 of Figure 6-2 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')

The macro can set any of the bits in bytes 2 or 3 on. Normally, however, you need not be concerned with setting the remaining bits on.

sense address: This operand, when supplied, indicates user error recovery (see Figure 6-2 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 you supply 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 you desire.

If the user has his own user error routine (byte 2, bit 7 of the CCB is on, or X'nnnn' in the CCB macro = X'0100') but has not specified the parameter 'sense address' in the CCB macro, the sense information is cleared by the supervisor in order to prevent deadlocks in the control unit. If the user then issues an EXCP with the CCW address for SENSE from the error routine, the information has already been destroyed.

CCB Format
From the above specifications, the macro sets up a 16-byte or 24-byte field (Figure 6-1) 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, your program can determine the length of the record that was transferred. This residual count is set to zero for negative values.
2-3	These next two bytes are for transmission of information between physical IOCS and your program. Your program can test any bit in bytes 2 and 3, using the mask given in the last column of Figure 6-2. More than one bit can be tested by the hexadecimal sum of the test values. All bits are set to 0 when your program is assembled unless the X'nnnn' operand is specified. If this operand is specified, it is assembled into these two bytes. You 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 to 1 by your 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 6-2 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	This byte indicates the type of CCB.
7	This byte is 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	You must not modify this byte.

Figure 6-1 Command Control Block (CCB)

	Count	Transmission Information	CSW Status Bits		Type Code		Reserved for Logical IOCS	CCW Address	Reserved for Physical IOCS	CCW Address in CSW	Optional Sense CCW
Bytes	0 1	2 3	4 5		6 7		8	9 11	12	13 15	16 23
Used for	Residual Count	Transmitting Information Between Physical IOCS and Problem Program For details see Figure 6-2	(Note 1) Byte 4	Byte 5	Byte 6	Byte 7	Buffer Offset: ASCII Input Tapes X'00'-X'63' ASCII Output Tapes Fixed X'00' Variable: X'00' or X'04' Undefined: X'00'	Virtual or real address of CCW associated with this CCB depending on byte 6: Real address if byte 6 = X'2u', X'6u', or X'8u'; Virtual address if byte 6 = X'0u', or X'4u'	X'80'-CCB being used by ERP X'40'-Channel Appendage Routine Present X'20'- Sense Information Desired X'10'- Message Writer X'08'- EU Tape Error X'04'- OLTEP Appendage Available X'02'- Tape ERP Read Opposite Recovery X'01'- Seek Separation	Virtual address of CCW pointed to the CSW at Channel End; (if byte 6 = X'8u', it is the real address) or address of the Channel End Appendage Routine	8 Bytes Appended to the CCB when Sense Information is Desired
			Bits 0 Attention 1 Status modifier 2 Control unit end 3 Busy 4 Channel end 5 Device end 6 Unit check 7 Unit exception	Bits 0 Program-controlled interruption 1 Incorrect length 2 Program check 3 Protection check 4 Channel data check 5 Channel control check 6 Interface control check 7 Chaining check	X'0u' Original CCB X'2u' Translated CCB X'4u' BTAM request original CCB X'6u' BTAM request translated CCB X'8u' User-translated CCB in virtual partition Note: Any one of the above incremented by X'10' (bit 3 on) indicates automatic switching to the beginning of the next cylinder at End of Cylinder condition. u: 0 = the address in byte 7 refers to a System Logical Unit 1 = the address in byte 7 refers to a Programmer Logical Unit	Hexadecimal Representation of SYSnnn SYSRDR = 00 SYSIPT = 01 SYSPCH = 02 SYSLST = 03 SYSLOG = 04 SYSLNK = 05 SYSRES = 06 SYSSLB = 07 SYSRLB = 08 SYSUSE = 09 SYSREC = 0A SYSCLB = 0B SYSVIS = 0C SYSCAT = 0D SYS000 = 00 SYS001 = 01 SYS002 = 02 : SYSmax (Note 2)					

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. SYSmax=255- (number of partitions* 14).

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. 3211 UCSB Parity Check (line complete)	/* or /& on SYSRDR or SYSIPT. Byte 4, Unit exception Bit is also on. Yes	 No		X'40'
	2 Irrecoverable 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 Irrecoverable 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. 1017/1018 data checks. Retrun any DASD data checks.	Operator Options: Ignore, Retry, or Cancel. Ignore or Cancel. Return to user.	Operator Option: Retry or Cancel. Cancel.	X'0800'	X'08'
	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, 2560 data check; 2520 or 2540 punch equipment check;2560 or 5425 equipment check; 3504, 3505, or 3525 permanent errors; DASD read or read 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; when 1018 or 2560 data check; when 2560 or 5425 equipment check; when 3504, 3505, or 3525 permanent error (byte 3, bit 3 is also on).	Operator Option: Ignore or Cancel for tapes, paper tape punch (1018), card punches other than 2560 and 5425. Retry or Cancel for DASD, 2560, or 5425.	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. Data check - 1287 or 1288. MICR - SCU not operational. 3211 Print Check (equipment check).	Yes-Byte 3, bit 3 is off; Byte 2, bit 2 is on. Yes Yes Yes	No No No No		X'80'
	1 DASD Track overrun. 1017 broken tape. Keyboard correction 1287 in Journal Tape Mode. 3211 print quality error (equipment check) MICR intervention required.	Yes Yes Yes Yes Yes	No No No No No		X'40'
	2 End of DASD Cylinder. Hopper Empty 1287/1288 Document Mode. MICR - 1255/1259/1270/1275/1419, disengage. - 1275/1419D, I/O error in external interrupt routine. 3211 line position error.[5]	Yes Yes Document feeding stopped. Channel data check or Busout check. Yes	No No No No		X'20'

Figure 6-2 **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); 2520 or 2540 punch equipment check; 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, 2560 data check.	Yes	No		
	1287, 1288 equipment check.	Yes	No		
	2560, 5425 equipment check.	Byte 2, bit 6 is also on.	No		
	3504, 3505, 3525 permanent errors.	Byte 2, bit 6 is also on.	No		
	3211 data check (print check).	Yes	No		
	4 Questionable Condition.	Card: Unusual command sequence (2540). DASD: No record found.			X'08'
	Nonrecovery	1287/1288: Document jam or torn tape.			
	UCSB parity check (command retry).	Yes	Yes		
	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 or 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] You 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, you 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 you 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 6-2 Conditions indicated by CCB bytes 2 and 3 (part 2 of 2)

Bytes	Contents
13-15	These bytes contain the virtual address of the CCW pointed to by the CSW at channel-end interrupt for this I/O operation.
	Note: Bytes 13-15 contain the address of the channel appendage routine when 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 your program.

EXCP Macro

Name	Operation	Operand
[name]	EXCP	{blockname[,REAL] / (1)}

The EXCP (execute channel program) macro requests physical IOCS to start an input/output operation for a particular I/O device.

Physical IOCS determines the device from the CCB specified by blockname, and places the 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 your 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.

blockname	is the virtual address of the CCB established for the device. It can be given as a symbol or in register notation.
REAL	indicates that the addresses in the CCWs and the address in the CCB pointing to the first CCW have already been translated into real addresses. The system's CCW translation routine will be skipped. The CCB, the channel program, and the I/O areas must be PFIXed prior to issuing EXCP...,REAL. **Note:** For a program running in real mode, the parameter REAL is ignored. If the supervisor is generated without the option ECPREAL=YES in the FOPT supervisor generation macro, and REAL is specified, the issuing task is canceled.

WAIT Macro

Name	Operation	Operand
[name]	WAIT	{blockname (1)}

This macro is issued whenever your program requires that an I/O operation (started by an EXCP macro) be completed before execution of the program continues. For example, transferring data (a physical record) to virtual storage must be completed before data can be added or moved to another area of virtual storage, or otherwise processed. When WAIT 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, processing 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 (specified as a symbol or in register notation) of the CCB established for the I/O device is the only operand required. This is also the same name as that specified in the EXCP macro for the device.

DTFPH Macro

When physical IOCS macros (EXCP, WAIT, etc.) are used in a program, DASD, diskette, 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 disk; the following operands can be specified:

Operand	Optional	Required
CCWADDR=name	X	
DEVADDR=SYSnnn	X	
DEVICE=2311, 2314, 3330, 3340, 3540		X
LABADDR=name	X	
MOUNTED=SINGLE		X
TYPEFLE=OUTPUT		X

Enter the symbolic name of the file (filename) in the name field and DTFPH in the operation field. The detail entries follow the DTFPH header card in any order. Figure 6-3 lists the keyword operands contained in the operand field.

ASCII=YES
This operand is required to process ASCII tape files. If this operand is omitted, EBCDIC processing is assumed.

CCWADDR=name
This operand allows you 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 CCW.

If this operand is omitted, the location counter value of the CCB-CCW table address constant is substituted for the CCW address.

M	TYPFLE=xxxxxx	(INPUT or OUTPUT). Specifies type of file.
O	ASCII=YES	ASCII file processing is required.
O	CCWADDR=xxxxxxxx	If CCB is generated by DTFPH is to be used.
O	DEVICE=xxxx	(TAPE, 2311, 2314, 2321, 3330, 3340, 3540). If omitted, TAPE is assumed.
O	DEVADDR=SYSxxx	Symbolic unit required only when not provided on an EXTENT statement.
O	HDRINFO=YES	Print header label information.
O	LABADDR=xxxxxxxx	Routine to check or built user standard labels.
O	MOUNTED=xxxxxx	(ALL or SINGLE). Required for DASD files only; for diskette files, specify SINGLE.
O	XTNTXIT=xxxxxxxx	If EXTENT statements are to be processed. DASD only.

M=Mandatory; O=Optional

Figure 6-3 DTFPH macro

DEVICE={<u>TAPE</u> | 2311 | 2314 | 2321 | 3330 | 3340 | 3540}

If the file is contained on DASD or diskette, enter the proper identification. Specify 2314 for 2319 and 3330 for 3333. TAPE applies to any 2400/3400-series tape unit, and is the only valid entry in this operand for ASCII files.

DEVADDR=SYSxxx

This operand must specify the symbolic unit (SYSxxx) associated with the file if a symbolic unit is not provided via an EXTENT job control statement. If a 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 ASSGN job control 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* in *The Macro System* chapter. 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 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 file with standard labels 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

You may require one or more DASD or tape labels in addition to the standard file labels. If so, you must include your own routine to check (on input) or build (on output) your label(s). Specify the symbolic name of your routine in this operand. IOCS branches to this routine after the standard label is processed.

LABADDR may be included to specify a routine for your header or trailer labels as follows:

- DASD input or output: header labels only.
- Tape input or output: header and trailer labels.

Thus, if LABADDR is specified, your header labels can be processed for an input/output DASD or tape file, and your trailer labels can be built for a tape output file. Physical IOCS reads input labels and makes them available to you 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 the *Label Processing* chapter.

If physical IOCS macros are used for a tape file, an OPEN must be issued for the new volume. This causes IOCS to check the HDR1 label and provides for your checking of user standard labels, if any.

When physical IOCS macros are used and DTFPH is specified for standard tape label processing, FEOV must not be issued for an input file.

MOUNTED={ALL | SINGLE}
This operand must be included to specify how many extents (areas) of the file are available for processing when the file is initially opened. This operand must not be specified for tape.

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 your control statements. Only one OPEN or OPENR is required for the file. ALL should be specified whenever you plan to process records in a manner similar to the direct access method. In any case, you must supply a LBLTYP statement.

After an OPEN or OPENR is performed, you 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 you 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 you plan to process records in sequential order. IOCS checks the labels on the first pack and makes the first extent specified by your control cards available for processing. You must keep track of the extents and issue a subsequent OPEN or OPENR whenever another extent is required for processing. You will find the information in the DTFPH table helpful in keeping track of the extents. The DTFPH table contains:

Bytes	Contents
0-15	CCB (symbolic unit has been initialized in the CCB).
54-57	Extent upper limits (cchh).
58-59	Seek address (bb-bin or cell number). For a one-celled device such as disk it must be zero.
60-63	Extent lower limit (cchh).

On each OPEN or OPENR after the first, IOCS makes available the next extent specified by the control cards. When you issue a CLOSE or OPENR for an output file, the volume on which you are currently writing records is indicated, in the file label, as the last volume for the file.

TYPEFLE={INPUT | OUTPUT}
This operand must be included to specify the type of file: input or output.

XTNTXIT=name
This entry is included if you want to process label extent information. It specifies the symbolic name of your extent routine. The DTFPH MOUNTED=ALL operand must also be specified for the file.

Whenever XTNTXIT is included, IOCS branches to your routine during the initial OPEN for the file. It branches after each specified extent is completely checked and after conflicts, if any, have been resolved.

Upon entry to your routine, IOCS stores the address (in register 1) of a 14-byte area from which you can retrieve label extent information (in binary form). This area contains:

Bytes	Contents	
0	Extent type code:	
	00	Next three fields do not indicate any extent.
	01	The extent containing your data record.
	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.
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 6-1).	
12	Old bin (cell) number. For a one-celled device such as disk, byte 12 contains zero.	
13	Present bin number of the extent (b2).	

Return to IOCS by using the LBRET macro.

OPEN and OPENR Macros

Op	Operand
for self-relocating programs	
OPENR	$\left\{\begin{matrix}\text{filename1}\\ \text{(r1)}\end{matrix}\right\} \left[, \left\{\begin{matrix}\text{filename2}\\ \text{(r2)}\end{matrix}\right\} \ldots, \left\{\begin{matrix}\text{filenamen}\\ \text{(rn)}\end{matrix}\right\}\right]$
for programs that are not self-relocating	
OPEN	$\left\{\begin{matrix}\text{filename1}\\ \text{(r1)}\end{matrix}\right\} \left[, \left\{\begin{matrix}\text{filename2}\\ \text{(r2)}\end{matrix}\right\} \ldots, \left\{\begin{matrix}\text{filenamen}\\ \text{(rn)}\end{matrix}\right\}\right]$

The OPENR or OPEN macro activates files processed with the DTFPH macro. These macros associate the logical file declared in your program with a specific physical file on a DASD. The association by OPENR or OPEN of your program's logical file with a specific physical file remains in effect throughout your processing of the file until you issue a CLOSE or CLOSER macro.

When OPENR is specified, the symbolic address constants generated from the parameter list are self-relocating. When OPEN is specified, the symbolic address constants are not self-relocating.

To write the most efficient code in a multiprogramming environment it is recommended that OPENR be used.

Self-relocating programs using LIOCS must use OPENR to activate all files, including console files. In addition to activating files for processing, OPENR relocates all address constants within the DTF tables (zero constants are relocated only when they constitute the module address). If symbolic notation is used, you must establish addressability through a base register.

If OPEN or OPENR attempts to activate a logical IOCS file (DTF) whose device is unassigned, the job is terminated. If the device is assigned IGN, OPEN or OPENR does not activate the file and turns DTF byte 16, bit 2 on, to indicate the file is not activated.

Enter the symbolic name of the file (DTF filename) 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, you can load the address of the DTF filename into a register and specify the register using ordinary register notation. The high-order 8 bits of this register must contain zeros. For OPENR, the address of filename may be preloaded into any of the registers 2-15. For OPEN, the address of filename may be preloaded into register 0 or any of the registers 2-15.

Note: If you use register notation, we recommend that you follow the standard practice of using only registers 2-12.

Whenever an input/output DASD or magnetic tape file is opened and you plan to process user-standard labels (UHL or UTL), or nonstandard tape labels, you 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 OPENR, or by issuing a separate OPEN or OPENR preceding the OPEN or OPENR 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.

Single Volume Mounted--Output
When processing output files with physical IOCS, OPEN or OPENR is used only if you want to build

standard labels. When the first OPEN or OPENR for the volume is issued, OPEN or OPENR checks the standard VOL1 label and the extents specified in the EXTENT job control statements 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 long.
3. Only type 1 and type 8 extents are valid.

OPEN or OPENR 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. After this check, OPEN or OPENR creates the standard label(s) for the file and writes the label(s) in the VTOC.

If you wish to create your own user standard header labels (UHL) for the file, you must include the LABADDR operand in the DTF. OPEN or OPENR reserves the first track of the first extent for these labels and gives control to your label routine. After this, the first extent of the file can be used. Each time you determine that all processing for an extent is completed, issue another OPEN or OPENR for the file to make the next extent available. When the last extent on the last volume of the file is processed, OPEN or OPENR 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 you.

Single Volume Mounted--Input
When processing input files with physical IOCS, OPEN or OPENR is used only if you want to check standard labels.

When the mounted volume is opened for the first time, OPEN or OPENR checks the extents specified in the extent cards (for example, checks that the extent limit address for the device being opened is valid). OPEN or OPENR 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 for use.

If LABADDR is specified, OPEN or OPENR makes the user standard header labels (UHL) available to you one at a time for checking. Then, OPEN or OPENR makes the first extent available for processing.

Each time you determine that all processing for an extent is completed, issue another OPEN or OPENR for the file to make the next extent available. If another extent is not available, OPEN or OPENR stores the character F (for EOF) in byte 31 of the DTFPH table. You can determine the end of file by addressing and checking the byte at filename+30.

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 or OPENR is used only if standard labels are checked or written.

For each volume, OPEN or OPENR checks the standard VOL1 label and checks the extents specified in the EXTENT job control statements:

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.
4. For 3340, all data modules must be of the same type.

OPEN or OPENR 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 or OPENR creates the standard label(s) for the file and writes the label(s) in the VTOC.

If you wish to create your own user standard header labels for the file, include the LABADDR operand in the DTF. OPEN or OPENR reserves the first track of the first extent for these labels and gives control to your label routine.

If the XTNTXIT operand is specified, OPEN or OPENR stores the address of a 14-byte extent information area in register 1. (See the DTFPH XTNTXIT operand, above, for the format of this area.) Then, OPEN or OPENR gives control to your extent routine. You can save this information for later use in specifying record addresses. If your DASD file is file protected, you cannot write on any extents while in the XTNTXIT routine. When checking is complete, return control to OPEN or OPENR by issuing the LBRET 2 macro which opens the next volume. After all volumes are opened, the file is ready for processing.

All Volumes Mounted--Input
When all volumes containing the input file are on-line and ready at the same time, each volume is opened one at a time before any processing is done. OPEN(R) is used only when standard labels are to be processed. For each volume, OPEN(R) checks the extents specified in the EXTENT job control statements, and checks the standard VOL1 label on track 0 and the file label(s) in the VTOC. If LABADDR is specified in the DTF, OPEN or OPENR makes the user standard labels available, one at a time, for checking.

If XTNTXIT is specified in the DTF, OPEN or OPENR stores the address of a 14-byte extent information area into register 1. (See the DTFPH XTNTXIT operand, above, for the format of this area.) Then OPEN(R) gives control to your extent routine. For example, you can save this area and use the information later on for specifying record addresses. If the DASD file is file protected, you cannot write on any extents while in the XTNTXIT routine.

Diskette Volumes--Output
When processing output files on diskettes with physical IOCS, OPEN or OPENR is used to build standard labels. When OPEN or OPENR is issued for the first volume, it checks the VTOC on the diskette, and

- ensures that the file to be created does not have the same name as an existing unexpired file,
- ensures there is at least one track available to be allocated, and
- allocates space for the file, starting at the track following the last unexpired or write-protected file on the diskette.

After this check, OPEN or OPENR creates the format-1 label for the file and writes the label in the VTOC. Each time you determine that all processing for an extent is complete, you must feed to make the next diskette available and then issue another OPEN or OPENR for the file, to make the next extent available. CLOSE or CLOSER will automatically cause the last volume to be fed out. If the last extent of the file is completely processed before a CLOSE(R) is issued, OPEN or OPENR assumes an error condition and the job is canceled.

Diskette Volumes--Input
When processing input files on diskettes with physical IOCS, OPEN or OPENR is used to check standard labels.

When the first volume is opened, OPEN or OPENR checks the VTOC on the diskette and determines the extent limits of the file from the file label.

After the label is checked, OPEN or OPENR makes the first extent available for processing. Each time you determine that all processing for a diskette is complete, you must feed to make the next diskette available, and then issue another OPEN or OPENR for the file, to make the next extent available. If another extent is not available, OPEN or OPENR stores the character F (for EOF) in byte 31 of the DTFPH table. You can determine the end of file by addressing and checking the byte at filename +30.

For a programmer logical unit, the last diskette will always be fed out; for a system logical unit, the last diskette will not be fed out.

LBRET Macro

Name	Operation	Operand
[name]	LBRET	{1 | 2 | 3}

The LBRET macro is issued in your subroutines when processing is completed and you wish 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 the *Label Processing* chapter).

Checking User Standard DASD Labels: IOCS passes labels to you one at a time until the maximum allowable number is read and updated, or until you signify you want no more. Use LBRET 3 in your label routine if you want IOCS to update (rewrite) the label read and pass you the next label. Use LBRET 2 if you want IOCS to read and pass you 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 you want to eliminate the checking of one or more remaining labels, use LBRET 1.

Writing User Standard DASD Labels: Build the labels one at a time and use LBRET to return to IOCS to write the labels. Use LBRET 2 if you wish 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. Use LBRET 1 to stop writing labels before the maximum number is written.

Checking DASD Extents: When processing an input file with all volumes mounted, you can process your extent information. After each extent is processed, use LBRET 2 to receive the next extent. When extent processing is complete, use LBRET 1 to return control to IOCS.

Checking User Standard Tape Labels: IOCS reads and passes the labels to you one at a time until a tapemark is read, or until you signify you want no more labels. Use LBRET 2 if you want to process the next label. If IOCS reads a tapemark, label processing is automatically terminated. Use LBRET 1 if you want to bypass any remaining labels.

Writing User Standard Tape Labels: Build the labels one at a time and return to IOCS, which writes the labels. You are responsible for accumulating the block count, if desired, and supplying it to IOCS for inclusion in the standard trailer label; for this, the count (in binary form) must be moved to the 4-byte field named filenameB. When LBRET 2 is used, IOCS returns control to you (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: You must process all your nonstandard labels at once. LBRET 2 is used after all label processing is completed and you want to return control to IOCS. For an example see *Appendix C.*

FEOV Macro

Name	Operation	Operand
[name]	FEOV	{filename / (1)}

The FEOV (forced end-of-volume) macro is used 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*, below.

The name of the file is the only parameter required. The name can be specified either as a symbol or in register notation.

When physical IOCS macros are used and DTFPH is specified for standard label processing, FEOV may be issued for output files 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 must only be used with physical IOCS to automatically switch volumes if SYSLST or SYSPCH are assigned to a tape output file. SEOV 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, you must check for the end-of-volume condition in the CCB.

CLOSE and CLOSER Macros

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)}]

The CLOSE or CLOSER macro is used to deactivate any file that was previously opened. Console files, however, cannot be closed. These macros end the association of the logical file declared in your program with a specific physical file on an I/O device. A file may be closed at any time by issuing this macro. No further commands can be issued for the file unless it is opened.

When CLOSER is specified, the symbolic address constants that CLOSER generates from the parameter list are self-relocating. When CLOSE is specified, the symbolic address constants are not self-relocating.

To write the most efficient code in a multiprogramming environment it is recommended that CLOSER be used.

Enter the symbolic name of the file (assigned in the DTF header entry) in the operand field. A maximum of 16 files may be closed by one macro by entering additional filename parameters as operands. Alternately, you 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. For CLOSER, the address of filename may be preloaded into any of the registers 2-15. For CLOSE, the address of filename may be preloaded into register 0 or any of the registers 2-15.

Notes:

1. If you use register notation, we recommend that you follow the standard practice of using only registers 2-12.

2. If CLOSE or CLOSER is issued to an unopened tape input file, the option specified in the DTF rewind option is performed. If CLOSE or CLOSER is issued to an unopened tape output file, no tapemark or labels are written.

PART 7

SUPERVISOR

MULTITASKING

PROGRAM LINKAGE

Supervisor Macros

CANCEL	GETIME	RELEASE
CHKPT	GETVIS	RELPAG
COMRG	JDUMP	RUNMODE
DUMP	LOAD	SETIME
EOJ	MVCOM	SETPFA
EXIT	PAGEIN	STXIT
FCEPGOUT	PDUMP	TECB
FETCH	PFIX	TTIMER
FREEVIS	PFREE	VIRTAD
GENL	REALAD	WAIT
		WAITM

Multitasking Macros

ATTACH	ENQ	POST
DEQ	FREE	RCB
DETACH		WAITM

Program Linkage Macros

CALL	RETURN	SAVE

SUPERVISOR MACROS

The supervisor itself, and the services provided by supervisor macros, are introduced in the *DOS/VS System Management Guide*, GC33-5371. The present chapter describes specific macros available to you, their interrelationship, and requirements for their usage.

Program Loading

Phases may be loaded into virtual storage from the system core image library or a private core image library with the FETCH and LOAD macros. FETCH gives control to the phase which was loaded; LOAD returns control to the phase which issued the macro.

