Jan 1965

File No. 7090-29
Form C28-6345-2

IBM Systems Reference Library

IBM 7090/7094 IBSYS Operating System

Input/Output Control System

This publication provides a description of the 7090/7094 Input/Output Control System and includes detailed programming information. IOCS is a flexible programming system that automatically controls transmission of data to and from recording devices and makes the data readily available for processing.

The publication contains discussions of basic concepts and explains the use of IOCS commands and routines. The techniques of sequential processing are discussed in the main body of the manual, and a separate section is provided on random processing. Several sample programs are included.

Two forms of IOCS are accessible to the programmer. Library IOCS is contained in the IBJOB Subroutine Library and is used in conjunction with the Macro Assembly Program (MAP). Full IOCS (7090-I/O-919) is provided as an independent system for use with the IBM 7090/7094 FORTRAN II Assembly Program (IBSFAP). Differences in the two systems are made explicit in the publication. A third form, FORTRAN IOCS, is used by FORTRAN IV object programs.

Preface

The 7090/7094 Input/Output Control System (IOCS) described in this publication is designed for source programs that are assembled and executed under control of the IBM 7090/7094 IBSYS Operating System.

The two major forms of IOCS are discussed in detail. Library IOCS is a relocatable form of IOCS used in conjunction with the IBM 7090/7094 IBJOB Processor and the Macro Assembly Program. Full IOCS is an independent subsystem used in conjunction with the IBM 7090/7094 FORTRAN II Assembly Program (IBSFAP).

Full IOCS can be used with an IBM 709 Data Processing System if it is equipped with the Data Channel Trap feature.

Readers who intend to use Library IOCS are assumed to be familiar with the contents of the following publications:

IBM 7090 Data Processing System — Reference Manual, Form A22-6528, or *IBM 7094 Data Processing System—Reference Manual*, Form A22-6703

IBM 7090/7094 IBSYS Operating System: IBJOB Processor, Form C28-6275

IBM 7090/7094 Programming Systems: Macro Assembly Program (MAP) Language, Form C28-6311

Readers who intend to use Full IOCS are assumed to be familiar with either of the first two manuals cited above, and with the contents of the following publications:

IBM 7090/7094 IBSYS Operating System: System Monitor (IBSYS), Form C28-6248

IBM 7090/7094 Programming Systems: FORTRAN II Assembly Program (FAP), Form C28-6235

Readers who intend to use IOCS with IBM 7340 Hypertape Drives should be familiar with the