Provided the relocating load option was selected during supervisor generation, a relocation factor will be applied to all address constants when a relocatable phase is loaded. The system calculates this relocation factor by subtracting the load address determined at link-edit time from the load address specified in (or implied by) the FETCH or LOAD macro. If the load address is implied, the system maintains the correct displacement of the entire phase relative to the beginning of the partition for which the phase was link-edited.

FETCH Macro

Name	Operation	Operand
[name]	FETCH	$\left\{\begin{matrix}\text{phasename}\\(1)\end{matrix}\right\}\left[,\left\{\begin{matrix}\text{entrypoint}\\(0)\end{matrix}\right\}\right]$ $\left[,\text{LIST=}\left\{\begin{matrix}\text{listname}\\(r)\end{matrix}\right\}\right]\left[,\text{SYS=}\left\{\begin{matrix}\text{NO}\\\text{YES}\end{matrix}\right\}\right]$ $\left[,\text{DE=}\left\{\begin{matrix}\text{NO}\\\text{YES}\end{matrix}\right\}\right]$

The FETCH macro 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 parameter. If the second parameter is not specified, or if the specification causes an unresolved reference, control is passed to the entry point determined at link-edit time. If LIST is specified, the local directory list specified by listname (or r) is scanned for the required phasename. The directory list, which is generated by the GENL macro, is assumed to consist of 34-byte entries in collating sequence. If SYS=YES is specified, the System Directory is scanned first. (This is the search order equivalent to that used when a $-phase is specified.) If either or both LIST and SYS are specified, register 1 points to a parameter list rather than to the phasename. This parameter list has the format:

```
DC     A(phasename)

DC     B'options'

DC     AL3(listname)
```

If the first byte pointed to by register 1 contains X'00', the parameter list is pointed to.

DE=YES indicates that the phasename parameter points to a directory entry rather than a phase. If the directory entry that is then in virtual storage is active, the directory scan mechanism is bypassed, if not, the entry will be filled in by the supervisor.

If relocating load was specified during supervisor generation, a relocatable phase will be loaded at the address calculated at link-edit time, adjusted with the relocation factor. Also the entry point and the address constants in the phase will then be updated using this relocation factor.

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. The phasename must be left-adjusted and padded with blanks, if necessary.

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 supervisor generation time, an increase in throughput can result by overlapping FETCH I/O operations of one partition with program execution in another partition.

GENL--Generate a Directory List

Name	Op	Operand
[name]	GENL	phase1,..........,phasen

The GENL macro generates a directory list with a 34-byte entry for each of the specified phases. The format of the local directory list is compatible with the format of the core image directory. The phase name and the length field are generated at assembly time; the remainder of the entry is filled with X'00' except the 4th byte which is made X'0B'. This local directory list can be scanned using the FETCH and LOAD macros.

LOAD Macro

Name	Operation	Operand
[name]	LOAD	{phasename | (1)} [, {loadpoint | (0)}] [, LIST= {listname | (r)}] [,SYS= {NO | YES}] [, DE= {NO | YES}] [, TXT= {YES | NO}]

The LOAD macro loads the phase specified in the first parameter and returns control to the calling phase. The phasename can be 1-8 characters long. LIST, SYS, and DE have the same function as they have when used with FETCH; for a full description of that use, look under FETCH. the TXT parameter has the same function as the DE parameter, only no text is loaded. This has two distinct advantages:

1. The directory entry can be filled in for later FETCH/LOAD calls without the overhead of text transfer.

2. You can establish whether a given phase is present by looking for the 'phase not found' condition, X'06', in the directory entry.

   In case TXT=NO is specified in combination with LIST=listname or DE=YES, the 'phase not found' condition, X'06', is set in the core directory entry if the phase is not present. The address of this condition byte minus a displacement of 16 is returned in register 0. You can also check in which library or area the phase resides.

After execution of the macro, the entry-point address of the called phase is returned to you in register 1. For a non-relocatable phase, this address is the entry-point determined at link-edit time. For a relocatable phase the entry point is adjusted by the relocation factor.

If the optional load-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. If the relocating load option was specified during supervisor generation, and the phase is non-relocatable, none of the other addresses in the phase are relocated; if the phase is relocatable, however, the entry point and address constants are updated with the relocation factor. If the optional load-address parameter is provided and the relocating load option was specified during supervisor generation, a program containing V-type address constants must not be link-edited as a relocatable phase because the updated V-type address constants will be invalid.

If the relocating load option was specified during supervisor generation and the address for LOAD is implicit (the load-address parameter is not specified), a relocatable phase will be loaded at the address calculated at link-edit time, adjusted with the relocation factor. The address constants in the relocatable phase will be updated with the relocation factor.

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. The phasename should be left-justified and padded with blanks, if necessary. If ordinary register notation is used for loadpoint, this parameter should not be preloaded into register 1.

If the physical transient overlap option is specified at supervisor generation time, an increase in throughput can result by overlapping LOAD I/O operations of one partition with program processing in another partition.

Virtual Storage

The nature of virtual storage requires that certain programs be specially handled. DOS/VS provides a set of supervisor macros to perform these special functions.

The PFIX macro allows you to fix pages of virtual mode programs in real storage until you specify their release with the PFREE macro.

The RELPAG macro allows you to release the contents of one or more pages. When a location in a page thus released is referenced again during the same program execution, it is assigned a page frame of all zeros.

The FCEPGOUT and PAGEIN macros allow you to override the selection algorithm and have specific pages paged out and in at your discretion.

With the RUNMODE macro, you can inquire in which mode -- virtual or real -your program is running. To use this macro, the supervisor must have been generated with support for page fault handling overlap. See *Appendix G*.

The SETPFA macro allows you to establish linkage to routines which are to be entered at the beginning of a page fault and when a page fault has been satisfied.

The VIRTAD macro returns the virtual address corresponding to a specified real address and the REALAD macro returns the real address corresponding to a specified virtual address.

With the GETVIS and FREEVIS macros you can dynamically retrieve and release blocks of storage in the GETVIS area of your partition or of the shared virtual area (SVA). The GETVIS area in a partition can be allocated in multiples of 2K. The GETVIS area in the SVA must be allocated in multiples of 4K. Any remainder above a multiple of 4K is ignored. The maximum size of the GETVIS area is 994K bytes in a partition and 12 168K bytes in the SVA. The amount of storage to be reserved as the GETVIS area in a partition must be set aside by means of the SIZE parameter of the EXEC job control statement. If the GETVIS area is part of the SVA, the system must have been generated with SVA=(nK,nK) in the VSTAB supervisor generation macro.

PFIX Macro

Name	Operation	Operand
[name]	PFIX	{ begin address, end address [,begin address, end address]... } or { listname / (1) }

The PFIX macro causes specific pages to be brought into real storage and fixed in their page frames until they are released at some later time. Pages may only be fixed in the real partition corresponding to the virtual partition doing the fixing (for instance, pages from BG can only be fixed in BGR). For this reason, you must have allocated a real partition large enough to contain all the pages which are likely to be fixed at any one time. In a single-partition system all except 16K of the real background partition is automatically available for fixing pages using the PFIX macro. Each time a page is fixed a counter for that page is incremented. This counter may never exceed 255 for any page.

begin address: Points to the first location of the area to be fixed.

end address: Points to the last location of the area to be fixed.

listname: Is the symbolic name of a list of consecutive 8-byte entries as shown below.

0	address constant	length minus 1

0 1 4 7

address constant: Points to the first byte of the area to be fixed.

length: A binary constant indicating the length of the area to be fixed.

A non-zero byte following an entry indicates the end of the list. Register notation may also be used.

Exceptional Conditions
If a PFIX causes the count of fixes for a page to exceed 255, the task issuing the PFIX is canceled.

If it is not possible to fix all pages requested, then none will be fixed.

If PFIX is issued in a program running in real mode it is ignored, and register 15 contains 0.

If the supervisor was not generated with PFIX=YES, the program issuing PFIX will be canceled.

Return Codes in Register 15

0 if the pages were successfully fixed.

4 if the number of pages to be fixed for one request exceeds the number of page frames in the real partition; in order for this PFIX request to be satisfied, the real partition must be reallocated and the PFIX reissued.

8 if not enough page frames available in the real partition due to previous PFIXes; this PFIX request could, however, be satisfied at another time without reallocating the real partition.

12 if one of the addresses specified was invalid.

PFREE Macro

Name	Operation	Operand
[name]	PFREE	{ beginn address, end address [,begin address, end address] . . . } or { listname / (1) }

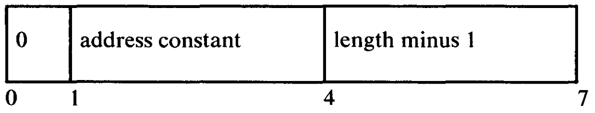
Pages in the virtual address area are each assigned a 'PFIX counter'. If a page is not fixed--that is, if it is subject to normal page management--the counter is 0. Whenever a page is fixed by using a PFIX macro its counter is increased by one. All pages whose counters are greater than 0 remain fixed in real storage.

The PFREE macro decrements the counter of a specified page by 1. A PFREE issued for a page whose counter is 0 is ignored since the page has already been freed.

begin address: Points to the first location of the area to be freed.

end address: Points to the last location of the area to be freed.

listname: Is the symbolic name of a list of consecutive 8-byte entries as shown below.



address constant: Points to the first byte of the area to be freed.

length: A binary constant indicating the length of the area to be freed.

A non-zero byte following an entry indicates the end of the list. Register notation may also be used.

Exceptional Conditions

If PFREE is issued by a program running in real mode, it is ignored.

If the supervisor was not generated with PFIX=YES specified in the FOPT macro, the program issuing PFREE will be canceled.

Return Codes in Register 15

0 if the pages were successfully freed.

12 if one of the addresses specified was invalid.

RELPAG Macro

Name	Operation	Operand
[name]	RELPAG	{ begin address, end address [, begin address, end address] . . . } or { listname / (1) }

The RELPAG macro causes the contents of one or more storage areas to be released. If the affected areas are in real storage when the RELPAG macro is executed, their contents are not saved but simply overwritten when the associated page frames are needed to satisfy pending page frame requests.

After the RELPAG macro has been executed for an area and a location in that area is referenced again during the current program execution, the related page is attached to a page frame which contains all zeros.

The storage area is released only if it contains at least one full page. You can be sure of this only if your area is 4K minus 1 byte or bigger.

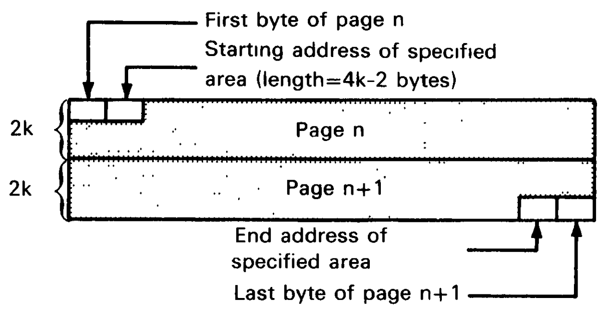

First byte of page n
Starting address of specified area (length=4k-2 bytes)
2k
Page n
2k
Page n+1
End address of specified area
Last byte of page n+1

Figure 7-1 **Worst case of an area not containing one full page**

begin address: Points to the first location of the area to be released.

end address: Points to the last location of the area to be released.

listname: Is the symbolic name of a list of consecutive 8-byte entries as shown below.

0	address constant	length minus 1

0 1 4 7

address constant: Points to the first byte of the area to be released.

length: A binary constant indicating the length of the area to be released.

A non-zero byte following an entry indicates the end of the list. Register notation may also be used.

Exceptional Conditions

- The program is running in real mode.
- The area is, fully or partially, outside of the virtual partition of the requesting program.
- A page handling request is pending for the referenced page(s).
- The page(s) is (are) fixed.
  For these pages, the RELPAG request will be ignored.
- The supervisor was not generated with PAGEIN=n in the SUPVR macro (in this case the program will be canceled).

Return Codes in Register 15

0 - All referenced pages have been released or the request has been ignored because the requesting program is running in real mode.

2 - The begin address is greater than the end address, or a negative length has been found.

4 - The area, fully or partially, does not belong to the partition where the issuing program is running. The RELEASE request has only be executed for those pages which belong to the partition of the issuing program.

8 - 1. At least one of the requested pages is temporarily fixed (via CCW-translation) and/or PFIXed. The RELEASE request has only been executed for the unfixed pages.

2. A page handling request (page fault, temporary fix, PFIX) for at least one of the requested pages is pending (caused by asynchronous processing within a partition). The RELEASE request has not been executed for those pages which are reinvolved in a page handling request.

Any combination of the return codes is possible.

FCEPGOUT Macro

Name	Operation	Operand
[name]	FCEPGOUT	{ begin address, end address [, begin address, end address] . . . } { listname (1) }

The FCEPGOUT macro causes a specific area in real storage to be paged-out at the next page fault. This request is ignored if the specified area does not contain a full page. This can happen up to an area size of 4K minus 2 bytes (see Figure 7.1).

begin address: Points to the first location of the area to be paged out.

end address: Points to the last location of the area to be paged out.

listname: Is the symbolic name of a list of consecutive 8-byte entries as shown below.

0	address constant	length minus 1

0 1 4 7

address constant: Points to the first byte of the area to be paged out.

length: A binary constant indicating the length of the area to be paged out.

A non-zero byte following an entry indicates the end of the list. Register notation may also be used.

Exceptional Conditions

- The program is running in real mode.
- The page(s) referenced by the macro is (are) outside of the requesting partition.
- The page handling request(s) is (are) pending for the referenced page(s).
- The page(s) is (are) not in real storage.
- The page(s) is (are) fixed.

  For those pages the FCEPGOUT request will be ignored.
- The supervisor was not generated with PAGEIN=n in the SUPVR macro.

  (In this case the program is canceled.)

Return Codes in Register 15

0 - All specified pages have been forced for page-out or the request has been ignored because the issuing program is running in real mode.

2 - The begin address is greater than the end address, or a negative length has been found.

4 - At least one of the requested pages do not belong to the partition where the issuing program is running. The PAGEOUT request has only been executed for those pages which are belonging to the partition of the issuing program.

8 - 1. At least one of the requested pages is temporarily fixed (via CCW-translation) and/or PFIXed. This PAGEOUT request has only been executed for the unfixed pages.

2. A page handling request (page fault, temporary fix, PFIX) for at least one of the requested page is pending (caused by asynchronous processing within a partition). The PAGEOUT-request has not been executed for those pages which are involved in a page handling request.

Any combination of return codes is possible.

PAGEIN Macro

Name	Operation	Operand

[name]	PAGEIN	{ begin address, end address [, begin address, end address]... }
		 { { listname }
		 { (1) } }
		 [{ , ECB= (ecbname) }
		 { (0) }]

The PAGEIN macro causes specific areas to be brought into real storage before their contents are needed by the requesting program. If the requested area is already in real storage the attached page frame will get low priority for the next page-outs. This function, however, does not include any fixing, so that it is not sure that all areas requested will still be in real storage when the entire request has been completed.

begin address: Points to the first location of the area to be paged in.

end address: Points to the last location of the area to be paged in.

listname: Is the symbolic name of a list of consecutive 8-byte entries as shown below.

0	address constant	length minus 1
0	1	4 7

address constant: Points to the first byte of the area to be paged in.

length: A binary constant indicating the length of the area to be paged in.

A non-zero byte following an entry indicates the end of the list. Register notation may also be used.

ECB=ecbname: Specifies the name of the ECB, a fullword defined by your program, which is to be posted when the operation is complete. Register notation may also be used. The ECB parameter is optional.

Return Information

The return information can be obtained from the ECB, byte 2.

Bits of ECB byte 2:	Meaning if bit is one:
0	PAGEIN request is finished.
1	The page table is full, the request cannot be queued at this time for further handling. The request is ignored, bit 0 is set.
2	One or more of the requested pages are outside the requesting program's partition. PAGEIN is not performed for these pages.
3	At least one negative length has been detected in the area specifications, PAGEIN is not performed for these areas.
4	List of areas that are to be paged in is not completely in the requesting program's partition. The request is ignored, bit 0 is set.
5	Paging activity is too high in the system, no performance improvement is possible.

Use the WAIT macro with the ecbname as operand for completion of the PAGEIN macro, before the return code is tested.

Any combination of the return bits in the ECB is possible.

RUNMODE Macro

Name	Operation	Operand
[name]	RUNMODE	

The RUNMODE macro returns the following information to the program issuing it:

- Register 1 contains 0 if the issuing program is running in virtual mode.
- Register 1 contains 4 if the issuing program is running in real mode.

No operand is required for this macro.

SETPFA Macro

Name	Operation	Operand
[name]	SETPFA	$\left\{\begin{matrix}\text{entry address}\\ (0)\end{matrix}\right\}$

The SETPFA macro either establishes or terminates linkage to a page fault appendage routine that is to be entered each time a page fault occurs or is completed. You will find more information on how to write such a routine in *Appendix G* of this manual.

If an entry address is specified, then the routine pointed to by the address will be entered every time a page fault in its task occurs or is satisfied. The routine to be entered and all areas referenced by the routine must be fixed in real storage using the PFIX macro before SETPFA is issued. The entry address may be specified as a symbol or in register notation.

If SETPFA is issued without an operand, the linkage to the page fault appendage is terminated. Each issuance of SETPFA supersedes all previous SETPFA's for that task.

The page fault appendage is only called when a page fault occurs in the task owning the appendage. If a page fault occurs in a supervisor service working for the owning task, the appendage is not called.

See *Appendix G* for instructions on setting up a page fault appendage.

VIRTAD Macro

Name	Operation	Operand
[name]	VIRTAD	$\left\{\begin{matrix}\text{address}\\ (1)\end{matrix}\right\}$

The VIRTAD macro returns the virtual address corresponding to a specified real address.

address: Is the real storage address to be converted. It can be given as a symbol or in register notation.

Register 0 returns the virtual address corresponding to the specified real address only if the page frame containing the specified real address contains a fixed page; otherwise register 0 contains 0.

Note:

- The pages of a program running in real mode are considered to be fixed.

- If the supervisor has been generated without the parameter ECPREAL=YES in the FOPT supervisor generation macro instruction the issuing task is canceled.

REALAD Macro

Name	Operation	Operand
[name]	REALAD	{address | (1)}

The REALAD macro returns the real address corresponding to a specified virtual address.

address: Is the virtual address to be converted. It can be given as a symbol or in register notation.

Register 0 returns the real address corresponding to the specified virtual address if and only if the virtual address points to a fixed page, otherwise register 0 contains 0.

Note:

- The pages of a partition running in real mode are treated as if they were fixed.

- If the supervisor is generated without the parameter ECPREAL=YES in the FOPT supervisor generation macro the issuing task is canceled.

GETVIS Macro

Name	Operation	Operand
[name]	GETVIS	[ADDRESS= {name1 | (1)}]
		[, LENGTH= {name2 | (0)}]
		[, PAGE= {NO | YES}]
		[, POOL= {NO | YES}]
		[, SVA= {NO | YES}]

The GETVIS macro retrieves a block or blocks of storage from the GETVIS area of your partition or of the SVA.

The start address (ADDRESS) of the requested virtual storage area is returned by the system either in the 4-byte field addressed by name1 or in the specified register. (Register 15 must not be used because it contains the return code.) The returned address is only valid if the return code in register 15 is zero. If the operand is omitted, the address is returned in register 1.

The length (LENGTH) of the requested storage block may be specified by you either in the 4-byte field addressed by name2 or in a register. The length is specified in bytes. The smallest unit that can be requested by GETVIS is (a) 128 bytes if the GETVIS area is part of a partition or (b) 512 bytes if the GETVIS area is part of the SVA. If the specified length is not a multiple of 128 or 512, respectively, it is rounded to the next higher multiple of 128 or 512. If the operand is omitted, the system assumes that you have specified the length in register 0.

If you want the requested storage area to start on a page boundary, specify PAGE=YES. This may reduce page faults.

If POOL is specified, GETVIS starts searching for the requested virtual storage area at the address specified in register 1. In this case, it is your responsibility to provide a meaningful address in register 1.

If SVA=YES is specified, the GETVIS area in the SVA is used. Otherwise, the GETVIS area of the current partition is taken.

User programs can only use the SVA if they have a storage protection key of zero.

Return Codes in Register 15

0	GETVIS completed successfully.
4	The GETVIS area is OK.
8	The GETVIS macro was issued by a program running in real mode.
12	No more virtual storage is available in the GETVIS area, or the length specified is smaller than zero.

FREEVIS Macro

Name	Operation	Operand
[name]	FREEVIS	$\left[\text{ADDRESS=}\begin{Bmatrix}\text{name1}\\(1)\end{Bmatrix}\right]$ $\left[\text{,LENGTH=}\begin{Bmatrix}\text{name2}\\(0)\end{Bmatrix}\right]$ $\left[\text{, SVA=}\begin{Bmatrix}\text{NO}\\\text{YES}\end{Bmatrix}\right]$

The FREEVIS macro releases a block (or blocks) of virtual storage that was retrieved by the GETVIS macro.

The start address (ADDRESS) of the virtual storage block to be released in the GETVIS area may be specified by you either in a 4-byte field addressed by name1 or in a register. If the operand is omitted, the system assumes that you have specified the address in register 1.

The length (LENGTH) of the virtual storage block to be released may be specified by you in a 4-byte field addressed by name2 or in a register. The length is specified in bytes. The smallest unit of virtual storage that can be released by FREEVIS is (a) 128 bytes if the GETVIS area is part of a partition or (b) 512 bytes if the GETVIS area is part of the SVA. If the specified length is not a multiple of 128 or 512, respectively, it is rounded to the next higher integral multiple of 128 or 512. If the operand is omitted, the system assumes that you have specified the length in register 0.

If SVA=YES is specified, the GETVIS area in the SVA is used. Otherwise, the GETVIS area of the current partition is taken.

User programs can only use the SVA if they have a storage protection key or zero.

Return Codes in Register 15

8 The FREEVIS macro was issued by a program running in real mode.

12 The specified address is not within the GETVIS area or the address is not (a) a multiple of 128 bytes if the GETVIS area is part of a partition, or (b) a multiple of 512 bytes if the GETVIS area is in the SVA.

16 The specified storage block (ADDRESS+LENGTH) to be released exceeds the GETVIS area, or the length specified is smaller than zero.

If the return code is not zero, no action is taken by FREEVIS.

Program Communication

For each partition the supervisor contains a storage area called the communication region. The supervisor uses the communication region, and your program also can use it. Your program can check the communication region of the partition in which your program runs; your program can also modify the user area of this communication region.

Figure 7-2 shows the portion of the communication region containing information of interest. This information is also described below.

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.
4 bytes	Reserved.
11 bytes	User area for communication within a job step or between job steps. All 11 bytes are 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. If the program was initiated with the SIZE parameter in the EXEC job control statement, this address gives the highest byte of the area determined by the SIZE parameter. If the SIZE parameter was not specified, the address is the highest address in the partition (either real or virtual).
4 bytes	Address of the uppermost byte of the current phase placed in the program area by the last FETCH or LOAD macro in the job.

4 bytes	Highest ending virtual 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 if the program loads any of these phases above its link-edited origin address and the relocating loader is not used. 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 program label area.

The COMRG and MVCOM macros allow your program to check and to modify the communication region.

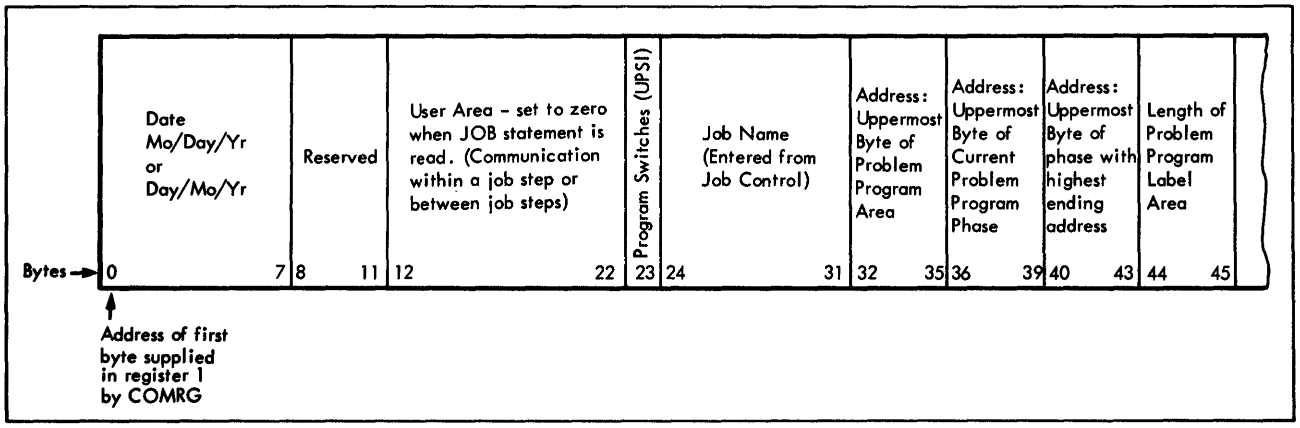

Date Mo/Day/Yr or Day/Mo/Yr
Reserved
User Area - set to zero when JOB statement is read. (Communication within a job step or between job steps)
Program Switches (UPSI)
Job Name (Entered from Job Control)
Address: Uppermost Byte of Problem Program Area
Address: Uppermost Byte of Current Problem Program Phase
Address: Uppermost Byte of phase with highest ending address
Length of Problem Program Label Area
Bytes
0 7 8 11 12 22 23 24 31 32 35 36 39 40 43 44 45
Address of first byte supplied in register 1 by COMRG

Figure 7-2 **Communication region**

COMRG Macro

Name	Operation	Operand
[name]	COMRG	

The COMRG macro places the address of the communication region of the partition from which the macro is issued in register 1. Your program can read any portion of its own partition's communication region by using register 1 as a base register.

MVCOM Macro

Name	Operation	Operand
[name]	MVCOM	to,length, $\left\{\begin{array}{l}\text{from}\\ (0)\end{array}\right\}$

The MVCOM macro modifies the content of bytes 12-23 of the communication region of the partition from which the macro is issued.

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 address (relative to the first byte of the region) of the first communication region byte modified (12-23).

The following example shows how to move three bytes from the symbolic location DATA into bytes 16-18 of the communication region:

```
MVCOM 16,3,DATA
```

Releasing I/O Units

RELEASE Macro

Name	Operation	Operand
[name]	RELEASE	(SYSnnn,...,SYSnnn) [,savearea]

RELEASE specifies the names of programmer logical units to be released. RELEASE may be used only for units used within a given partition running in batched job mode. Up to 16 programmer logical units may be specified in the parameter list for release.

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 your program. If not provided, the contents or registers 0 and 1 are destroyed.

All the units specified are checked to assure that no system logical units are requested for release. If system logical units are specified, an MNOTE is issued and such units are 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: You should inform the system operator via a message that the assignment was released.

Time of Day Macro

GETIME Macro

Name	Operation	Operand
[name]	GETIME	$\left\{\begin{array}{l}\underline{\text{STANDARD}}\\\text{BINARY}\\\text{TU}\\\text{MICRO}\end{array}\right\}\left[,\left\{\begin{array}{l}\underline{\text{LOCAL}}\\\text{GMT}\end{array}\right\}\right]$

The GETIME macro obtains the time-of-day at any time during program execution, provided the time of day option was specified at system generation. (If the time of day option was not specified at system generation, issuing GETIME only obtains zeros instead of a valid time.) STANDARD and LOCAL are assumed if no GETIME operands are given.

The time-of-day clock is independent of the interval timer options (for the latter see *Interval Timer and Exit Macros*, below). The use of GETIME and use of interval timer macros have no effect on one another.

As long as no DATE job control statement is supplied, the job date and system date in the communication region are updated every time GETIME is issued. Those dates are therefore accurate at any given moment during processing as long as no DATE statement is encountered. However, when the job stream contains a DATE job control statement, only the system date in the communication region is updated when GETIME is used; the job date is not changed in that case.

If STANDARD is specified, the time-of-day is returned in register 1 as a packed decimal number of the form *hhmmss*, where *hh* is hours, *mm* is minutes, and *ss* is seconds, with the sign in the low-order half-byte. The time-of-day may be stored, unpacked, or edited.

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 seconds.

If MICRO is specified, the time-of-day is returned in registers 0 and 1 as a 64-bit binary integer in microseconds; bit 63 represents the unit of microseconds. If MICRO is specified, GMT must also be specified. In case MICRO is specified by itself or in combination with LOCAL, the specification is not accepted and GMT is assumed. (LOCAL is not accepted in combination with MICRO because the conversion routines from Greenwich Mean Time to local time take up a great number of microseconds). The system date in the communication region (offset 79) is not updated when MICRO is specified.

Interval Timer and Exit Macros

The interval timer macros--SETIME, TTIMER, WAIT, WAITM, TECB, STXIT IT, and EXIT IT--can be used only if the supervisor contains the interval timer routines. You specify at system generation time whether the interval timer is to be supported.

In a multiprogramming and/or multitasking environment, all tasks--main as well as subtasks--may use the interval timer macros, if the interval timer was specified at system generation.

There are two distinct methods, described below, of using the interval timer macros. In each task only one method can be used at a time.

The first method allows your program to set the timer and enter a routine in your program when the time elapses. The SETIME, TTIMER, STXIT, and EXIT macros do this.

In the second method, it is possible to put the task into the wait state until the time interval has elapsed. The SETIME, TTIMER, TECB, WAIT, and WAITM macros are used for this method.

Entering a Routine When Time Elapses

SETIME Macro

Name	Operation	Operand
[name]	SETIME	{seconds | (1)}

The SETIME macro sets the interval timer to the value 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 you supply is entered.

If a routine is not supplied to the supervisor (via the STXIT macro) 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.

TTIMER Macro

Name	Operation	Operand
[name]	TTIMER	[CANCEL]

The TTIMER macro is used to test how much time has elapsed of an interval which was set in the same task by the associated SETIME macro. The TTIMER macro returns the time remaining of the interval, expressed in hundredths of seconds in binary, in register 0.

If CANCEL is specified, the remaining time of the interval set in that task is canceled and the interval timer routine is not entered (See *DOS/VS System Management Guide*, GC33-5371, for programming considerations and examples).

STXIT Macro

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 your program's routine for processing abnormal task termination, interval timer, program check, or operator communication interrupts. Use the EXIT macro (described later in this section) to return from these routines. This linkage must be established before an interrupt occurs. If only the first operand is present, linkage to your routine is terminated.

Hexadecimal representation	**Specific abnormal termination code meaning**
10	Normal EOJ
11	No channel program translation for unsupported device
12	Insufficient buffer space for channel program translation
13	CCW with count greater than 32K
14	Page pool too small
15	Page fault in disabled program (not a supervisor routine)
16	Page fault in MICR stacker select or page fault appendage routine
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 interrupt status information)
1B	Channel failure
1C	CANCEL ALL macro issued in another task
1D	Main task terminated with subtask still attached
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
1F	CPU failure
20	A program check occurred
21	An invalid SVC was issued by the problem program or macro
22	Phase not found in the core image library
23	CANCEL macro issued
24	Canceled due to an operator request
25	Invalid virtual storage address given (outside partition)
26	SYSxxx not assigned (unassigned LUB code)
27	Undefined logical unit
28	QTAM cancel in progress
29	Relocatable phase fetched or loaded by a supervisor without relocating loader support
2A	I/O error on page data set
2B	I/O error during fetch from private core image library
2C	Page fault appendage routine passed illegal parameter to supervisor

Figure 7-3 Abnormal termination codes (part 1 of 2)

Hexadecimal representation	Specific abnormal termination code meaning
2D	Program cannot be executed/restarted due to a failing storage block
2E	Invalid resource request (possible deadlock)
2F	More than 255 PFIX requests for one page
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
35	Job control open failure
36	Page fault in I/O appendage routine
38	Wrong privately translated CCW
39	Reserved
FF	Unrecognized cancel code

Figure 7-3 **Abnormal termination codes (part 2 of 2)**

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 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 (see Figure 7-3).
- Register 1 contains the address of the task's abnormal-termination save area, which contains the interrupt status information and the contents of registers 0-15 at the time of the abnormal termination. Details of the status information are contained in the section *Save Areas* in *DOS/VS Serviceability Aids and Debugging Procedures*, GC33-5380.

The abnormal termination routine can then examine this data and take whatever action is necessary.

Macros which might be used in this routine are 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, your program's abnormal termination routine should avoid macros such as ENQ, CHKPT, and any other I/O macros which may cause an abnormal termination.

Note: For systems operating in a QTAM environment, QTAM files must be closed before issuing this macro.

After the appropriate action is taken, your abnormal termination routine should end with a CANCEL, DETACH, DUMP, or EOJ macro. 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 specified interval elapses.

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 console and request the foreground OC routine. In case of 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. The address can be specified as a symbol or in special or ordinary register notation.

savearea
Address of a 72-byte area in which the supervisor stores the old interrupt status information and general registers 0-15, in that order. The address can be specified as a symbol or in special or ordinary register notation. Your program must have a separate save area for each routine that is included.

If a STXIT macro is issued and the supervisor is not generated to handle the requested facility, the job is abnormally terminated.

If an abnormal termination condition occurs and linkage has not been established to an abnormal termination routine, processing in 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, processing in 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	I	I	I	T
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 the 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 checkpoint, any STXIT linkages established prior to the checkpoint are destroyed.

Note 2: If a task is using a logical transient routine when a timer interrupt occurs, your 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.

Note 5: Your exit routine will run under the associated task PIB and may be located anywhere in the program.

Note 6: If a timer interrupt occurs and your associated exit routine is in process, the interrupt will be ignored. (This can only occur if a short time interval has been issued in your exit routine.)

Note 7: If an operator communication interruption routine or a program check interruption routine is in process when a timer interrupt occurs, your timer routine will be processed; when it completes, control returns to interrupted routine.

Note 8: When subtasks are detached or canceled, associated time intervals and exit linkages are cleared.

Note 9: Timer intervals will not be restarted when a program is restarted from a checkpoint.

EXIT Macro

Name	**Operation**	**Operand**
[name]	EXIT	{PC | IT | OC | MR}

The EXIT macro is used to return from your routine, to the instruction in your interrupted program immediately after the instruction where the interruption occurred. Your routine is specified in the STXIT macro (except for MR).

For PC, IT, and OC, the interrupt status information and registers are restored from the save area; thus, the save area contents should not be destroyed. The operands have the following meanings:

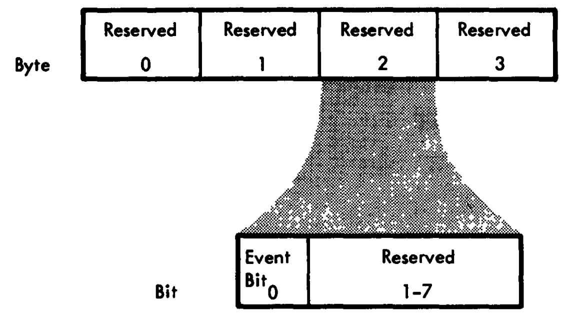
PC	Exit from your program check routine.
IT	Exit from your interval timer routine.
OC	Exit from your routine which handles the operator attention interrupt.
MR	The MR indicates that your stacker selection routine (MICR document processing) exits to the external interrupt routine of the supervisor. The name of your stacker selection routine is specified in the DTFMR macro.

Executing a Program at Given Intervals

TECB Macro

Name	Operation	Operand
[name]	TECB	

The TECB macro generates a timer event control block (see Figure 7-4) 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 7-4 **Timer event control block (TECB)**

SETIME Macro

Name	Opera-tion	Operand
[name]	SETIME	$\left\{\begin{matrix}\text{seconds}\\ (1)\end{matrix}\right\}$, $\left\{\begin{matrix}\text{tecbname}\\ (r)\end{matrix}\right\}$

The SETIME macro sets the amount of time that must elapse before the TECB event bit is set to 1 and the execution of the program is resumed with the instruction following the WAIT macro. 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.

You can specify the tecbname or specify the register in which the address of the corresponding TECB is placed. (Registers 0 and 1 must not be used.) After SETIME is executed, the supervisor returns the TECB address in register 1.

TTIMER Macro

Name	Operation	Operand
[name]	TTIMER	[CANCEL]

The TTIMER macro is used to test how much time has elapsed of an interval which was set in the same task by the associated SETIME macro. The TTIMER macro returns the time remaining of the interval in register 0.

If CANCEL is specified, the time interval set in that task is canceled, and the event bit in the associated TECB is set on. This bit indicates to the task issuing the WAIT or WAITM that a time interval has elapsed or has been canceled.

WAIT Macro

Name	Operation	Operand
[name]	WAIT	$\left\{\begin{matrix}\text{tecbname}\\ (1)\end{matrix}\right\}$

The WAIT macro sets programs or tasks into the wait state until the timer interval specified in SETIME has elapsed (event bit in the TECB turned on) before execution of the program or task issuing the WAIT continues. Each task of the system can wait on any TECB within the same partition. When a WAIT macro is processed in a multiprogramming and/or multitasking environment, control is given to the supervisor, which makes the time available to another task or partition.

You can either specify the tecbname or use register notation. The WAIT macro loads the TECB address into register 1 unless a different register is specified.

Note: The SETIME macro leaves the TECB address in register 1.

WAITM Macro

Name	Operation	Operand
[name]	WAITM	{ecb1,ecb2... / listname / (1)}

The WAITM macro sets programs or tasks into the wait state until one of the events specified has occurred (event bit turned on) before execution of the program or task issuing the WAITM continues. One of the events to be waited upon can be the elapsing of the interval specified in SETIME.

The operand provides the address of the ECBs--such as the TECB--to be waited upon.

The symbolic names 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 specified as a symbol, in special register notation, or in ordinary register notation.

The WAITM macro loads the address of the posted ECB into register 1. (The SETIME macro also places the TECB address in register 1.)

Dump Macros

PDUMP Macro

Name	Operation	Operand
[name]	PDUMP	{address1 / (r)} , {address 2 / (r)}

This macro provides a hexadecimal dump of the general registers and of the virtual storage area contained between the two address expressions (address1 and address2). One or both of the addresses can be given in registers. If address2 is not greater than address1, or address1 is greater than the highest address in virtual storage, the macro results in no operation. If the value in address2 is greater than the end of real storage, the virtual storage between address1 and the end of virtual storage is dumped. The contents of registers 0 and 1 are destroyed, but the CPU status is retained. Thus, PDUMP furnishes a dynamic dump (snapshot) useful for program checkout. Processing continues with your next 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, you must issue an OPEN or OPENR macro to any DTF assigned to SYSLST after each PDUMP that is executed. The OPEN or OPENR macro updates the disk address maintained in the DTF table to agree with the address where the PDUMP output ends. If the OPEN or OPENR is not issued, the address is not updated, and the program is canceled when the next PUT is issued.

If nonaddressable areas were included in the range of PDUMP, a message will be printed to indicate this (nonaddressable areas are explained in the *DOS/VS System Management Guide*, GC33-5371).

DUMP Macro

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.

If DUMP is issued by a job in real mode, only that part of the partition available to the program will be dumped. This is the part limited by the SIZE parameter of the EXEC job control statement if it was specified, or the active partition if SIZE was not specified. If DUMP is issued by a program running in virtual mode, the entire virtual partition is dumped.

JDUMP Macro

Name	Operation	Operand
[name]	JDUMP	

The JDUMP macro gives a hexadecimal dump of the supervisor, the partition issuing the macro, and the general registers. If JDUMP is issued by a main task, the entire job is terminated. If JDUMP is issued by a subtask, the issuing task is detached and the dump is taken. The dump is always directed to SYSLST upon JDUMP macro execution. SYSLST, if disk or tape, must be opened, and if tape it must be positioned as desired.

If JDUMP is issued by a program running in real mode, only that part of the partition available to the program is dumped. This is the part limited by the SIZE parameter of the EXEC job control statement if it was specified, or the entire real partition if it was not specified. If JDUMP is issued by a program in virtual mode, the entire virtual partition is dumped.

Cancel and EOJ Macros

CANCEL Macro

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 all processing in 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.

If the CANCEL macro is issued without an operand, the macro 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 main task with subtasks attached.

EOJ Macro

Name	Operation	Operand
[name]	EOJ	

The EOJ macro is issued in the main task or in the 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 Program

A general description of how to checkpoint a program, and suggestions and restrictions for use of the CHKPT macro, can be found in the *DOS/VS System Management Guide*, GC33-5371, and *DOS/VS Data Management Guide*, GC33-5372.

CHKPT Macro

Name	Op	Operand
[name]	CHKPT	SYSnnn, $\left\{\begin{matrix}\text{restart address}\\ \text{(r1)}\end{matrix}\right\}$ $\left[,\left\{\begin{matrix}\text{end address}\\ \text{(r2)}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\text{tpointer}\\ \text{(r3)}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\text{dpointer}\\ \text{(r4)}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\text{filename}\\ \text{(r5)}\end{matrix}\right\}\right]$

The CHKPT macro is used to record the status of your program so that, in the event that processing is terminated before the program has completed processing, the program may be restarted using job control.

Special register notation cannot be used with any of the CHKPT macro operands.

SYSnnn
Specifies the logical unit on which the checkpoint information is to be stored. It must be an EBCDIC magnetic tape or a disk pack (see *Checkpoint File* later in this section).

restart address (or r1)
Specifies a symbolic name of the 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 program area required for restart (see *Program Communication*, above). This address must follow the logic modules being included from the relocatable library, and must be a multiple of 2K. If not, it is rounded to the next 2K boundary. If this operand is omitted, all storage allocated to the partition is checkpointed.

If a checkpoint is taken in a program running in real mode, the implied end address is the one specified by the SIZE parameter of the EXEC job control statement if specified. Otherwise it is the end of the real partition.

If a checkpoint is taken in a program running in virtual mode and using VSAM or the GETVIS/FREEVIS macros, you must be sure to specify the end of the virtual partition as the end address operand.

This operand has two advantages:

1. Less time and space is required for recording the checkpoint record set.
2. If a program using 96K of storage is being run in a larger system and only 96K is checkpointed, that program can be restarted either on a 96K 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. (The use of this field is described under *Repositioning Magnetic Tape*, later in this section.)

dpointer (or r4)
The symbolic name of a DASD operator verification table which you can set up in your own area of storage. (See *DASD Operator Verification Table*, later in this section.)

filename (or r5)
Used only for checkpoint records on disk. It is the name of the associated DTFPH macro. (For additional information see *Checkpoints on Disk*, later in this section.)

Information That Is 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*, above).
- All DASD file protection extents attached to logical units that belong to the checkpointed program.
- Real and virtual addresses of any PFIXed pages
- PFIX-counter of any PFIXed pages
- Real partition limits of any PFIXed pages

Information That 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 your restart routine.)
- Any linkages to your routines set by the STXIT or SETPFA macros. (If needed, STXIT or SETPFA should be used in your restart routine.)
- Any timer values set by the SETIME macro. (If needed, SETIME should be used in your restart routine.)
- The program mask in your program's PSW. (If anything other than all zeros is desired, the mask should be reset in your restart routine.)

Considerations for DASD, Diskette, MICR, and 3886 Files
DASD or diskette system input or output files (SYSIPT, SYSLST, etc) must be reopened at restart

time. In your restart routine, you must be able to identify the last record processed before the checkpoint.

For MICR files, your 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 your restart routine which clears the document buffer.

For 3886 files, the SETDEV macro must be issued at restart time. This ensures that the proper format record will be loaded into the 3886 if the job must be restarted. If the job processing the 3886 data set uses line marking with reflective ink, the job can not be restarted.

Checkpoint File

The checkpoint information must be written on disk or on an EBCDIC magnetic tape (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 records are written in data convert mode, with odd parity.

Checkpoints on Tape

You can either establish a separate file for checkpoints or embed the checkpoint records in an output file. When the file is read at a later time using LIOCS, the checkpoint records are automatically bypassed. If physical IOCS is used, you must program to bypass the checkpoint record sets (see the *Physical IOCS Macros* chapter).

If a separate magnetic tape checkpoint file with standard labels is maintained, the labels should be either checked by an OPEN or OPENR 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 disk volume 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:

$$n\left[\frac{1+\frac{w}{15}+\frac{x}{30}+\frac{y}{20}}{18}+\frac{c}{z}\right]$$

c = the number of bytes to be checkpointed in your program up to the end address specified in the CHKPT macro or by the SIZE parameter of the EXEC job control statement.

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.)

v = the following values:

for 2311	18
for 2314/2319	20
for 3330/3333	49
for 3340	27

w = maximum number of page frames which are fixed by PFIX at the time the checkpoint is taken.

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 = the following values

for 2311	3000
for 2314/2319	6000
for 3330/3333	12000
for 3340	7000

For each division, the quotient is rounded 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 or OPENR and the checkpoint routine.

Repositioning I/O Files

The I/O files used by the checkpointed program must be repositioned on restart to the record you want to read or write next. Checkpoint provides no aids for repositioning unit-record files. You must establish your 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.

Sequential DASD input, output, and work files require no repositioning.

When updating DASD records in an existing file, you must be able to identify the last record updated at checkpoint in case you need 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 7-5 describe the procedure. The fourth operand of the CHKPT macro points to two V-type address constants which you specify in your 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.

You 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, you 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 repositioned to the beginning of the desired file.
- The correct volume of a multi-volume 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 you keep 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 your 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 tape-mark and the position at checkpoint.

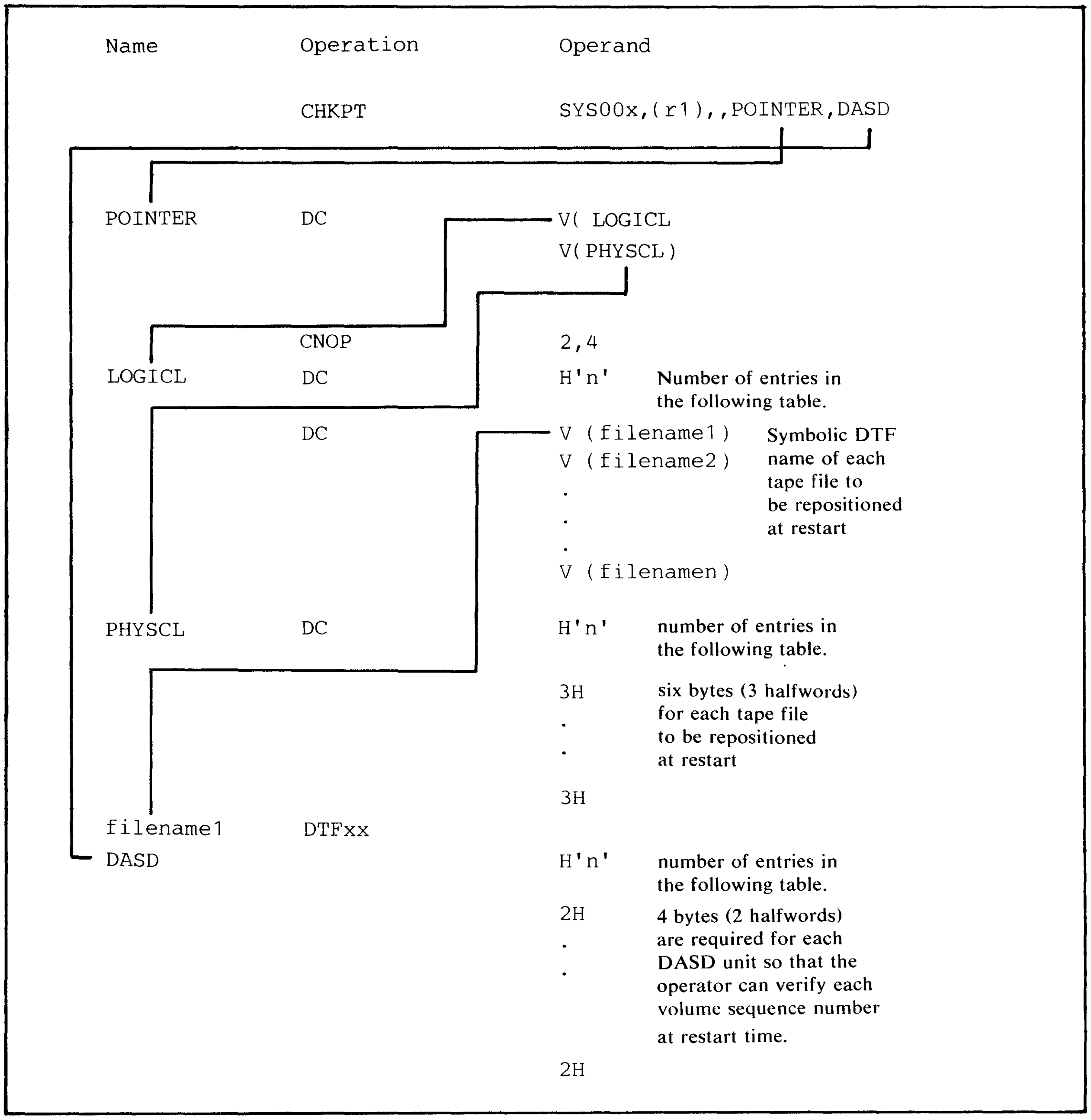
DASD Operator Verification Table

If the **dpointer** operand of the CHKPT macro is used, you can build a table (in your own area of virtual storage) to provide the symbolic unit number and the bin (cell) number of each DASD file used by your 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.



```
Name          Operation        Operand

              CHKPT            SYS00x,(r1),,POINTER,DASD

POINTER       DC               V( LOGICL
                               V(PHYSCL)

              CNOP             2,4
LOGICL        DC               H'n'      Number of entries in
                                         the following table.
              DC               V (filename1)   Symbolic DTF
                               V (filename2)   name of each
                               .               tape file to
                               .               be repositioned
                               .               at restart
                               V (filenamen)

PHYSCL        DC               H'n'      number of entries in
                                         the following table.
                               3H        six bytes (3 halfwords)
                               .         for each tape file
                               .         to be repositioned
                                         at restart
                               3H
filename1     DTFxx
DASD                           H'n'      number of entries in
                                         the following table.
                               2H        4 bytes (2 halfwords)
                               .         are required for each
                               .         DASD unit so that the
                                         operator can verify each
                                         volume sequence number
                                         at restart time.
                               2H
```

Figure 7-6 **Repositioning magnetic tape**

MULTITASKING MACROS

The *DOS/VS System Management Guide*, GC33-5371, gives a general description of multitasking, some helpful techniques for using the multitasking macros, and examples of their use.

The present chapter describes the multitasking macros themselves. The macros are used for initiating and terminating subtasks, for resource protection, for intertask communication, and for DASD track protection. At the end of the chapter macro considerations for using shared modules and files are discussed.

Subtask Initiation and Normal Termination Macros

ATTACH Macro

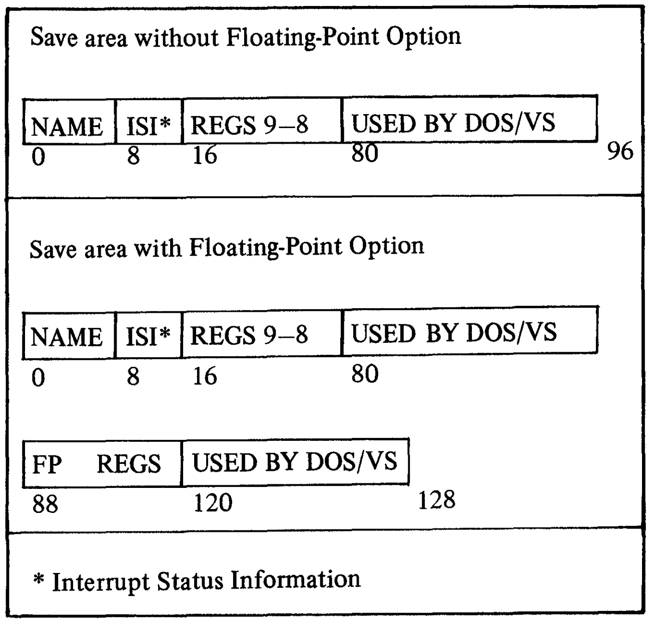
Name	Op	Operands
[name]	ATTACH	{entrypt | (r0)} , SAVE= {savearea | (r1)} [,ECB= {ecbname | (r2)}] [,ABSAVE= {savearea | (r3)}]

A subtask can only be inititated by issuing the ATTACH macro within the main task. The part of the subtask containing the entry point must be in storage before the subtask can be successfully attached.

Normally all the tasks are linked and cataloged together. It is possible, however, to LOAD a phase into storage just before issuing the ATTACH macro.

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 save area for the subtask. The second operand can be given as a symbol, or in special or ordinary register notation. The save area is 96 or 128 bytes in length depending upon whether or not the floating-point option (CONFG FP=YES) was specified at system generation time.

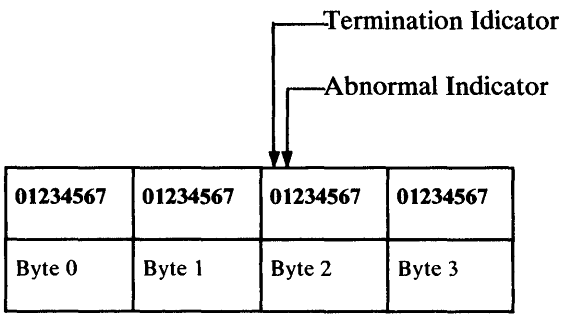


Save area without Floating-Point Option
NAME
ISI*
REGS 9–8
USED BY DOS/VS
0
8
16
80
96
Save area with Floating-Point Option
NAME
ISI*
REGS 9–8
USED BY DOS/VS
0
8
16
80
FP REGS
USED BY DOS/VS
88
120
128
* Interrupt Status Information

Figure 7-6 **Subtask save area**

The save area (see Figure 7-6) contains the subtask's interrupt status information, general-purpose registers, floating point registers (option-dependent), and a 16-byte area used by DOS/VS. A subtask name should be provided in the first 8 bytes of the save area. The name is used to identify 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 your 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 set to 1 when the subtask terminates abnormally; that is, if task termination is not the result of issuing the CANCEL, DETACH, DUMP, or EOJ macros. 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 Your Routine(s)* in the *Supervisor Macros* chapter. Your 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 the main task registers 2-15 will be passed to the appropriate subtask registers. 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.

The maximum possible number of subtasks is 13 for a two-partition system, 12 for a three-partition system, 11 for a four-partition system, or 10 for a five-partition system. In the event that the maximum possible number of subtasks is already attached, any attempt to attach another subtask will be unsuccessful. In this event the main task will keep control and register 1 (main task) will contain the address of an ECB within the supervisor that will be posted when the system can initiate another subtask. Register 1 will also have the high order bit 0 on to aid the main task in testing for an unsuccessful ATTACH. (See description of multitasking in the *DOS/VS System Management Guide*, GC33-5371, for detailed programming considerations.

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 DETACH macro without specifying an operand, all programs in the partition are terminated abnormally.

The DETACH macro sets byte 2 bit 0 of the ECB to 1 (if specified in the ATTACH macro) 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.

Resource Protection Macros

RCB Macro

When two or more tasks in the same partition manipulate 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.

Name	**Operation**	**Operand**
[name]	RCB	

The RCB macro generates an 8-byte word-aligned Resource Control Block (RCB),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. 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

All bits of byte 0 of this RCB are set to 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.

ENQ Macro

Name	Operation	Operand
[name]	ENQ	rcbname

A task protects a resource by issuing an ENQ (enqueue) macro. When the RCB, (identified by the rcbname) 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 but attempts to dequeue that resource is terminated, unless 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 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	{rcbname (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 the ENQ macro 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 virtual storage:

```
MTASK START  0      Example
        .
        .
        .
      ATTACH STASK1,SAVE=SVE1,ECB=ECB1
      ATTACH STASK2,SAVE=SVE2,ECB=ECB2
        .
        .
        .
STASK1ENQ    RCBA
      update TOTAL
      DEC    RCBA
        .
        .
        .
STASK2ENQ    RCBA
      update TOTAL
      DEQ    RCBA
        .
        .
        .
RCBA  RCB
TOTAL DS or DTFxx
```

TOTAL can be simply an area in virtual 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 all references to TOTAL, TOTAL is protected during the time each task takes to process instructions from the task's ENQ macro to its DEQ macro. This is readily apparent if

TOTAL is an area in virtual storage. However, if TOTAL is a file, the record that is being operated upon is protected while in virtual storage, but it is not necessarily protected on the external storage device. If the file is on a DASD, the HOLD function should therefore 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 your 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 specified 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 which is to post 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 routine of your program. In this particular case, a branch instruction points to a table containing a list of ECBs with corresponding branch instruction 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 (it turns 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 by this issuance of the POST macro. This does not guarantee that they will stay in the wait state until another POST is issued. On the contrary, other events could cause the other tasks to be dispatched. For this reason the POST macro should not be used with the SAVE operand to control subtask operation unless separate ECB's are used. Otherwise, it should be only used to save time. 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 one task 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 specifies the HOLD=YES operand in its DTFSD-SDMOD, DTFIS-ISMOD, and/or DTFDA-DAMOD 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, 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 program must issue the FREE macro.

For situation 3, the 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(R) macro for that file, or a DETACH macro for that task.

For situation 4, the method of implementation depends on the function being performed.

FREE Macro

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 your 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/O 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 your 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 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(R) 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. Your 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. Your 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, you should not issue ESETL if SETL has not been successful.

Shared Modules and Files

The DTF and logic modules for the card, device

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 MODNAME operand of the corresponding DTF macro.

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 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 *Interrelationship of the Macros* in *The Macro System* chapter.

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. You can either supply a separate set of label statements (DLBL-EXTENT, TLBL, etc) for each corresponding DTF filename, or you 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.

independent, direct access, indexed sequential, printer, sequential disk, diskette, and tape macros 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 task requires its own unique save area. The fact that the module save areas are unique 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 read-only modules should not be confused with save areas required for multitasking macros.

If RDONLY=YES is omitted, the module generated is not reenterable, and no save area need be established. If an ERROPT or WLRERR routine issues I/O macros that use the same read-only module that passed control to either error routine, your 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 the task.

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 program ignores module considerations, two tasks may attempt to use a single nonreentrant module. When this occurs, unpredictable 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.

PROGRAM LINKAGE MACROS

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 macros 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 program is referred to as direct linkage.

Figure 7-7 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 the 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.

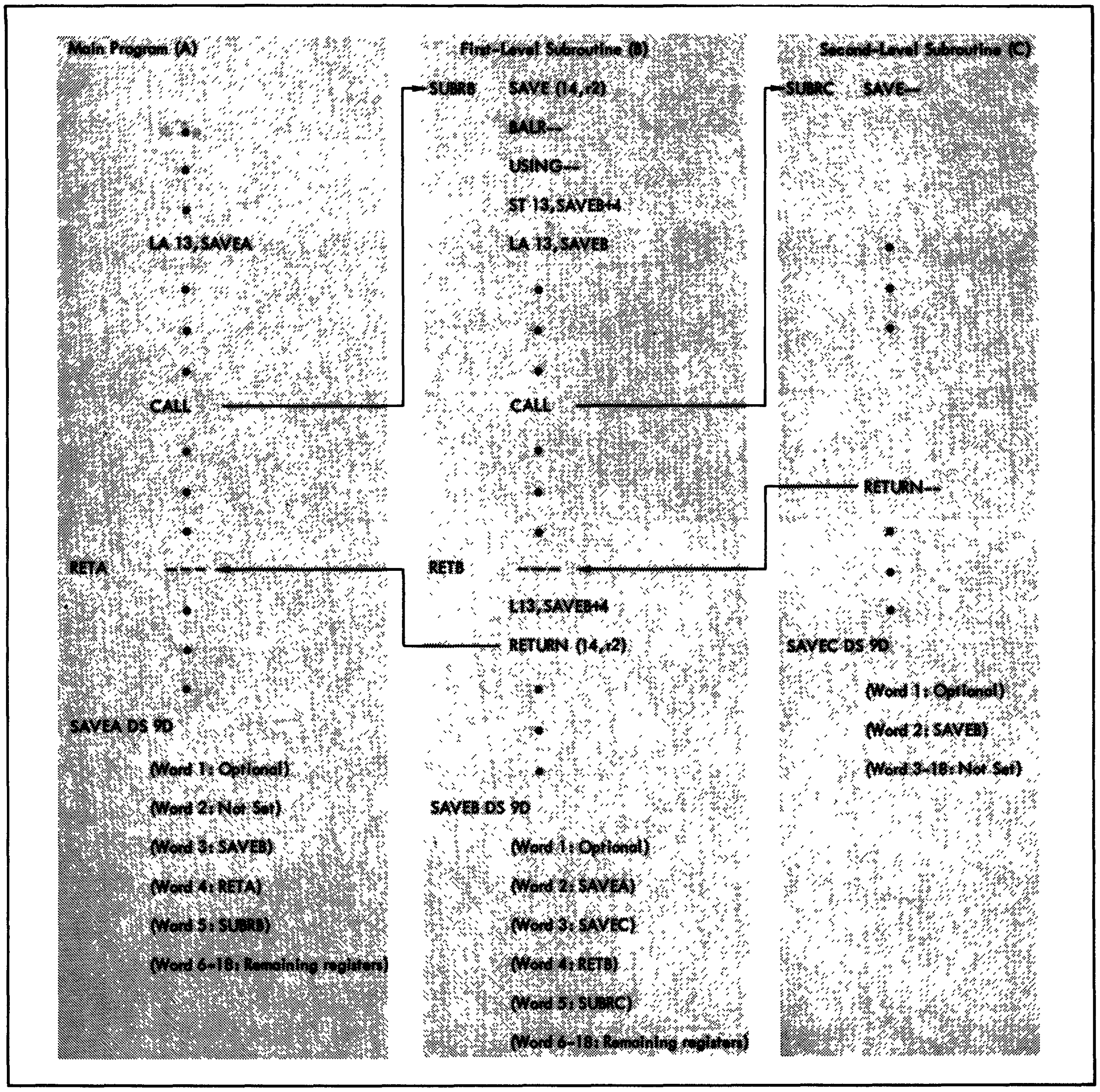


Figure 7-7 **Direct linkage**

Linkage Registers

To standardize branching and linking, registers are assigned specific roles (see Figure 7-8). 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
para                           meter
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 the 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 BALR.

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 your 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 7-8. Linkage registers

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 have been changed.
3. The parameter list addresses contain 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 storage because the instruction to save and restore registers need not be repeated 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 aligned on a doubleword boundary. For programs to save registers in a uniform manner, the save area has a standard format shown in Figure 7-9 and described below:

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 7-9 **Save area words and contents in calling programs**

- **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 it loads register 13 with the address of the current save area that is passed to a lower level routine (Figure 7-9, 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 Macro

The CALL macro passes control from a program to a specified entry point in another program. The program issuing the CALL macro is the calling program. The program receiving control is the called program or routine. The called program must be in virtual storage when the CALL macro is executed. The called program is brought into virtual storage in one of two ways:

1. As part of the program issuing the CALL. In this case, the CALL macro must specify an entry point by symbolic name. The linkage editor includes the phase containing that entry point in the phase containing the CALL macro.

2. As the phase specified by a LOAD macro. In this case, the CALL macro specifies register 15 (the entry-point register) into which the address of the program to be called was loaded. The LOAD macro must precede the first CALL for that program.

The format of the CALL macro is shown below.

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 specified, 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 a symbolic name is specified for the entry point operand, the called program resides in storage throughout execution of the calling program. This wastes storage if the called program is not needed throughout execution of the calling program.

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, in which case the V-type address constant instruction is not generated. Control is given to the called program at the address in register 15. Specifying register 15 preceded by a LOAD macro is most useful when the same program is called many times during execution of the calling program, but is not needed in storage throughout execution of the calling program.

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 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. When the called program is entered, register 1 (the parameter list register) contains the address of the 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 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. 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 Macro

The SAVE macro stores the contents of specified registers in the save area 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 machine 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 Macro

The RETURN macro restores the registers whose contents were saved and returns control to the calling program.

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 machine instruction, the operands cause the desired 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.

APPENDIX A: CONTROL CHARACTER CODES

CTLCHR=ASA

If the ASA option is chosen, a control character must appear in each record. If the control character for the printer is not valid, a message is given and the job is canceled. If the control character for card devices other than the 2560 and 5425 is not V or W, the card is selected into stacker 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
X	Select stacker 3 (2560 and 5425 DTFCD files only)
Y	Select stacker 4 (2560 and 5425 DTFCD files only
Z	Select stacker 5 (2560 DTFCD files only)
For DTFDI files on 2560 and 5425	
V	Primary hopper: select stacker 1
W	Primary hopper: select stacker 2
V	Secondary hopper: select stacker 5 (on 2560)
V	Secondary hopper: select stacker 4 (on 5425)
W	Secondary hopper: select stacker 3

* For 3525 print (not associated) files, either space one or skip to channel 1 must be used to print on the first line of a card. For 3525 print associated files, only space one must be used to print on the first line of a card.

CTLCHR=YES

The control character for this option is the command-code portion of the CCW used in printing a line or spacing the forms. If the character is not one of the following characters, unpredictable events will occur.

Hexa-decimal Code	Punch Combina-tion	Function
Stacker Selection on 1442 and 2596		
81	12,0,1	Select into stacker 1
C1	12,1	Select into stacker 2
Stacker Selection on 2520		
01	12,9,1	Select into stacker 1
41	12,0,9,1	Select into stacker 2
Stacker Selection on 2540		
01	12,9,1	Select into stacker 1
41	12,0,9,1	Select into stacker 2
81	12,0,1	Select into stacker 3

Hexa decimal Code	Punch Combina-tion	Function
Stacker Selection on 2560 and 5425		
13	11,3,9	Primary hopper: select into stacker 1
23	0,3,9	Primary hopper: select into stacker 2
33	3,9	Primary hopper: select into stacker 3
43	12,0,3,9	Primary hopper: select into stacker 4
53	12,11,3,9	Primary hopper: select into stacker 5 (2560 only)
93	12,11,3	Secondary hopper: select into stacker 1
A3	11,0,3	Secondary hopper: select into stacker 2
B3	12,11,0,3	Secondary hopper: select into stacker 3
C3	12,3	Secondary hopper: select into stacker 4
D3	11,3	Secondary hopper: select into stacker 5 (2560 only)
Stacker Selection on 3504, 3505 and 3525		
01	12,9,1	Select into stacker 1
41	12,0,9,1	Select into stacker 2
Printer Control (Except for 3525)		
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

Hexa-decimal Code	Punch Combina-tion	Function
Printer Control (Except for 3525)		
89	12,0,9	Write and skip to channel 1 after printing
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 Combina-tion	Function
Printer Control (Except for 3525)		
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

Printer Control for 3525 with Print Feature

Hexa-decimal Code	Punch Combina-tion	Function
0D	12,5,8,9	Print on line 1
15	11,5,9	Print on line 2
1D	11,5,8,9	Print on line 3
25	0,5,9	Print on line 4
2D	0,5,8,9	Print on line 5
35	5,9	Print on line 6
3D	5,8,9	Print on line 7
45	12,0,5,9	Print on line 8
4D	12,5,8	Print on line 9
55	12,11,5,9	Print on line 10
5D	11,5,8	Print on line 11
65	11,0,5,9	Print on line 12
6D	0,5,8	Print on line 13
75	12,11,0, 5,9	Print on line 14
7D	5,8	Print on line 15
85	12,0,5	Print on line 16
8D	12,0,5,8	Print on line 17
95	12,11,5	Print on line 18
9D	12,11,5,8	Print on line 19
A5	11,0,5	Print on line 20
AD	11,0,5,8	Print on line 21
B5	12,11,0,5	Print on line 22
BD	12,11,0 5,8	Print on line 23
C5	12,5	Print on line 24
CD	12,0,5,8, 9	Print on line 25

APPENDIX B: ASSEMBLING YOUR 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: 2540 (SYS004).
2. Disk: 3330 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 work area 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 your 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 your 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 YOUR PROGRAM, DTFs, AND LOGIC MODULES TOGETHER

Figure B-1 shows the assembly of the DTFs, logic modules, and your program. The assembly source deck is:

```
CDTODISK  START     0                     Col. 72
          BALR      12,0
          USING     *,12
          LA        13,SAVEAREA              Initialize base register
          OPEN      CARDS,DISK               Establish addressability.
NEXT      GET       CARDS,(2)                Use reg 13 as pointer to save area.
          PUT       DISK                     Open both files.
          B         NEXT                     Read one card and move it
SAVEAREA  DS        9D                       to the disk output buffer.
                                             Return for next card.
EOFCD     CLOSE     CARDS,DISK               Save area is 72-byte, doubleword
          EOJ                                aligned.

                                             At card-reader EOF, close
MYLABELS  .                                  both files and exit to job control.
          .
          LBRET     2                        Your label-processing routine.
                                             Return to main program.
CARDS     DTFCD
                    DEVADDR=SYS004,     X
                    EOFADDR=EOFCD,      X
                    IOAREA1=A1,         X
                    WORKA=YES           X

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      X
                    DEVICE=3330         X

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
```

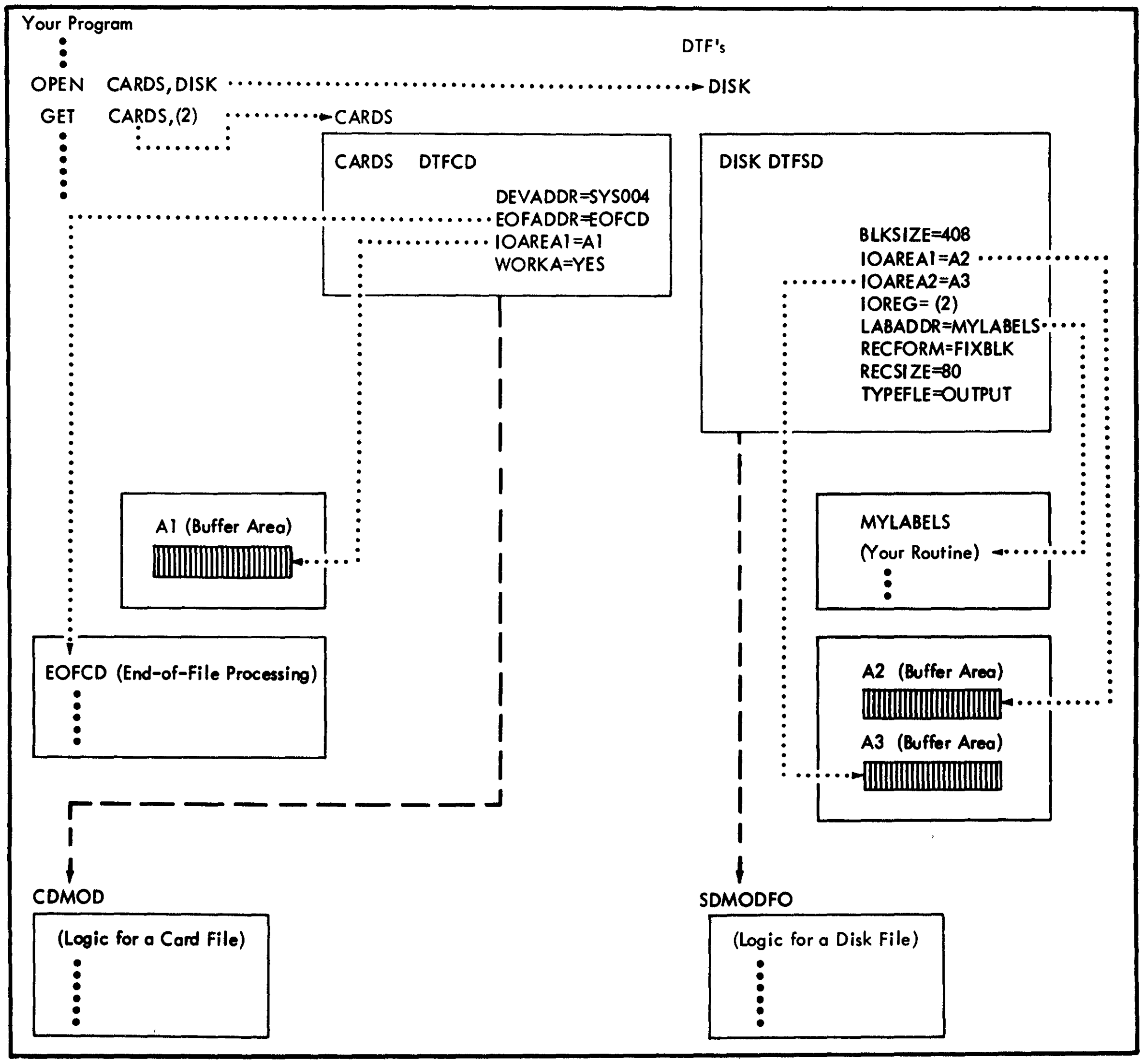


Figure B-1. Assembling Your Program DTFs and Modules Together (Example 1)

EXAMPLE 2: ASSEMBLING THE LOGIC MODULES SEPARATELY

The main-program source deck is identical to that in Example 1 until (1); at this point, you simply furnish the END card. Figure B-2 shows the separation of the I/O logic modules.

The two logic modules are assembled as follows:

```
                                                Col. 72
             CDMOD                                   X
                            DEVICE=2540,             X
Card logic                  SEPASMB=YES,             X
module                      TYPEFLE=INPUT,           X
                            WORKA=YES
             END

             SDMODFO                                 X
Disk logic                  SEPASMB=YES
module
             END
```

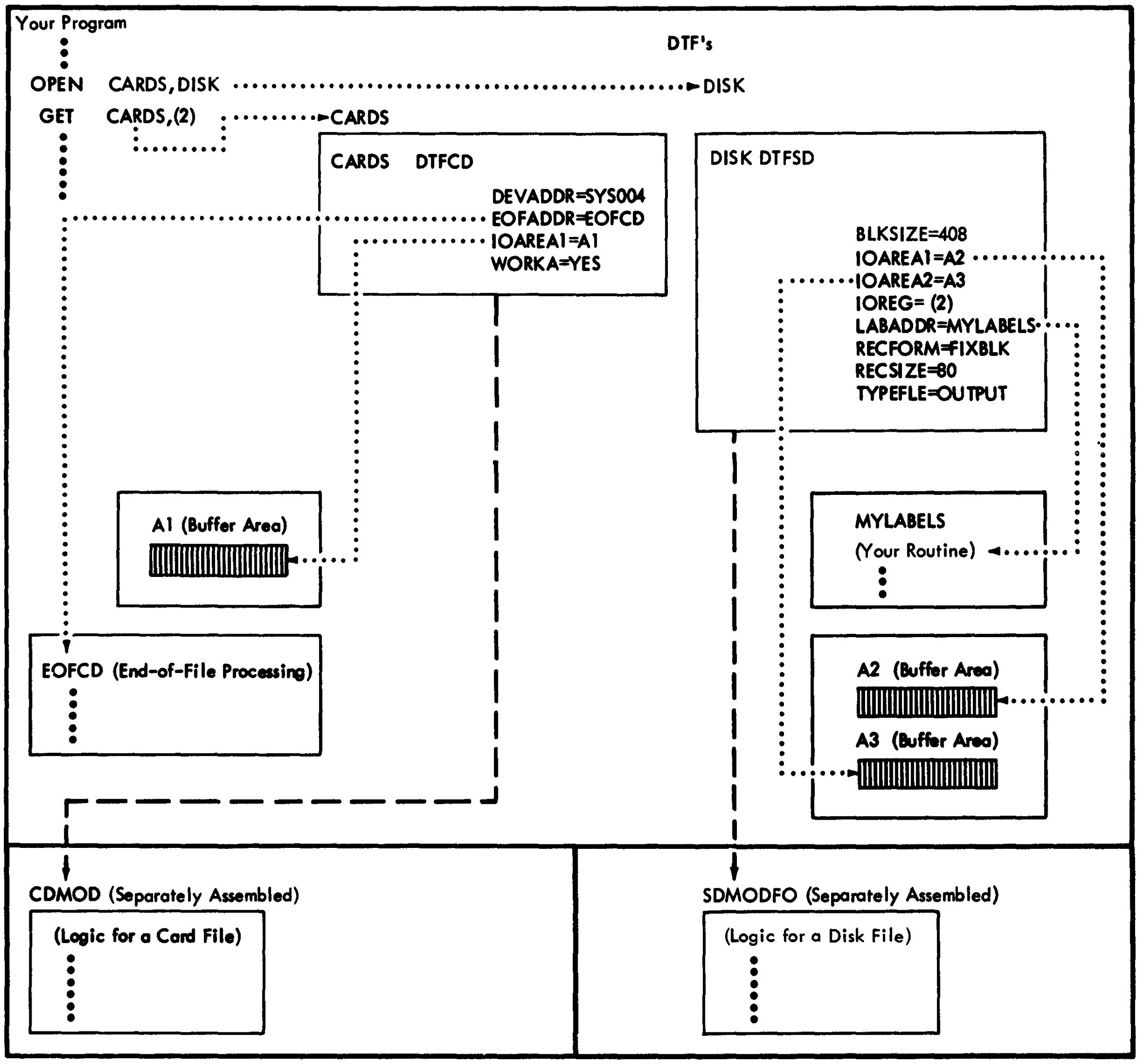


Figure B-2. Logic Modules Assembled Separately (Example 2)

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 your 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 B-3 shows the separation of the DTFs and logic modules.

The DTFCD 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 DTFCD and related functions are assembled:

```
                                        Col. 72
     DISK      DTFSD                       X
                      BLKSIZE=408,         X
                      SEPASMB=YES,         X
                      .
                      .
                      .
                      TYPEFLE=OUTPUT X
                      DEVICE=3330
     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 routines are segregated from the main program and moved into the DTF assembly.

When your routines, such as error, label processing, or EOF routines, are segregated from the main progrm, it is necessary to establish addressability for these routines. You 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 you segregate your error routines, it is your responsibility to establish addressability for them.

Figure B-4 contains the printer output. to show the coding of Example 3 would look when assembled.

In Figure B-4, the standard name was generated for the logic modules: statement 13 of the DTFCD--V(IJCFZIW0), 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.

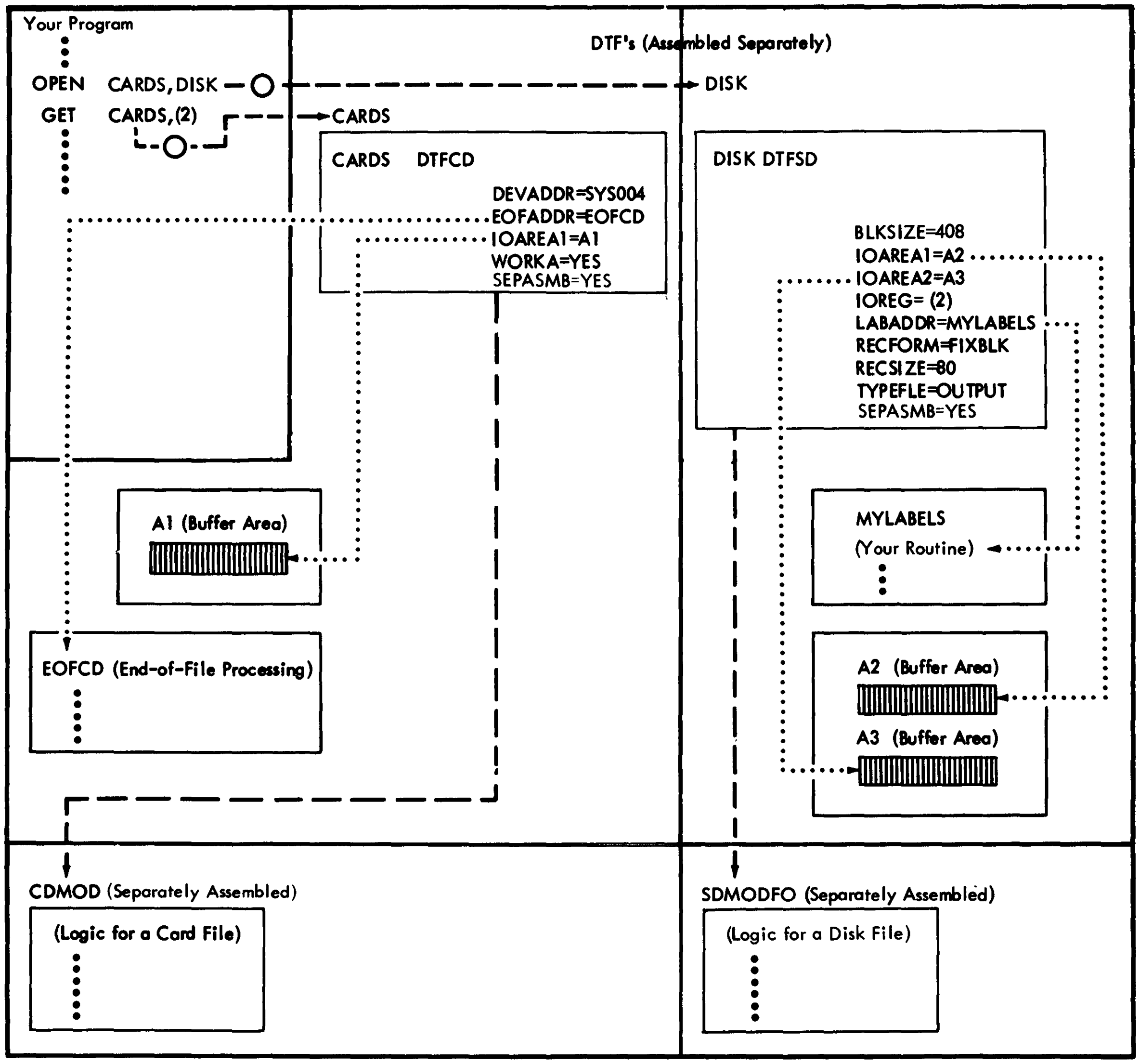


Figure B-3. Logic Modules and DTFs Assembled Separately (Example 3)

```
MAIN PROGRAM
                                          EXTERNAL SYMBOL DICTIONARY                              PAGE    1

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. }  Defined by EXTRN statement.
DISK      ER   03                     External reference. }
```

```
                                                EXAMPLE 3                                           PAGE    1

  LOC  OBJECT CODE    ADDR1 ADDR2   STMT   SOURCE STATEMENT                          DOS/VS ASSEMBLER V 28.0 09.44 73-05-16

000000                                 1 CDTODISK    START   0
000000 05C0                            2             BALR    12,0            INITIALIZE BASE REGISTER
000002                                 3             USING   *,12            ESTABLISH ADDRESSABILITY
000002 41D0 C03E            00040      4             LA      13,SAVEAREA     USE REGISTER 13 AS POINTER TO SAVE
                                       5 * OPEN THE FILE
                                       6             OPEN    CARDS,DISK      OPEN BOTH FILES
                                       7+* IOCS - OPEN - 5745-SC-IOX - REL. 28.0
000006 0700                            8+         CNOP  0,4
000008                                 9+         DC    OF'0'
000008 4110 C086            00088     10+         LA    1,=C'$$BOPEN '
00000C 1BFF                           11+         SR    15,15 ZERO R15 FOR ERROR RETURN    5-0
00000E 0700                           12+         NOPR  0 WORD ALIGNMENT                   5-0
000010 4500 C01A            0001C     13+IJJ00001 BAL   0,*+4+4*(3-1)
000014 00000000                       14+         DC    A(CARDS)
000018 00000000                       15+         DC    A(DISK)
00001C 0A02                           16+         SVC   2
                                      17 NEXT        GET     CARDS,(2)       READ ONE CARD, MOVE TO WORK AREA
                                      18+* IOCS AND DEVICE INDEPENDENT I/O - GET - 5745-SC-IOX - REL. 28.0
00001E 5810 C08E            00090     19+NEXT     L     1,=A(CARDS) GET DTF TABLE ADDRESS
000022 1802                           20+         LR    0,2 GET WORK AREA ADDRESS
000024 58F1 0010            00010     21+         L     15,16(1) GET LOGIC MODULE ADDRESS
000028 45EF 0008            00008     22+         BAL   14,8(15) BRANCH TO GET ROUTINE
                                      23             PUT     DISK            WRITE ON DISK
                                      24+* IOCS AND DEVICE INDEPENDENT I/O - PUT - 5745-SC-IOX - REL. 28.0
00002C 5810 C092            00094     25+         L     1,=A(DISK) GET DTF TABLE ADDRESS
000030 58F1 0010            00010     26+         L     15,16(1) GET LOGIC MODULE ADDRESS     3-5
000034 45EF 000C            0000C     27+         BAL   14,12(15) BRANCH TO PUT ROUTINE        3-5
000038 47F0 C01C            0001E     28             B       NEXT            GO FOR NEXT CARD
000040                                29 SAVEAREA    DS      9D              72-BYTE SAVE AREA
                                      30             EXTRN   CARDS,DISK
000000                                31             END     CDTODISK
000088 5B5BC2D6D7C5D540               32                =C'$$BOPEN '
000090 00000000                       33                =A(CARDS)
000094 00000000                       34                =A(DISK)
```

Figure B-4. 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.
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.
```

```
                                                                                                  PAGE    1

                                              EXAMPLE 3                  DOS/VS ASSEMBLER V 28.0 09.44 73-05-16

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                    1 CARDS       DTFCD  DEVADDR=SYS004,                                  X
                                                         SEPASMB=YES,                                     X
                                                         EOFADDR=EOFCD,                                   X
                                                         IOAREA1=A1,                                      X
                                                         WORKA=YES
                                    2+* IOCS AND DEVICE INDEPENDENT I/O - DTFCD - 5745-SC-IOX - REL. 28.0
                                    3+          PUNCH '   CATALR CARDS,4.0' 4-0
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) SWITCH 3                          4-0
000011 000000                      13+          DC    VL3(IJCFZIWO) ADDRESS OF LOGIC MODULE   3-3
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 0000004C                    18+          DC    A(A1) ADDR. OF IOAREA1
00001C 00                          19+          DC    AL1(0) JJ
00001D 00000034                    20+          DC    AL3(EOFCD) EOF ADDRESS                            JJ
000020 0200004C20000050            21+IJCX0001 CCW    2,A1,X'20',80
000028 4700 0000          00000    22+          NOP   0 LOAD USER POINTER REG.
00002C D24F D000 E000 00000 00000  23+          MVC   0(80,13),0(14) MOVE IOAREA TO WORKA
000032                             24+IJJZ0001 EQU    *
000032                             25              USING   *,14            ESTABLISH ADDRESSABILITY
                                   26 * CLOSE THE FILE
                                   27 EOFCD        CLOSE   CARDS,DISK      END OF FILE ADDRESS FOR CARD READER
                                   28+* IOCS - CLOSE - 5745-SC-IOX - REL. 28.0
000032 0700                        29+          CNOP  0,4
000034                             30+EOFCD     DC    0F'0'
000034 4110 E06E          000A0    31+          LA    1,=C'$$BCLOSE'
000038 1BFF                        32+          SR  , 15,15 ZERO R 15 FOR ERROR RETURN   5-0
00003A 0700                        33+          NOPR  0 WORD ALIGNMENT                 5-0
00003C 4500 E016          00048    34+IJJC0002 BAL    0,*+4+4*(3-1)
000040 00000000                    35+          DC    A(CARDS)
000044 00000000                    36+          DC    A(DISK)
000048 0A02                        37+          SVC   2
                                   38              EOJ
                                   39+* SUPVR COMMN MACROS - EOJ - 5745-SC-SUP - REL. 28.0
00004A 0A0E                        40+          SVC   14
                                   41              EXTRN   DISK
00004C                             42 A1           DS      80C            CARD I/O AREA
                                   43              END
0000A0 5B5BC2C3D3D3D6E2C5          44                 =C'$$BCLOSE'
```

Figure B-4. 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.
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                DOS/VS ASSEMBLER V 28.0 09.44 73-05-16

                                    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,                                  X
                                                      DEVICE=3330
                                    2+* SEQUENTIAL DISK IOCS - DTFSD - 5745-SC-DSK - REL. 28.0
                                    3+         PUNCH '   CATALR DISK,4.0' 4-0
000000                              4+DISKC    CSECT
                                    5+         ENTRY DISK
000000                              6+         DC    OD'0'
000000 000080040000                 7+DISK     DC    X'000080040000' 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) 3-3
000011 000000                      13+         DC    VL3(IJGFOZZZ) 3-8
000014 20                          14+         DC    X'20' DTF TYPE
000015 49                          15+         DC    AL1(73) OPEN/CLOSE INDICATORS
000016 C4C9E2D2404040              16+         DC    CL7'DISK' FILENAME
00001D 04                          17+         DC    X'04' INDICATE 3330            4-0
00001E 000000000000                18+         DC    6X'00' BCCHHR ADDR OF F1 LABEL IN VTOC
000024 0000                        19+         DC    2X'00' VOL SEQ NUMBER
000026 08                          20+         DC    X'80' OPEN COMMUNICATIONS BYTE
000027 00                          21+         DC    X'00' XTENT SEQ NO OF CURRENT EXTENT
000028 00                          22+         DC    X'00' XTENT SEQ NO LAST XTENT OPENED
000029 0000A0                      23+         DC    AL3(MYLABELS) USER'S LABEL ADDRESS
00002C 000000A4                    24+         DC    A(A2) ADDRESS OF IOAREA        4-0
000030 80000000                    25+         DC    X'80000000' CCHH ADDR OF USER LABEL TRACK
000034 0000                        26+         DC    2X'00' LOWER HEAD LIMIT
000036 00000000                    27+         DC    4X'00' XTENT UPPER LIMIT
00003A 0000                        28+DISKS    DC    2X'00' SEEK ADDRESS-BB
00003C 0000FF00                    29+         DC    X'0000FF00' SEARCH ADDRESS-CCHH
000040 00                          30+         DC    X'00' RECORD NUMBER
000041 00                          31+         DC    X'00' KEY LENGTH
000042 0190                        32+         DC    H'400' DATA LENGTH
000044 00000000                    33+         DC    4X'00' CCHH CONTROL FIELD
000048 17                          34+         DC    AL1(23) R    CONTROL FIELD
000049 04                          35+         DC    B'00000100' 3-2
00004A 018F                        36+         DC    H'399' SIZE OF BLOCK-1
00004C FFFFFFFFFF                  37+         DC    5X'FF' CCHHR BUCKET            3-7
000051 00                          38+         DC    X'00'
000052 32E6                        39+         DC    H'13030' TRACK CAPACITY CONSTANT
000054 5821 0058         00058     40+         L     2,88(1) LOAD USER'S IOREG
000058 000000AC                    41+         DC    A(A2+8) DEBLOCKER-INITIAL POINTER    4-0
00005C 00000050                    42+         DC    F'80' DEBLOCKER-RECORD SIZE
000060 0000023B                    43+         DC    A(A2+8+400-1) DEBLOCKER LIMIT              3-9
000064 0A                          44+         DC    AL1(10) LOGICAL INDICATORS
000065 000000                      45+         DC    AL3(0) USER'S ERROR ROUTINE
000068 0700003A40000006            46+IJGC0001 CCW   7,*-46,64,6 SEEK
000070 3100003C40000005            47+         CCW   X'31',*-52,64,5 SEARCH ID EQUAL
000078 0800007000000000            48+         CCW   8,*-8,0,0 TIC
```

Figure B-4. Separate Assemblies, (Example 3) (Part 3 of 4)

DTFSD (Continued)

```
                                            EXAMPLE 3                                          PAGE    2

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                        DOS/VS ASSEMBLER V 28.0 09.44 73-05-16
00
000080 1D00023C00000198             49+          CCW   X'1D',A3,0,400+8 WRITE COUNT KEY AND DATA
000088 3100003C40000005             50+          CCW   X'31',DISKS+2,64,5 SEARCH ID EQUAL
000090 0800008800000000             51+          CCW   8,*-8,0,0 TIC
000098 1E00009830000001             52+          CCW   30,*,48,1 VERIFY
0000A0                              53+IJJZ0001 EQU   *
0000A0 05A0                         54 MYLABELS  BALR    10,0                     INITIALIZE BASE REGISTER
0000A2                              55           USING   *,10                     ESTABLISH ADDRESSABILITY
                                    56 *                                          USER'S LABEL               *
                                    57 *                                          PROCESSING ROUTINE         *
                                    58                  LBRET 2                   RETURN TO LIOCS
                                    59+* IOCS - LBRET - 5745-SC-IOX - REL. 28.0
0000A2 0A09                         60+          SVC   9 BRANCH BACK TO IOCS
0000A4                              61 A2        DS      408C                     FIRST DISK I/O AREA
00023C                              62 A3        DS      408C                     SECOND DISK I/O AREA
                                    63           END
```

CDMOD ASSEMBLY

```
                                         EXTERNAL SYMBOL DICTIONARY
  SYMBOL  TYPE ID  ADDR  LENGTH LD ID

IJCFZIWO  SD  01 000000 000060
```
Section definition. CSECT name generated by CDMOD macro.

```
                                            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.

```
                                           'EXAMPLE 3

  LOC  OBJECT CODE    ADDR1 ADDR2 STMT   SOURCE STATEMENT

                                   2           PRINT NOGEN
                                   3           SDMODFO                                         X
                                                     SEPASMB=YES
                                 169           END
```

Figure B-4. 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 appropiate CATALR card. These two object decks can be cataloged as follows 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 B-5 shows the separation of the logic modules, DTFs and I/O areas.

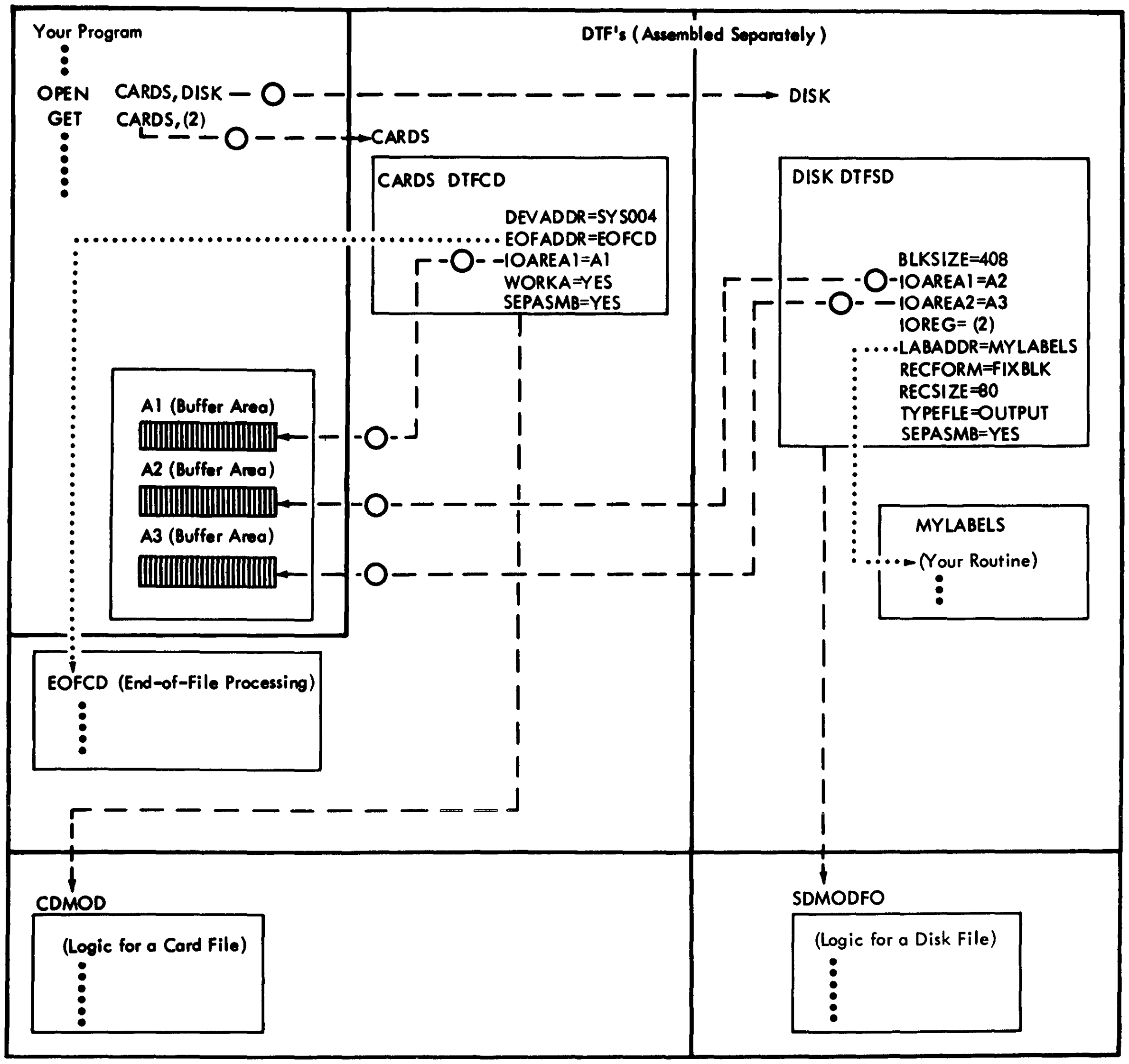


Figure B-5. Logic Modules and DTFs Assembled Separately, I/O Areas with Main Program (Example 4)

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 B-6 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
```

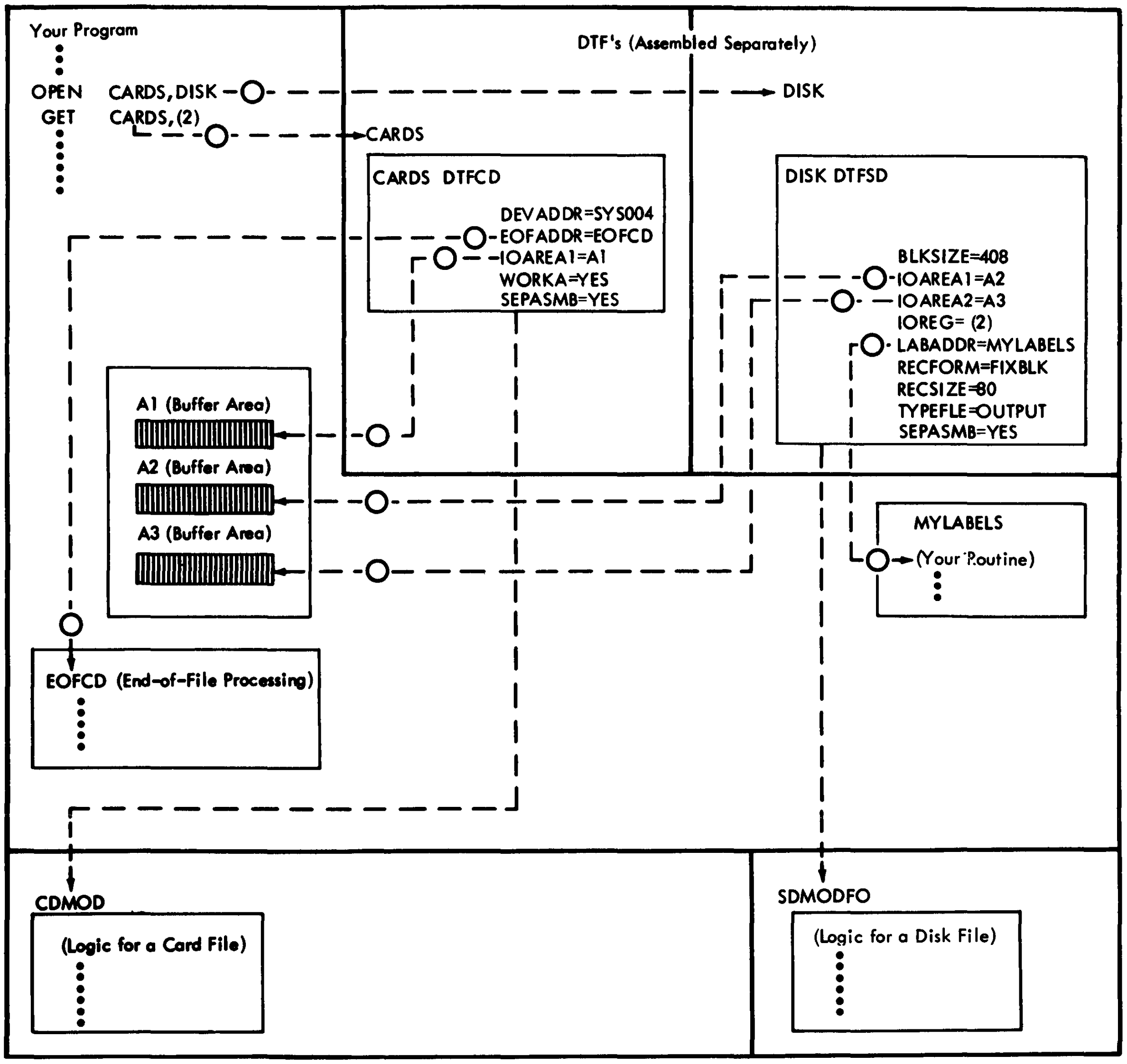


Figure B-6. DTFs, and Logic Modules Assembled Separately; I/O Areas, Label Exit, EOF Exit with Main Program (Example 5)

The file definitions 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
              .
              .
              .
              .
              ICAREA1=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 the Five Methods

Example 1 requires the most assembly time and the least link-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 macros OPEN, OPENR, CLOSE, CLOSER, GET, and PUT remain. 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, you need only be concerned with the record format and the general register pointing to the record. You can virtually ignore the operands BLKSIZE, LABADDR, etc. in your program, although you must ultimately consider their effect on virtual 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 B.1: ASSEMBLING A FORMAT RECORD FOR THE 3886 OPTICAL CHARACTER READER

This section describes a use for the IBM 3886 Optical Character Reader. Included are a sample document, a format record assembly and the data provided by the 3886.

Document Example

A typical application for an optical character reader is processing insurance premiums. Figure B-7 shows an insurance premium notice for the Standardacme Life Insurance Company. The document has three lines of data to be read (see Figure B-4 for sample data). The first line contains one field, the name of the policy holder. The second line contains four fields: the second line of the policyholder's address, the policy number, the premium amount due and a code to be hand printed if the amount paid is different from the amount due. The third line contains one field that contains the amount paid if different from the amount due.

Format Record Assembly Example

To process documents like that in Figure B-7, one format record is used. The format record must be created in a separate assembly. The coding necessary to create the format record is shown in Figure B-8. The numbers at the left of the coding form correspond to those in the following text.

1 The job control language (JCL) statements indicate that the job is an assembly. The output of the assembly is to be cataloged with the phase name FORMAT.

2 The DFR macro specifies the characteristics common to all lines on the document:

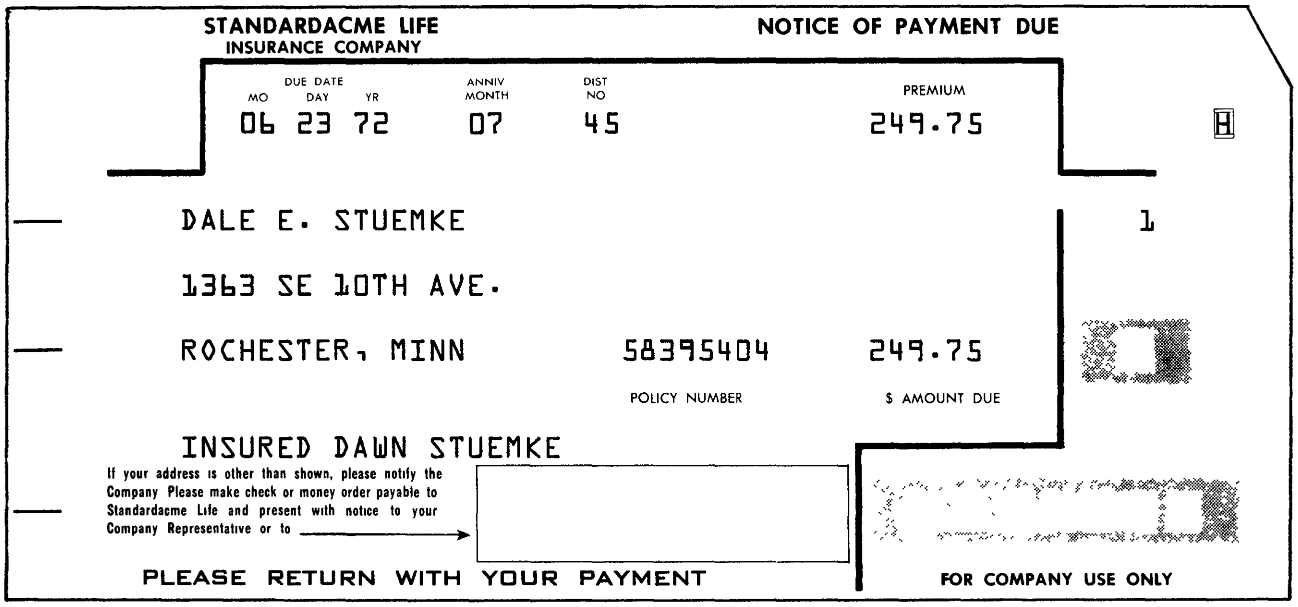
FONT=ANA1: The alphameric OCR-A font is used for reading any fields that do not have another font specified in the DLINT macro field entries.

REJECT=@: The commercial at sign (@) is substituted for any reject characters encountered.

EDCHAR=(',',.): The comma and period are removed from one or more fields as indicated in DLINT entries (line 2, field 3).

3 The DLINT macro describes one line type in a format record described by the DFR macro. The following information is provided about the first line:

LFR=1,LINBEG=4: The first line on the document has a line format record number of 1. The first field read from the line begins four-tenths of an inch from the



STANDARDACME LIFE INSURANCE COMPANY

NOTICE OF PAYMENT DUE

DUE DATE MO	DAY	YR	ANNIV MONTH	DIST NO	PREMIUM
06	23	72	07	45	249.75

H

DALE E. STUEMKE

1

1363 SE 10TH AVE.

ROCHESTER, MINN   58395404   249.75

POLICY NUMBER $ AMOUNT DUE

INSURED DAWN STUEMKE

If your address is other than shown, please notify the Company Please make check or money order payable to Standardacme Life and present with notice to your Company Representative or to

PLEASE RETURN WITH YOUR PAYMENT

FOR COMPANY USE ONLY

Figure B-7. Premium Notice Example

left edge of the document. The data record is in the standard mode; editing is performed on the field.

FLD1=(32,20,NCRIT),EDIT1=HLBLOF: The first and only field on the line ends 3.2 inches from the left edge of the document, the edited data is placed in a 20-character field. The field is not considered critical. All leading and trailing blanks are removed, the data should be left-justified, and the field is padded to the right with blanks.

The second line on the document is described as follows:

LFR=2,LINBEG=4: The second line of the document has a line format record number of 2. The first field read begins four-tenths of an inch from the left edge of the document. The data record is in standard mode; editing is performed on all fields on the line.

FLD1=(30,20,NCRIT),EDIT1=HLBLOF: The first field on the line ends 3.0 inches from the left edge of the document, the edited data is placed in a 20 byte field. The field is not considered critical. All leading and trailing blanks are removed, the data is left-justified, and the field is padded to the right with blanks.

FLD2=(42,8),EDIT2=ALBNOF: The second field ends 4.2 inches from the left edge of the document, the edited data is placed in an eight-byte field, the field is critical. All leading and trailing blanks are removed from the field. The resulting field must be eight digits in length or a wrong length field indicator is set.

FLD3=(54,6),EDIT3=HLBHIF,EDCHAR: The third field ends 5.4 inches from the left edge of the document, the edited data is placed in a six-byte field, the field is critical. All leading and trailing blanks are removed, the data is right-justified, and the field is padded to the left with zeros. A comma, if present, and the decimal point are removed from the edited field.

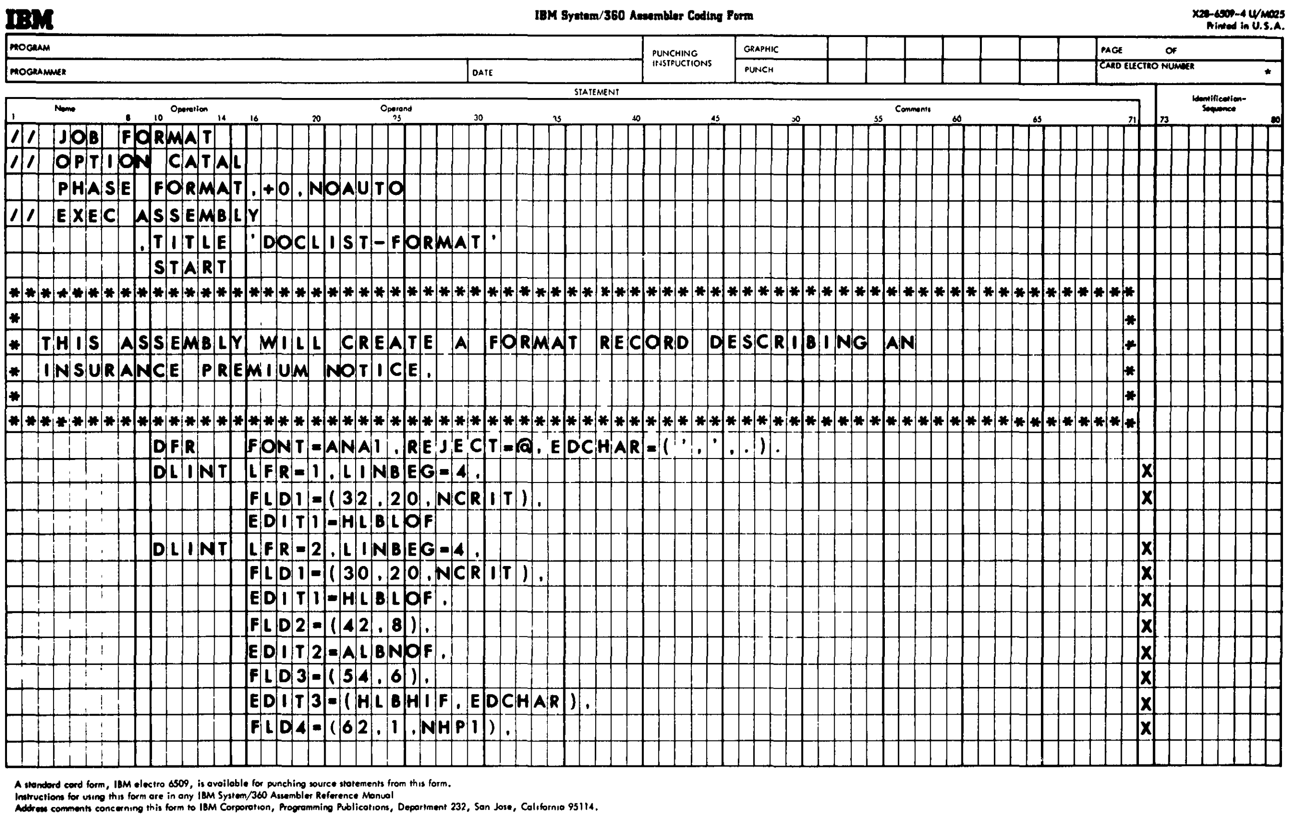
FLD4=(62,1,HHP1),EDIT4=ALBHIF: The fourth field ends 6.2 inches from the left edge of the document, the edited data is placed in a one-byte field, the field is critical and is read using the numeric handprinting normal mode. All blanks are removed, the data is right-justified, and the field is padded to the left with zeros.

The third line on the document is described as follows:

LFR=3,LINBEG=45: The third line on the document has a line format record number of 3. The field to be read begins 4.5 inches from the left edge of the document. The data record is in standard mode; editing is performed.

FLD1=(63,7,NHP1),EDIT1=ALBHIF: The field on this line ends 6.3 inches from the left edge of the document, the edited data is placed in a seven-byte field, the field is critical and is read using the numeric handprinting normal mode. All blanks are removed, the data is right-justified, and the field is padded to the left with zeros.

FREND=YES: This is the format record end. No DLINT macros follow this statement.



IBM **IBM System/360 Assembler Coding Form** X28-6509-4 U/M025 Printed in U.S.A.

PROGRAM		PUNCHING INSTRUCTIONS	GRAPHIC
PROGRAMMER	DATE		PUNCH

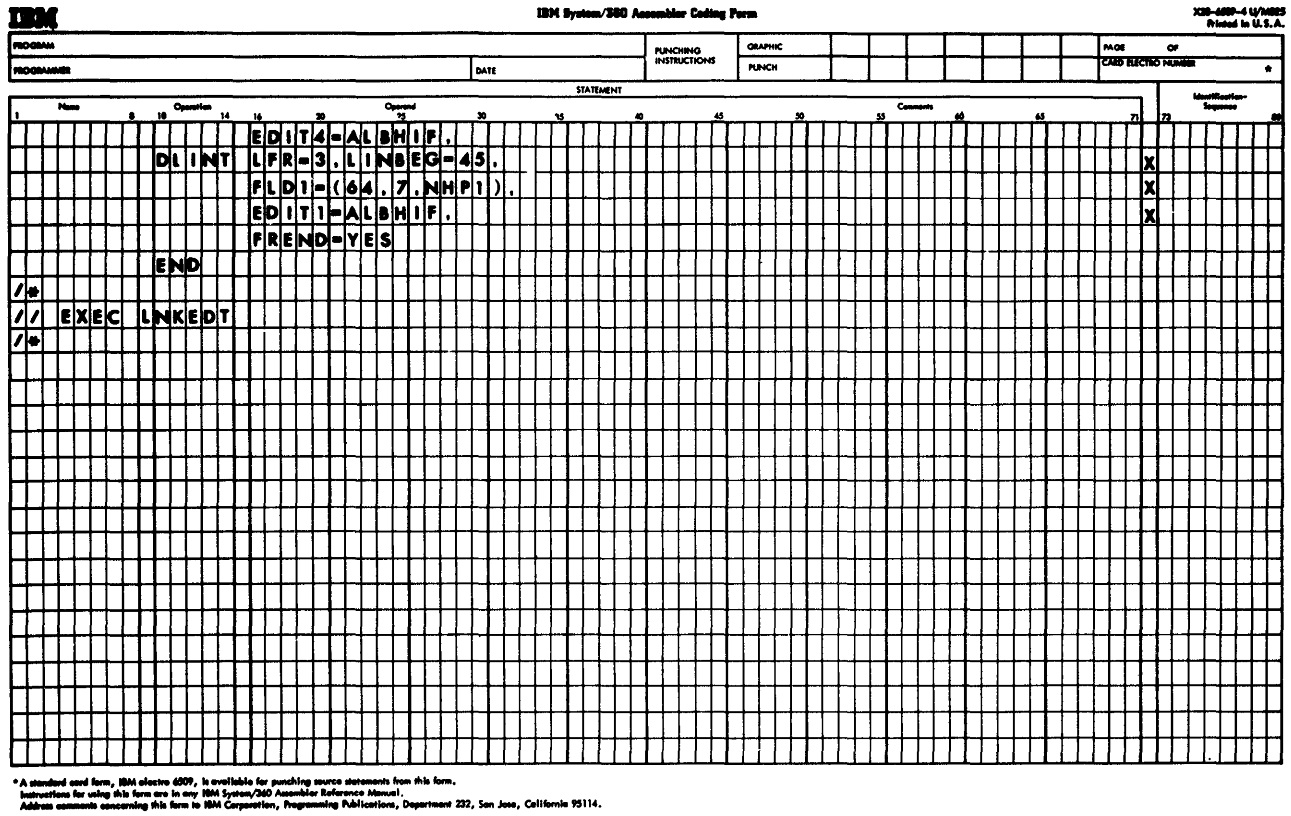
PAGE OF

CARD ELECTRO NUMBER

```
// JOB FORMAT
// OPTION CATAL
   PHASE FORMAT,+0,NOAUTO
// EXEC ASSEMBLY
        .TITLE 'DOCLIST-FORMAT'
         START
**********************************************************************
*                                                                    *
* THIS ASSEMBLY WILL CREATE A FORMAT RECORD DESCRIBING AN            *
* INSURANCE PREMIUM NOTICE.                                          *
*                                                                    *
**********************************************************************
         DFR   FONT=ANA1,REJECT=@,EDCHAR=(',',.).
         DLINT LFR=1,LINBEG=4,                                         X
               FLD1=(32,20,NCRIT),                                     X
               EDIT1=HLBLOF
         DLINT LFR=2,LINBEG=4,                                         X
               FLD1=(30,20,NCRIT),                                     X
               EDIT1=HLBLOF,                                           X
               FLD2=(42,8),                                            X
               EDIT2=ALBNOF,                                           X
               FLD3=(54,6),                                            X
               EDIT3=(HLBHIF,EDCHAR),                                  X
               FLD4=(62,1,NHP1),                                       X
```

A standard card form, IBM electro 6509, is available for punching source statements from this form.
Instructions for using this form are in any IBM System/360 Assembler Reference Manual
Address comments concerning this form to IBM Corporation, Programming Publications, Department 232, San Jose, California 95114.

Figure B-8. Format Record Assembly Example (Part 1 of 2)


IBM System/360 Assembler Coding Form

X28-6509-4 U/M025
Printed in U.S.A.

PROGRAM		PUNCHING INSTRUCTIONS	GRAPHIC
PROGRAMMER	DATE		PUNCH

PAGE OF

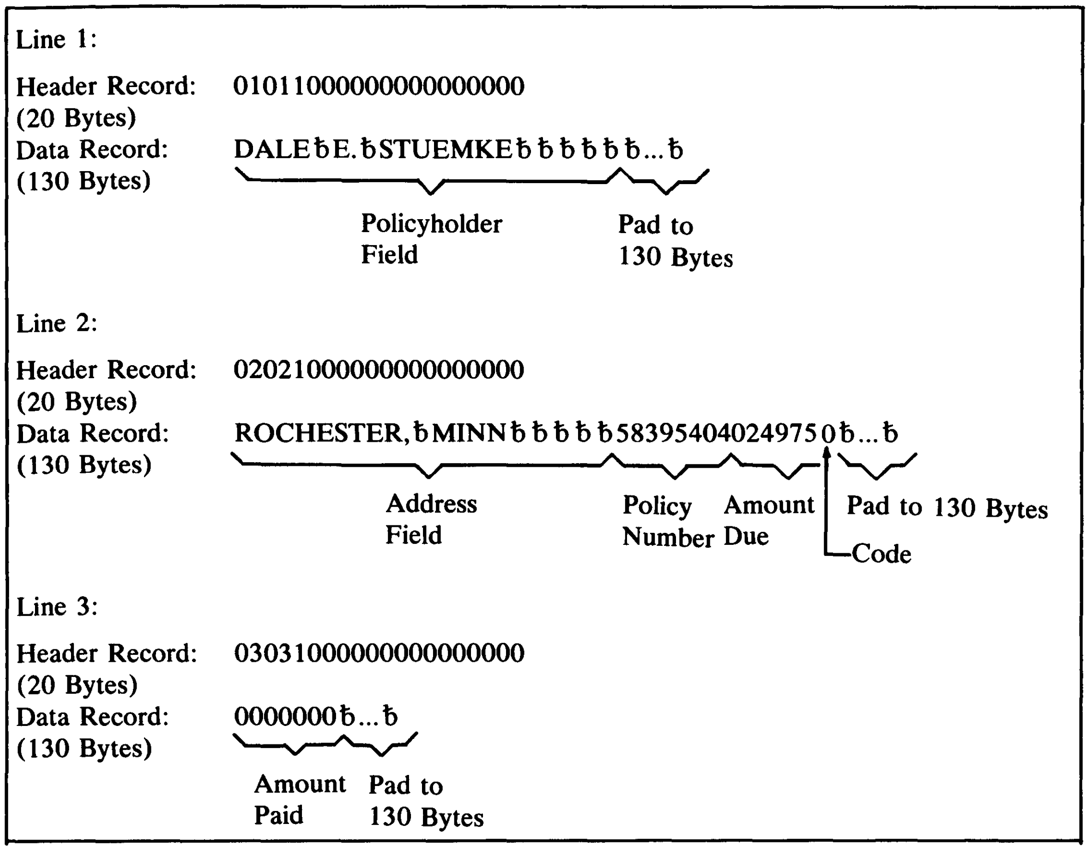
CARD ELECTRO NUMBER *

STATEMENT

```
               EDIT4=ALBHIF,
         DLINT LFR=3,LINBEG=45,                                        X
               FLD1=(64,7,NHP1),                                       X
               EDIT1=ALBHIF,                                           X
               FREND=YES
         END
/*
// EXEC LNKEDT
/*
```

*A standard card form, IBM electro 6509, is available for punching source statements from this form.
Instructions for using this form are in any IBM System/360 Assembler Reference Manual.
Address comments concerning this form to IBM Corporation, Programming Publications, Department 232, San Jose, California 95114.

Figure B-8. Format Record Assembly Example (Part 2 of 2)



Line 1:

Header Record: (20 Bytes) 01011000000000000000

Data Record: (130 Bytes) DALEƀE.ƀSTUEMKEƀƀƀƀƀƀ...ƀ

Line 2:

Header Record: (20 Bytes) 02021000000000000000

Data Record: (130 Bytes) ROCHESTER,ƀMINNƀƀƀƀƀ5839540402497 5 0ƀ...ƀ

Line 3:

Header Record: (20 Bytes) 03031000000000000000

Data Record: (130 Bytes) 0000000ƀ...ƀ

Figure B-9. Sample Data

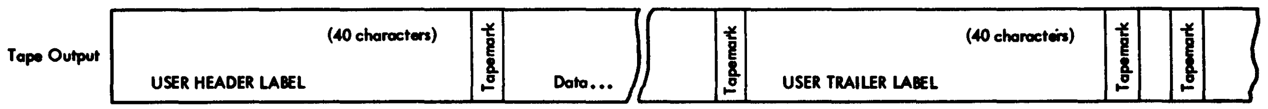
APPENDIX C: READING, WRITING, AND CHECKING WITH NONSTANDARD LABELS

```
                                                   EXTERNAL SYMBOL DICTIONARY                                      PAGE    1
 SYMBOL  TYPE ID  ADCR  LENGTH LD ID


          PC  01 003C00 0C048C
IJCFZIZO  ER  02
IJFFZZZZ  ER  03
IJFFBZZZ  ER  04
IJDFZZZZ  ER  05
IJ2L0006  SD  06 C03490 0CCC64
```

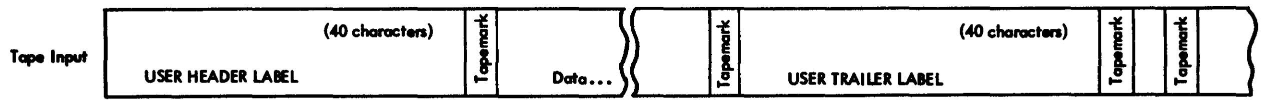
```
       TEST  CREATING AND PROCESSING NON-STANCARD LABELS                                                           PAGE.   1

  LOC  OBJECT CODE     ADDR1 ADCR2  STMT   SOURCE STATEMENT                          DOS/VS ASSEMBLER V 28.0 09.44 73-05-16

                                       2          PRINT ON,NOGEN,NODATA                                             NSTCOC04
003000                                 3          START 12288                                                       NSTD0005
                                       4 *                                                                          NSTC0006
                                       5 READER   DTFCD DEVICE=2540,DEVADDR=SYSIPT,BLKSIZE=80,TYPEFLE=INPUT,       *NSTD0007
                                                        EOFADDR=ENDCARD,IOAREA1=IOAREA                              NSTD0008
                                      26 *                                                                          NSTDOC09
                                      27 TAPEOUT  DTFMT DEVADDR=SYS004,IOAREA1=IOAREA,BLKSIZE=80,TYPEFLE=OUTPUT,*NSTD0010
                                                        LABADDR=LABELOUT,READ=FCRWARD,FILABL=NSTD                   NSTD0011
                                      58 *                                                                          NSTD0012
                                      59 TAPEIN  ·DTFMT DEVADDR=SYS004,IOAREA1=IOAREA,BLKSIZE=80,TYPEFLE=INPUT,    *NSTDOC13
                                                        EOFADDR=ENDTAPE,READ=FCRWARD,FILABL=NSTD,REWIND=NORWD,     *NSTD0014
                                                        LABADDR=LABELIN                                             NSTD0015
                                      93 *                                                                          NSTD0016
                                      94 TAPEIN2  DTFMT DEVADDR=SYS004,IOAREA1=IOAREA,BLKSIZE=80,TYPEFLE=INPUT,    *NSTCOC17
                                                        EOFADDR=ENDTAPE2,READ=BACK,FILABL=NSTD                      NSTD0018
                                     129 *                                                                          NSTC0019
                                     130 PRINT    DTFPR DEVICE=1403,DEVADDR=SYSLST,IOAREA1=IOAREA,BLKSIZE=80        NSTD0020
                                     151 *                                                                          NSTC0021
                                     152 CONSOLE  DTFCN BLKSIZE=80,DEVACDR=SYSLCG,IOAREA1=CAREA,RECFORM=FIXUNB,    *NSTD0022
                                                        WORKA=YES                                                   NSTD0023
                                     221 *                                                                          NSTDOC24
                                     222 *                                                                          NSTD0025
0031AC 0520                          223 START    BALR  2,0                         SET UP A BASE REGISTER          NSTD0026
0031AE                               224          USING *,2                                                         NSTDOC27
                                     225 * ** ROUTINE TO WRITE TAPE                                                 NSTD0028
                                     226          OPEN  TAPEOUT                     TO WRITE NSTD RECORDS           NSTDOC29
                                     234 GETCARC  GET   READER                      READ A CARC FROM CARD READER    NSTD0030
                                     239          PUT   TAPEOUT                     WRITE CARD IMAGE ON TAPE        NSTDOC31
0031D6 47F0 2010            031EE    244          B     GETCARD                     BRANCH ANC GET ANOTHER CARD     NSTDOC32
                                     245 ENDCARD  CLOSE TAPEOUT                     TO WRITE NSTD TRAILER LABEL     NSTD0033
                                     253 * ** ROUTINE TO READ TAPE FORWARD                                          NSTC0034
                                     254          OPEN  PRINT,TAPEIN                TO PROCESS NSTD LABEL           NSTD0035
                                     263 GETTAPE  GET   TAPEIN                      GET A CARD IMAGE FROM TAPE      NSTCOC36
                                     268          PUT   PRINT                       PRINT CARD IMAGE ON PRINTER     NSTDOC37
003216 47F0 2050            031FE    273          B     GETTAPE         BRANCH AND GET ANOTHER TAPE RECORD          NSTD0038
                                     274 ENDTAPE  CLOSE TAPEIN                      PROCESS NSTD LABELS             NSTDOC39
                                     282 * ** RCUTINE TO READ TAPE BACKWARDS                                        NSTDC040
                                     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 ANC GET ANOTHER TAPE RECORD         NSTD0044
                                     302 ENCTAPE2 CLOSE PRINT,TAPEIN2               BYPASS NSTC RECORDS             NSTD0045
                                     311          CNTRL TAPEIN2,REW                 REWIND TAPE TC LCAD POINT       NSTC0046
                                     317          EOJ                               NORMAL END OF JOB               NSTD0047
                                     320 * ** LABEL CREATION ROUTINE                                                NSTDOC48
00327A 4900 22A6            03454    321 LABELCUT CH    0,ALPHAO                    OPEN OR CLOSE                   NSTD0049
00327E 4770 20F0            0329E    322          BNE   TRAILCUT                    BRANCH IF CLOSE                 NSTC0050
003282 D227 221C 21CC 033CA 0337A    323          MVC   IOAREA(40),HEADER           MOVE HEADER TO I/O AREA         NSTD0051
                                     324 RITELAB  EXCP  OUTCCB                      WRITE LABEL                     NSTDOC52
                                     328          WAIT  OUTCCB                      WAIT FCR CCMPLETION             NSTDOC53
                                     334          LBRET 2                           RETURN CONTROL TO IOCS          NSTD0054
```

Figure C-1. 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/VS ASSEMBLER V 28.0 09.44 73-05-16

00329E D227 221C 21F4 033CA 033A2   337 TRAILOUT MVC    IOAREA(40),TRAILER     MOVE TRAILER LABEL TC I/O AREA    NSTD0055
0032A4 47F0 20DA            03288   338          B      RITELAB                BRANCH TO WRITE THE LABEL         NSTD0056
                                    339 * ** LABEL PROCESSING ROUTINE                                            NSTC0C57
0032A8 4900 22A6            03454   340 LABELIN  CH     0,ALPHA0               OPEN OR CLOSE                     NSTD0058
0032AC 4780 212C            032CA   341          BE     HEADIN                 OPEN TIME                         NSTD0059
                                    342 TRAILIN  EXCP   INCCB                  READ A TRAILER LABEL              NSTD0060
                                    346          WAIT   INCCB                  WAIT FCR 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     CCMPARE TRAILER LABEL             NSTD0C64
0032D2 4780 2102            03280   355          BE     TRAILIN                BRANCH TO GET ANOTHER RECORD      NSTD0065
0032D6 47F0 2152            03300   356          B      ERRLAB          BRANCH IF LABELS CC NOT COMPARE          NSTD0066
                                    357 HEADIN   EXCP   INCCB                  READ A HEADER LABEL               NSTD0067
                                    361          WAIT   INCCB                  WAIT FOR CCMPLETION               NSTC0068
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                     NSTC0070
0032F6 D527 221C 21CC 033CA 0337A   369 .        CLC    IOAREA(40),HEADER      DOES HEADER LABEL COMPARE         NSTD0071
0032FC 4780 212C            032CA   37C          BE     HEADIN                 IF YES, BRANCH ANC READ TAPE      NSTD0C72
                                    371 ERRLAB   PUT    CONSOLE,LABELERR       PUT LABEL ERROR MESSAGE           NSTD0073
                                    377          EOJ                           TERMINATE JOB                     NSTC0074
003312 5800 22A2            03450   380 EXITEOF  L      0,EOFIND               INCICATE ECF TO IOCS              NSTC0075
                                    381 EXIT     LBRET 2                       RETURN CONTROL TC IOCS            NSTD0076
                                    384 * CONSTANTS                                                              NSTC0C77
003318 404040404040C040             385 CAREA    DC     CL50' '                CONSOLE I/O AREA                  NSTD0078
00334A E4E2C5D940D3C1C2             386 LABELERR DC     C'USER LABELS DC NOT CCMPARE. ABNORMAL END OF JOB.'      NSTD0C79
00337A E4E2C5D940C8C5C1             387 HEADER   DC     CL40'USER HEADER LABEL'                                  NSTD0080
0033A2 E4E2C5D940E3D9C1             388 TRAILER  DC     CL40'USER TRAILER LABEL'                                 NSTD0081
0033CA 404040404040C40C             389 IOAREA   DC     CL80' '                 INPUT/OUTPUT AREA                NSTD0082
                                    390 INCCB    CCB    SYS004,INCCW           READ TAPE CCB                     NSTD0C83
                                    4C1 OUTCCB   CCB    SYS004,OUTCCW          WRITE TAPE CCB                    NSTC0084
00343A 0000000C000C
003440 020033CA0C00CC28             412 INCCW    CCW    X'02',IOAREA,X'00',40  READ TAPE CCW                     NSTC0085
003448 010033CA000CC028             413 OUTCCW   CCW    X'01',IOAREA,X'00',40  WRITE TAPE CCW                    NSTD0086
003450 0000C5C6                     414 EOFIND   DC     X'0000C5C6'                                              NSTD0C88
003454 00D6                         415 ALPHA0   DC     X'00D6'                                                  NSTC0089
0031AC                              416          END    START                                                    NSTD0C90
003458 5B5BC2D6D7C5D540             417                 =C'$$BOPEN '
003460 5B5BC2C3D3D6E2C5             418                 =C'$$BCLOSE'
003468 00003000                     419                 =A(READER)
00346C 00003038                     420                 =A(TAPEOUT)
003470 00003C90                     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 your label routine address
3. After you read the third tapemark you should issue a LBRET 1 and IOCS will branch to the end-of-file address.

Figure C-1. Reading, Writing, and Checking with Nonstandard labels (Part 2 of 2)

APPENDIX D: WRITING SELF-RELOCATING PROGRAMS

DOS/VS has the capability of executing self-relocating programs. A self-relocating program is an assembler-language program which can be executed at any location in virtual storage. If your system does not have the relocating loader, writing a self-relocating program is then an efficient coding technique because self-relocating programs are link-edited only once for execution in any partition. When link-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 storage 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 PHASE card must specify an origin of +0.
2. The program must relocate all address contants used in the program. Whenever possible, use the LA instruction to load an addres 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
```

3. If logical IOCS is used, the program must use the OPENR and CLOSER macro to open and close all files including console files.
4. If physical IOCS is used, the program must relocate all CCW address fields.
5. Register notation must be used with imperative I/O macros and supervisor macros. 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.

6. Use // LBLTYP before // EXEC card.

   The following rules apply to programs consisting of more than one control section.

7. 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.

8. 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

9. The calling program is responsible for relocating all address constants in the calling list(s). See Figure D-1 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 relocating 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 D-1. Relocating Address Constants in a Calling List

Advantage of Self-Relocating Programs

Self-relocating programs have the ability to run in any one of the 5 partitions without having to be link-edited again.

Another Way -- The Relocating Loader

Self-relocating programs are slightly more time-consuming to write and they usually require slightly more storage. They may only be written in assembler-language. For these reasons you may want to use the relocating loader instead. The relocating loader accomplishes the same thing as writing self-relocating programs but without any of these disadvantages. See the DOS/VS System Management Guide, GC33-5371, for a description of the relocating loader.

Programming Techniques

A self-relocating program 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 of 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 of 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 of 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 link-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:

```
       .
       LA      4,ADCONAME
       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):

A. If the CCW list dynamically changes during program execution:

```
         .
         LA    4,IOAREA
         STCM  4,X'07',TAPECCW
         .
TAPECCW  CCW   1,IOAREA,X'20',100
         .
         .
IOAREA   DS    CL100
```

B. 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
          .
          .
IOAREA    DS    CL100
```

or

```
          .
          USING *,12
          BALR
          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    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)
```

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 link-edited with an origin of +0. If a phase is link-edited with an origin of * or S, incorrect results will follow. This is because the linkage editor and the program have both added the load point to all address constants. Figure D-2 shows an example of a self-relocating program.

```
    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 D-2. Self-Relocating Sample Program

APPENDIX E: MICR DOCUMENT BUFFER FORMAT

Buffer Status Indicators		
Byte	Bit	Comment
0,	0	The document is ready for processing (you need never test this bit).
	1	Irrecoverable stacker select error, but all document data is present. You may continue to issue GETs and READs.
	2	Irrecoverable I/O error. An operator I/O error message is issued. The file is inoperative and must be closed.
	3	Unit Exception. You 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 your 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, or the file is closed (byte 0, bit 6 is on). The CHECK macro interrogates this bit. Note: You 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 E-1. MICR Document Buffer Format (Part 1 of 4)

Buffer Status Indicator (Continued)

Byte	Bit	Comment
1,	0	Your 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. You should interrogate byte 3 to determine the error 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 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 E-1. MICR Document Buffer Format (Part 2 of 4)

Buffer Status Indicators (Continued)		
Byte	Bit	Comment
		2. <u>1275 and 1419 (single address) without engineering change #125358</u>: 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. <u>1255, 1259, 1275, and 1419 (single or dual address) with engineering change #125358</u>: Bit 6 indicates that an auto-select codintion 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).** <u>Note</u>: 1. For the 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, 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 E-1. MICR document Buffer Format (Part 3 of 4)

Buffer Status Indicators (Continued)		
Byte	Bit	Comment
4		Inserted pocket code determination by your stacker select routine. Whenever byte 0, bits 2, 3, or 4 are on, this byte is X'00' because no document was read and your stacker selection routine was not entered. Whenever auto-selection occurs, this 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' (Except 1270 models 1 and 3) Reject Pocket - X'CF'
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 Areas
This additional buffer area can be used as a work area and/or output area. Its size is determined by the DTFMR ADDAREA operand. 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 your work area. The data is right-adjusted in the document data area. The length of this data area is determined by the DTFMR RECSIZE operand.
* 1275, 1419, and 1270 models 2 and 4 only. ** 1275 and 1419 only.

Figure E-1. MICR Document Buffer Format (Part 4 of 4)

APPENDIX F: AMERICAN NATIONAL STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII)

In addition to the EBCDIC mode, DOS/VS accepts magnetic tape files written in ASCII, 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, INnc.

DOS/VS processes ASCII 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.

Figure F-1 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 F1 list the three high-order bits. The rows along the left side of Figure F-1 are the four low-order bits.

For example, the letter P in ASCII is under column 5 and in row 0 and is described in ASCII notation as 5/0. ASCII 5/0 and EBCDIC X'50' represent the same binary configuration (B'01010000'). 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 F-2 for ASCII to EBCDIC correspondence.

Note: If an EBCDIC file is translated into ASCII, and then you translate back into EBCDIC, this substitute character may not receive the expected value.

Bits [3]				b7 →	0	0	0	0	1	1	1	1
				b6 →	0	0	1	1	0	0	1	1
				b5 →	0	1	0	1	0	1	0	1
b4 ↓	b3 ↓	b2 ↓	b1 ↓	Column → Row ↓	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 I (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

```
b7 b6 b5 b4 b3 b2 b1
0  0  1  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
I	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)	}	Closing Brace
.	Period (Decimal Point)	~	Tilde
/	Slant		
:	Colon		
;	Semicolon		

Figure F-1. ASCII Character Set

ASCII					EBCDIC				
Character	Col	Row	Bit Position		Col (in Hex)	Row (in Hex)	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 F-2. ASCII to EBCDIC Correspondence (Part 1 of 2)

ASCII				EBCDIC			
Character	Col	Row	Bit Pattern	Col (in Hex)	Row	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	
¦	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 F-2. ASCII to EBCDIC Correspondence (Part 2 of 2)

APPENDIX G: PAGE FAULT HANDLING OVERLAP

For some special types of processing, DOS/VS users often choose to write programs which, while executing as one DOS/VS user task, provide for the asynchronous processing of several tasks. This multitasking, which is not to be confused with DOS/VS multitasking, is generally used only in very sophisticated applications where no other technique would give acceptable performance.

This type of asynchronous processing (called 'private multitasking') is not supported by IBM and therefore is not documented as such. DOS/VS does, however, provide one tool which aids the programmer doing private multitasking. This is the ability to overlap the handling of a page fault in one private subtask with the processing of another private subtask.

If support is included in the supervisor (PHO=YES in the SUPVR macro), the user may set up an appendage routine which is to be entered whenever his DOS/VS task causes a page fault. This appendage routine acts as a 'dispatcher' for the private subtasks originating in the same DOS/VS task. The routine is called by the DOS/VS supervisor whenever the DOS/VS task causes a page fault and whenever a page fault has just been handled for that task.

The DOS/VS task is not put into the wait state when it causes a page fault as is usually the case, but remains dispatchable.

The linkage to the appendage is established by issuing a SETPFA macro. In addition, the appendage routine must not cause a page fault and therefore must be fixed in real storage using the PFIX macro before the SETPFA macro is issued. The appendage routine is given control in the supervisor state, with I/O interrupts disabled, and with protection key zero. Following is a description of the conventions observed by page fault appendages in DOS/VS as well as suggestions how an appendage should be set up.

REGISTER USAGE

The following registers are used to pass information between the supervisor and the page fault appendage:

- Register 7 contains the return address to the supervisor.
- Register 8 contains the address of the appendage routine, and can therefore be used as the base register of the routine.
- Register 13 contains a parameter with information about the page fault to be handled. The information in register 13 has the following format:

TIK Flags	address of page		PIK
0	1	2	3

Byte 0: TIK = Task interrupt key of interrupt task

Flags = used by the system.

Bytes 1-2: leftmost 16 bits of the address of the page which has to be handled. The remaining (rightmost) 8 bits of the address are considered to be zero.

Byte 3: PIK of requesting task

ENTRY LINKAGE

The entry coding for the appendage should be as follows:

```
USING  *,8     USE REGISTER 8 AS BASE REGISTER
B      entry A ENTRY POINT AFTER PAGE FAULT

B      entry B ENTRY POINT AFTER PAGE FAULT COMPLETION
```

The name specified in label is the entry point of the routine specified in the SETPFA macro. The branch to entry A is taken whenever the task causes a page fault and the branch to entry B is taken whenever a page fault for the task has been handled.

PAGE FAULT QUEUE

The appendage must have a queue with a four-byte entry for each private subtask controlled by that DOS/VS task. This queue is used to store the page fault information passed in register 13.

PROCESSING AT THE INITIATION OF A PAGE FAULT

When the routine is entered at entry A, register 13 contains information on the page fault which has just occurred. The appendage must store the contents of register 13 in the internal page fault queue, put the private subtask causing the page fault into an internal wait state, and, if possible, dispatch another private subtask.

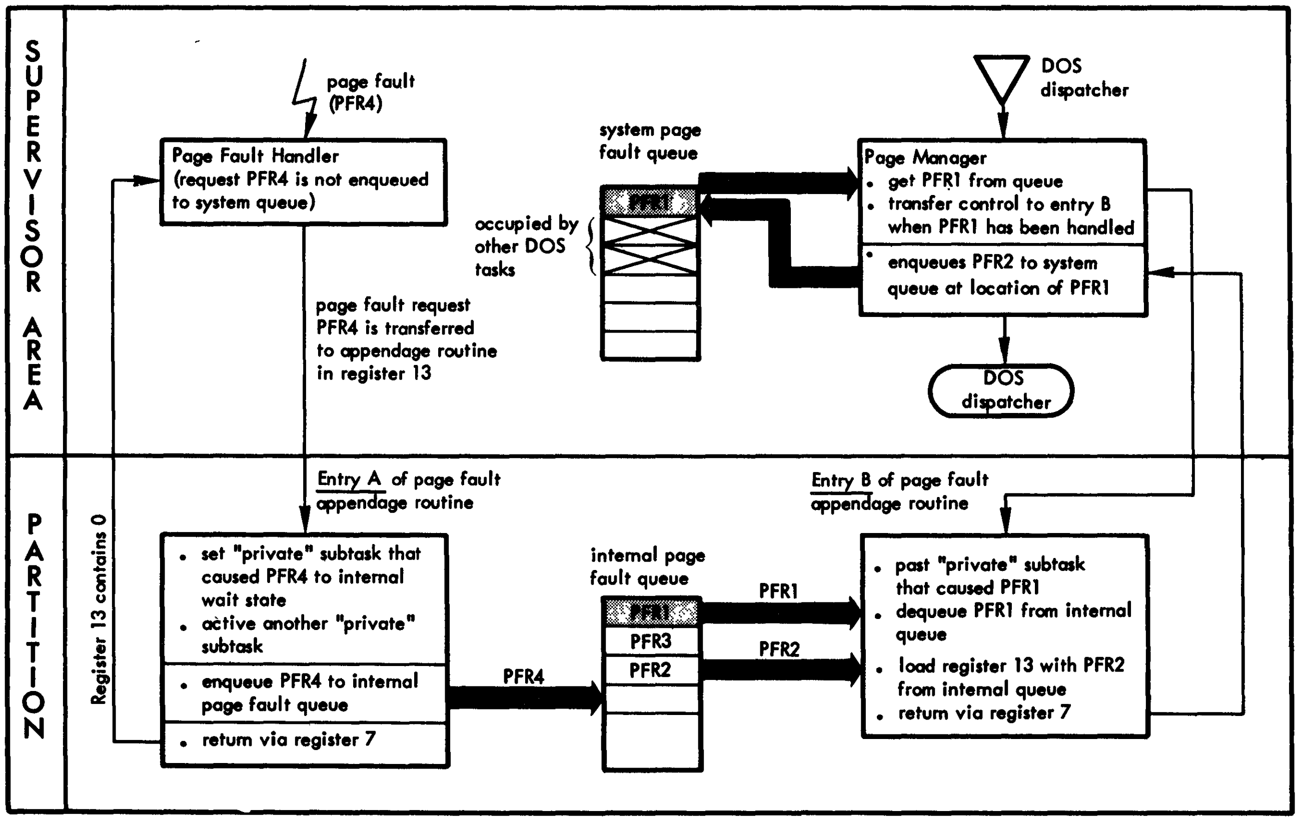
In addition, the appendage routine must store (1) the contents of the save area of the DOS/VS task in the save area of the private subtask which is being deactivated and (2) the contents of the private subtask to be activated in the save area of the DOS/VS task. If no private subtask is dispatchable it is the responsibility of the appendage routine to change the contents of the task save area so that the task will not continue to cause the same page fault each time it is dispatched. A possible solution is to dispatch a private task which does nothing but wait for a page fault to be handled.

The routine must then return control to the supervisor and pass a parameter in register 13. The parameter must either be zero, to indicate that a page fault request is pending for the DOS/VS task, or if no request is pending, the parameter must be the same as was passed to the appendage routine when it was called.

PROCESSING AT THE COMPLETION OF A PAGE FAULT

Whenever a page fault request has been handled for the DOS/VS task, control is passed to entry B. The appendage routine dequeues the request which has just been handled from the internal page fault queue and posts the private subtask which caused the page fault. If there are any more page faults requests in the queue, the information for the next request to be handled must be returned in register 13, otherwise register 13 must contain zero.

Figure G-1 is an example of how page faults are handled when the task causing the page faults has set up a page fault appendage.


SUPERVISOR AREA
PARTITION
page fault (PFR4)
Page Fault Handler (request PFR4 is not enqueued to system queue)
system page fault queue
PFR1
occupied by other DOS tasks
page fault request PFR4 is transferred to appendage routine in register 13
DOS dispatcher
Page Manager
• get PFR1 from queue
• transfer control to entry B when PFR1 has been handled
• enqueues PFR2 to system queue at location of PFR1
DOS dispatcher
Entry A of page fault appendage routine
Entry B of page fault appendage routine
Register 13 contains 0
• set "private" subtask that caused PFR4 to internal wait state
• active another "private" subtask
• enqueue PFR4 to internal page fault queue
• return via register 7
internal page fault queue
PFR1
PFR3
PFR2
PFR4
PFR1
PFR2
• post "private" subtask that caused PFR1
• dequeue PFR1 from internal queue
• load register 13 with PFR2 from internal queue
• return via register 7

Assumptions: . Sequence of page fault requests to be handled is PFR 1, 2, 3, 4.
. During handling of PFR1 3 other page fault requests have been caused by the same DOS/VS task

Figure G-1. Page Fault Handling Overlap

APPENDIX H: OPERAND NOTATION FOR VSAM GENCB, MODCB, SHOWCB, AND TESTCB MACROS

The addresses, names, numbers, and options required with operands in GENCB, MODCB, SHOWCB, and TESTCB can be expressed in a veriety of ways:

- An absolute numeric expression, for example, COPIES=10
- A character string, for example, DDNAME=DATASET
- A code or a list of codes separated by commas and enclosed in parentheses, for example, OPTCD=KEY or OPTCD=(KEY,DIR,IN)
- An expression valid for a relocatable A-type address constant, for example, AREA=MYAREA+4
- A register from 2 through 12 that contains an address or numeric value, for example SYNAD=(3); equated labels can be used to designate a register, for example, SYNAD=(ERR), where the following equate statement has been included in the program: ERR EQU 3
- An expression of the form (S,scon), where scon is any expression valid for an S-type address constant, including the base-displacement form
- An expression of the form (,scon), where scon is any expression valid for an S-type address constant, including the base-displacement form; the address specified by scon is indirect, that is, it points to the location that contains the value of the keyword

If an indirect S-type address constant is used, the value it points to must meet the following criteria:

- If it is a numeric quantity, a bit string representing codes, or a pointer, it must occupy a fullword of storage.
- If it is an alphameric character string, it must occupy two words of storage, be left aligned, and be filled on the right with blanks.

The expressions that can be used depend on the keyword specified. Additionally, register and S-type address constants cannot be used when MF=L is specified.

	Absolute Numeric	Code	Character String	Register	S-Type Address	Indirect S-Type Address	A-Type Address
GENCB Keywords							
BLK		X					
COPIES	X			X	X	X	
LENGTH	X			X	X	X	
WAREA				X	X	X	X
ACB Keywords (BLK=ACB)							
BUFND	X			X	X	X	
BUFNI	X			X	X	X	
BUFSP	X			X	X	X	
DDNAME			X			X	
EXLST				X	X	X	X
MACRF		X					
PASSWD				X	X	X	X
STRNO	X			X	X	X	
EXLST Keywords (BLK=EXLST)							
EODAD				X	X	X	X
EXCPAD				X	X	X	X
LERAD				X	X	X	X
SYNAD				X	X	X	X
RPL Keywords (BLK=RPL)							
ACB				X	X	X	X
AREA				X	X	X	X
AREALEN	X			X	X	X	
ARG				X	X	X	X
KEYLEN	X			X	X	X	
NXTRPL				X	X	X	X
OPTOD		X					
RECLEN	X			X	X	X	

MODCB MACRO OPERANDS

	Absolute Numeric	Code	Character String	Register	S-Type Address	Indirect S-Type Address	A-Type Address
<u>MODCB Keyword</u>							
{ACB|EXLST|RPL}				X	X	X	X
<u>ACB Keywords</u>							
BUFND	X			X	X	X	
BUFNI	X			X	X	X	
BUFSP	X			X	X	X	
DDNAME			X			X	
EXLST				X	X	X	X
MACRF		X					
PASSWD				X	X	X	X
STRNO	X			X	X	X	
<u>EXLST Keywords</u>							
EODAD				X	X	X	X
EXCPAD				X	X	X	X
LERAD				X	X	X	X
SYNAD				X	X	X	X
<u>RPL Keywords</u>							
ACB				X	X	X	X
AREA				X	X	X	X
AREALEN	X			X	X	X	
ARG				X	X	X	X
KEYLEN	X			X	X	X	
NXTRPL				X	X	X	X
OPTCD		X					
RECLEN	X			X	X	X	

SHOWCB MACRO OPERANDS

	Absolute Numeric	Code	Character String	Register	S-Type Address	Indirect S-Type Address	A-Type Address
{ABC|EXLST|RPL}				X	X	X	X
AREA				X	X	X	X
FIELDS		X					
LENGTH	X			X	X	X	
OBJECT		X					

	Absolute Numeric	Code	Character String	Register	S-Type Address	Indirect S-Type Address	A-Type Address
TESTCB Keywords							
{ABC|EXLST|RPL}				X	X	X	X
ERET				X	X	X	X
OBJECT		X					
ACB Keywords							
ACBLEN	X			X	X	X	
ATRB		X					
AVSPAC	X			X	X	X	
BUFND	X			X	X	X	
BUFNI	X			X	X	X	
BUFNO	X			X	X	X	
BUFSP	X			X	X	X	
CINV	X			X	X	X	
DDNAME			X			X	
ERROR	X			X	X	X	
EXLST				X	X	X	X
FS	X			X	X	X	
KEYLEN	X			X	X	X	
LRECL	X			X	X	X	
MACRF		X					
NCIS	X			X	X	X	
NDELR	X			X	X	X	
NEXCP	X			X	X	X	
NEXT	X			X	X	X	
NINSR	X			X	X	X	
NIXL	X			X	X	X	
NLOGR	X			X	X	X	
NRETR	X			X	X	X	
NSSS	X			X	X	X	
NUPDR	X			X	X	X	
OFLAGS		X					
PASSWD				X	X	X	X
RKP	X			X	X	X	
STMST						X	
STRNO	X			X	X	X	
EXLST Keywords							
EODAD				X	X	X	X
EXCPAD				X	X	X	X
EXLLEN	X			X	X	X	
LERAD				X	X	X	X
SYNAD				X	X	X	X
RPL Keywords							
ACB				X	X	X	X
AREA				X	X	X	X
AREALEN	X			X	X	X	
ARG				X	X	X	X
FDBK	X			X	X	X	
KEYLEN	X			X	X	X	
NXTRPL				X	X	X	X
OPTCD		X					
RBA	X			X	X	X	
RECLEN	X			X	X	X	
RPLLEN	X			X	X	X	

APPENDIX I: PARAMETER LISTS FOR VSAM GENCB, MODCB, SHOWCB, AND TESTCB MACROS

The VSAM macros for generating, modifying, displaying, and testing an access-method control block, exit list, or request parameter list use an internal parameter list to describe the actions that you specify when you code the macros. The _standard_ form of these macros builds a parameter list in-line and processes it; the _list_ form builds a parameter list in an area specified by the user; the _execute_ form processes a previously built parameter list; the _generate_ form builds a parameter list in an area specified by the user and also processes it.

For special purposes, such as developing high-level programming languages, you may want to build and process parameter lists without using the macros. This appendix describes the format of the parameter lists and gives the codes used for the operands of each of the macros. The formats and codes are fixed, so that you can build and alter them by your own methods.

A parameter list contains a variable number of entries of three types:

1. At the beginning of the list, _addresses_ of entries of the second and third types; the addresses are fullwords, and the high-order bit of the last fullword is 1.
2. Next, a _header entry_ containing general information about the block or list that you want to generate, modify, display, or test.
3. At the end of the list, _keyword entries_ describing each field that you want to generate, modify, display, or test.

Entries of the second and third types are described separately for GENCB, MODCB, SHOWCB, and TESTCB.

THE GENCB PARAMETER LIST

The Header Entry

0	block or list[1]	X'01'	number of copies
4	address of the area you are providing, or 0s		
8	length of the area, or 0s	(reserved)	

[1] X'A0' indicates access-method control block (ACB)
X'B0' indicates exit list (EXLST)
X'C0' indicates request parameter list (RPL)

Keyword Entries

The parameter list for GENCB contains no keyword entries if you are generating a default ACB, Exit List, or RPL.

0	keyword code[1]	(reserved)
4	{value|address|option[2]|name} of the keyword	
8	(required for some keywords[3])	

[1]

ACB	
BUFND	AL2(4)
BUFNI	5
BUFSP	7
DDNAME	9
EXLST	12
MACRF	18
PASSWD	30
STRNO	32

EXLST	
EODAD	AL2(37)
EXCPAD	38
LERAD	40
SYNAD	41

RPL	
ACB	AL2(43)
AREA	44
AREALEN	45
ARG	46
KEYLEN	48
NXTRPL	51
OPTCD	52
RECLEN	53

[2] You indicate the options for MACRF and OPTCD with a 1 in a bit of the fullword:

MACRF Option	bit
KEY	0
ADR	1
CNV	2
SEQ	3
SKP	4
DIR	5
IN	6
OUT	7
NUB	8
UBF	9

OPTCD Option	Bit
KEY	0
ADR	1
CNV	2
SEQ	3
DIR	4
SKP	5
NUP	8
UPD	9
NSP	10
KEQ	11
KGE	12
FKS	13
GEN	14
MVE	15
LOC	16

[3] The third fullword is required for the ACB operand DDNAME and for all of the EXLST operands, for which the third fullword indicates [,{A|N}] [,L]:

Bit	Meaning when set to 1
0	Addres is active (A)
1	Address is not active (N)
2	Address is of a field containing the name of an exit routine to be loaded (L)
3	Address is specified in the preceding fullword of this entry
4-31	Unused

An entry for any of the other operands is 8 bytes long.

THE MODCB PARAMETER LIST

The Header Entry

0	block or list[1]	X'02'	(reserved)
4	address of the block or list to be modified		

[1] X'A0' indicates access-method control block (ACB)
X'B0' indicates exit list (EXLST)
X'C0' indicates request parameter list (RPL)

Keyword Entries

0	keyword code[1]	(reserved)
4	{value|address|option[2]|name} of the keyword	
8	(required for some keywords[3])	

[1]

ACB	
BUFND	AL2(4)
BUFNI	5
BUFSP	7
DDNAME	9
EXLST	12
MACRF	18
PASSWD	30
STRNO	32

EXLST	
EODAD	AL2(37)
EXCPAD	38
LERAD	40
SYNAD	41

RPL	
ACB	AL2(43)
AREA	44
AREALEN	45
ARG	46
KEYLEN	48
NXTRPL	51
OPTCD	52
RECLEN	53

[2] You indicate the options for MACRF and OPTCD with a 1 in a bit of the fullword:

MACRF Option	Bit
KEY	0
ADR	1
CNV	2
SEQ	3
SKP	4
DIR	5
IN	6
OUT	7
NUB	8
UBF	9

OPTCD Option	Bit
KEY	0
ADR	1
CNV	2
SEQ	3
DIR	4
SKP	5
NUB	8
UPD	9
NSP	10
KEQ	11
KGE	12
FKS	13
GEN	14
MVE	15
LOC	16

With the MODCB macro, there are no defaults for these options. When you code a bit for the OPTCD operand, the contrary bit that was previously set is turned off. For example, if KEY was previously set, and you set ADR, KEY is turned off, since a request parameter list can be set for only one type of access.

[3] The third fullword is required for the ACB operand DDNAME and for all of the EXLST operands, for which the third fullword indicates [,{A|N}] [,L]:

Bit	Meaning when set to 1
0	Address is active (A)
1	Address is not active (N)
2	Address is of a field containing the name of an exit routine to be loaded (L)
3	Address is specified in the preceding fullword of this entry
4-31	Unused

An entry for any of the other operands is 8 bytes long.

THE SHOWCB PARAMETER LIST

The Header Entry

0	block or list[1]	X'03'	type of object to be displayed
4	address of the block or list to be displayed		
8	address of the display area you are providing		
12	length of the display area	(reserved)	

[1] X'00' indicates that no block or list is specified, to display the standard length of the block(s) or list(s) specified by the keyword ACBLEN, EXLLEN, or RPLLEN
X'A0' indicates ACB
X'B0' indicates Exit List
X'C0' indicates RPL

AL2(0) indicates the data of a file
AL2(1) indicates the index of a file

Keyword Entries

0	keyword code[1]	(reserved)

[1]

ACB	
ACBLEN	AL2(3)
AVSPAC	2
BUFND	4
BUFNI	5
BUFNO	6
BUFSP	7
CINV	8
DDNAME	9
ERROR	11
EXLST	12
FS	13
KEYLEN	16
LRECL	17
NCIS	19
NDELR	20
NEXCP	21
NEXT	22
NINSR	23
NIXL	24
NLOGR	25
NRETR	26
NSSS	27
NUPDR	28
PASSWD	30
RKP	31
STMST	35
STRNO	32

EXLST	
EODAD	AL2(37)
EXCPAD	38
EXLLEN	42
LERAD	40
SYNAD	41

RPL	
ACB	AL2(43)
AREA	44
AREALEN	45
ARG	46
FDBK	56
KEYLEN	48
NXTRPL	51
RBA	57
RECLEN	53
RPLLEN	55

THE TESTCB PARAMETER LIST

The Header Entry

Offset			
0	block or list[1]	X'04'	type of object to be tested
4	address of the block or list to be tested		
8	address of the routine to return to for unequal comparisons, or 0s		
12	(reserved)		

[1] X'00' indicates that no block or list is specified, to test the standard length of the block(s) or list(s) specified by the operand ACBLEN, EXLLEN, or RPLLEN
X'A0' indicates ACB
X'B0' indicates Exit List
X'C0' indicates RPL

AL2(0) indicates the data of a file
AL2(1) indicates the index of a file

Keyword Entries

Offset		
0	keyword code[1]	(reserved)
4	{value|address|option[2]|name|code[3]} of the keyword	
8	(required for some keywords[4])	

[1]

ACB	
ACBLEN	AL2(3)
ATRB	1
AVSPAC	2
BUFND	4
BUFNI	5
BUFNO	6
BUFSP	7
CINV	8
DDNAME	9
ERROR	11
EXLST	12
FS	13
KEYLEN	16
LRECL	17
MACRF	18
NCIS	19
NDELR	20
NEXCP	21
NEXT	22
NINSR	23
NIXL	24
NLOGR	25
NRETR	26
NSSS	27
NUPDR	28
OFLAGS	29
PASSWD	30
RKP	31
STMST	35
STRNO	32

EXLST	
EODAD	ALD2(37)
EXCPAD	38
EXLLEN	42
LERAD	40
SYNAD	41

RPL	
ACB	ALD2(43)
AREA	44
AREALEN	45
ARG	46
FDBK	56
KEYLEN	48
NXTRPL	51
OPTCD	52
RBA	57
RECLEN	53
RPLLEN	55

[2] You indicate the options for ATRB, MACRF, OFLAGS, and OPTCD by a 1 in a bit of the fullword:

ATRB Option	Bit
KSDS	0
ESDS	1
WCK	2
SSWD	3
REPL	4

MACRF Option	Bit
KEY	0
ADR	1
CNV	2
SEQ	3
SKP	4
DIR	5
IN	6
OUT	7
UPD	8

OFLAGS Option	Bit
OPEN	0

OPTCD Option	Bit
KEY	0
ADR	1
CNV	2
SEQ	3
DIR	4
SKP	5
NUP	8
UPD	9
NSP	10
KEQ	11
KGE	12
FKS	13
GEN	14
MVE	15
LOC	16

[3] The codes for ERROR are given in the section "Opening and Closing Files" in the VSAM chapter. The codes for FDBK are given in the section "Requesting Access to Files" in the VSAM chapter.

[4] The third fullword is required for the ACB operands DDNAME and STMST and for all of the EXLST operands, for which the third fullword indicates [,{A|N}][,L]:

Bit	Meaning when set to 1
0	Address is active (A)
1	Addres is not active (N)
2	Address is of a field containing the name of an exit routine to be loaded (L)
3	Address is specified in the preceding fullword of this entry
4-31	Unused

An entry for any of the other operands is 8 bytes long.

APPENDIX J: USING ISAM PROGRAMS WITH VSAM FILES

The ISAM Interface Program (IIP) permits ISAM programs to process VSAM files. ISAM programs do not have to be reassembled or relinkedited to use the IIP. Also, both ISAM and VSAM files, through the IIP, can be processed concurrently.

To use the IIP, you must convert your ISAM files to VSAM files and the job control statements of your ISAM program to VSAM job control statements. You must also ensure that ISAM programs meet certain restrictions for using the IIP. See the DOS/VS Data Management Guide, GC33-5372 for information on converting files and job control statements and on IIP restrictions.

You do not have to make any changes in the DTFIS to use ISAM programs with the IIP. The following DTFIS parameters are used by the IIP; all other parameters are ignored:

- ERREXT=YES
  (See Figure J-1 for a description of the ERREXT parameter list with IIP.)
- IOAREA=name
  (used when IOROUT=LOAD)
- IOAREAS=name
  (used if SETL BOF is issued)
- IOREG=(r)
- IOROUT= LOAD ADD RETRVE ADDRTR
- KEYARG=name
- RECFORM= FIXUNB FIXBLK
- WORKL=name
- WORKR=name
- WORKS=YES

The IIP interprets the error return codes from VSAM. If the VSAM condition corresponds to an ISAM condition, the respective bit in the filenameC byte in the DTFIS is posted. For unrecoverable errors that cannot be posted in the filenameC byte, a message is printed on the console, the VSAM file is closed (by the VSAM close routine), and the task is terminated.

If an I/O error occurs and ERREXT=YES is specified in the DTFIS, the IIP posts additional error information in the ERREXT parameter list. Figure J-1 shows the format of the ERREXT parameter list, and Figures J-2 and J-3 show the formats of the filenameC byte for ISAM processing through the IIP.

Bytes	Contents
0-3	DTF address
4-7	Not supported by the IIP
8-15	Not supported by the IIP
16:	
Bit 0	Data
Bit 1	VSAM Sequence Set
Bit 2	VSAM Index Set
Bits 3-5	Not used
Bit 6	Read operation
Bit 7	Write operation
17	Not supported by the IIP

Figure J-1. ERREXT Parameter List for ISAM Programs with the IIP

Bit	Cause in ISAM	Cause in IIP/VSAM
0	DASD error	DASD error
1	Wrong length record	Not set
2	End of file	End of file
3	No record found	No record found
4	Illegal ID specified	Not supported by IIP
5	Duplicate record	Duplicate record
6	Overflow area full	No more VSAM data space available
7	Overflow	Not set

Figure J-2. FilenameC with IIP when IOROUT=ADD, RETRVE, or ADDRTR

Bit	Cause in ISAM	Cause in IIP/VSAM
0	DASD error	DASD error
1	Wrong length record	Not set
2	Prime data area full	No more VSAM data space
3	Cylinder index area full	No more VSAM data space
4	Master index full	No more VSAM data space
5	Duplicate record	Duplicate record
6	Sequence check	Sequence check
7	Prime data area overflow	Not set

If there is no more VSAM data space, all three bits are set.

Figure J-3. FilenameC with IIP when IOROUT=LOAD

GLOSSARY

This glossary defines most of the terms used in this book. For a more complete list of data processing terms, refer to *IBM Data Processing Glossary*, GC20-1699.

access method: Any of the data management techniques (sequential, direct, indexed sequential, or virtual storage) available for transferring data between virtual 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/370 8-bit environment is zero.

associated file: A file for one function used in association with another file for another function to be performed in the same set of cards on a 2560, 3525, or 5425 card device. For example, with three associated files the same set of cards could be read and punched and printed. A file definition must be given for each associated file. **Not** synonymous with *combined file*.

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 saving storage space or increasing the efficiency of access or processing.
2. A physical record on magnetic tape or DASD.

block prefix: An optional, 0-99 byte field preceding an ASCII record on magnetic tape. It contains data which you specify or, for variable length ASCII records, the physical record length.

buffer:
1. A storage device in which data is assembled temporarily during data transfer. An example is the 2821 control unit, a control and buffer storage unit for card readers, card punches, and printers.
2. During I/O operations, a portion of virtual storage into which data is read or from which data is written. Synonymous with *I/O area*.

CCB: See *command control block*

CCW: See *channel command word*

channel command word (CCW): A doubleword at the location in virtual storage specified by the channel address word. One or more CCWs make up the *channel program*.

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 SIO machine 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 is issued in a program, checkpoint records are created. These records contain the status of the job and the system. When it is desired 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.

combined file: A single file on the 1442, 2520, or 2540 card device which both reads and punches the same set of cards. **Not** synonymous with *associated file*.

command control block (CCB): A 16-byte field required for each channel program executed by physical IOCS. This field is used for communication between physical IOCS and the program.

communication region: An area of the supervisor set aside for interprogram and intraprogram communication. It contains information useful to both the supervisor and your 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 DOS/VS.

control section: The smallest separately relocatable unit of a program; that portion of code which you specify to be an entity, all elements of which are to be loaded into contiguous virtual storage locations.

data conversion: The process of changing data from one form of representation to another.

data set security: A feature that provides protection for disk files. A secured file cannot be accidentally accessed by a program.

device independence: The capability of a program to process the same type of data on devices (printers, magnetic tape, or disk).

DTF (define the file) macro: A macro used for all access methods except VSAM to describe the characteristics of an input/output file, indicate the type of processing for the file, and specify the virtual storage areas and routines to be used in processing the file. These things are described using the appropriate parameters in the keyword operands of the DTF macro.

dump: To display the contents of virtual storage.

extent: A contiguous space on a direct access storage device occupied by, or reserved for, a particular file.

fetch:
1. To bring a program phase into virtual storage from a core image library for immediate execution.
2. The routine that retrieves requested phases and loads them into virtual storage,
3. The name of a macro (FETCH) used to transfer control to the system loader.
4. To transfer control to the system loader.

file:
1. The major unit of physical data, consisting of a collection of physical records in one of several prescribed arrangements and described by control information to which the system has access. For example, a deck of cards containing payroll data (one record for each employee describing his rate of pay, deductions, etc.), or a disk extent containing inventory information (one block for each inventory item describing the cost, selling price, number in stock, etc.).
2. A representation within a program of the logical characteristics of a *file* as defined in definition 1, above. The representation is achieved by a DTF macro or by an ACB, EXLST, and RPL macro. A given file within a program cannot represent any physical file which has the same logical characteristics.

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: A portion of storage into which data is read or from which data is written. I/O means Input/Output. I/O area is synonymous with *buffer* (definition 2).

IOCS (input/output control system): A group of macros and the routines which process them provided by IBM for handling the transfer of data between virtual storage and external storage devices.

load: To read a phase into virtual 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.

logical file: See *file,* definition 2.

logical record: A record identified from the standpoint of its content, function, and use rather than its physical attributes; that is, one which is meaningful with respect to a program. (Contrasted with *physical record*).

macro: A single assembler language instruction which is equivalent to a sequence of assembler language instructions. Thus when you specify one macro, the assembler generates a number of assembler language instruction.

macro definition: A set of statements which defines the name of, format of, and conditions for generating a sequence of assembler language instructions from a single source instruction.

macro system: The method by which macros are coded and by which the assembler program analyzes the macros and generates the appropriate sequence of assembler language instructions.

main task: The main program within a partition in a multiprogramming environment. (Compare with *subtask*).

MPS: See *multiprogramming system*.

multi-file volume: A unit of recording media, such as a magnetic tape reel or disk pack, that contains more than one file.

multiprogramming: A technique whereby two or more problem programs may execute concurrently in one computer, sharing system resources between them.

multitasking: A technique whereby one or more subtasks, attached to a main task within one partition, can execute concurrently.

multi-volume file: A file which, due to its size, requires more than one unit of recording media (such as magnetic tape reel or a disk pack) to contain the entire file.

nonstandard labels: Labels that do not conform to the System/370 standard label conventions. 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, language processors, and data as needed for continuous throughput of a series of jobs.

phase: The smallest complete unit that can be referenced in a core image library. Each program overlay is a complete phase. If the program has no overlays, the program itself is a complete phase.

physical file: See *file*, definition 1.

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 record*).

private library: A relocatable, core image, or source statement library that is separate and distinct from the corresponding system library.

problem program:

1. Your object program. It can be produced by any of the language translators. It consists of instructions and data necessary to solve your data-processing problem or to achieve a certain result.
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. This includes IBM-supplied programs such as language processors, linkage editor, librarian, sort/merge, and utilities, as well as programs which you supply. The term *processing program* is in contrast to the term *control program*.

real address area: The area of virtual storage where virtual addresses are equal to real addresses.

real partition: A division of the real address area of virtual storage that may be allocated for programs that are not to be paged, or programs that contain pages that are to be fixed.

real storage: The storage of a System/370 computing system from which the central processing unit can directly obtain instructions and data, and to which it can directly return results.

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: The attribute of a module, control section, or phase whose address constants can be modified to compensate for a change in origin.

relocatable program: A program that can be loaded into any area of virtual storage by the loader of a supervisor with relocating load support.

resource: Any facility of the computing or operating system required by a job or task. This includes virtual storage, input/output devices, the central processing unit, files, and control and processing programs.

restart: See *checkpoint/restart*.

self-relocating: A routine that is loaded at any doubleword boundary in the problem program area and can adjust its address values so as to be executed at that location.

self-relocating program: A program that can be loaded into any area in the problem program area of storage by having an initialization routine to modify all address constants at object time.

shared virtual area: An area located in the highest addresses of virtual storage. It can contain a system directory list of highly used phases and resident programs that can be shared between partitions.

storage: The term as used in this book is synonymous with *virtual storage*. See virtual storage.

subset module: A logic module which is a subset or component of a superset module.

subtask: A task in which control is initiated by a main task by means of a macro that attaches it.

superset module: A logic module which performs all of the functions of its subset or component modules, avoiding duplication and therefore saving storage space.

supervisor: The main 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. It coexists in storage with problem programs.

symbolic I/O assignment: A means by which your program 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.

system directory list: A list containing directory entries of highly used phases and of all phases resident in the shared virtual area. This list is placed in the shared virtual area.

telecommunication: Data transmission between a computer and remote stations.

teleprocessing: Same as telecommunication.

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 one program cannot be concurrently accessed by another program.

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.

virtual address area: The area of virtual storage whose addresses are greater than the highest address of the real address area.

virtaul partition: A division of the virtual address area of virtual storage that is allocated for programs that be paged.

virtual storage: Addressable space that appears to you as real storage, from which instructions and data are mapped into real storage locations. The size of virtual storage is limited bu the addressing scheme of the computing system and by the amount of auxiliary storage available, rather than by the actual number of real storage locations.

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 a 3330 disk storage drive.

work area: A portion of virtual storage used for processing an individual record.

INDEX

DOS/VS
Supervisor and I/O Macros

READER'S COMMENT FORM

GC33-5373-2

This sheet is for comments and suggestions about this manual. We would appreciate *your* views, favorable or unfavorable, in order to aid us in improving *this* publication. This form will be sent directly to the author's department. Please include your name and address if you wish a reply. Contact your IBM branch office for answers to technical questions about the system or when requesting additional publications. Thank you.

Name

Address

What is your occupation?

How did you use this manual?

As a reference source

As a classroom text

As a self-study text

Your comments* and suggestions:

*** We would especially appreciate your comments on any of the following topics:**

Clarity of the text	Accuracy	Index	Illustrations	Appearance	Paper
Organization of the text	Cross-references	Tables	Examples	Printing	Binding

YOUR COMMENTS, PLEASE . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM. ·

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

CUT ALONG THIS LINE

Fold

Fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
1133 Westchester Avenue
White Plains, N.Y. 10604

Attention: Department 813 BP

Fold

Fold

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

GC33-5373-2

DOS/VS Supervisor and I/O Macros Printed in U.S.A. GC33-5373-2

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)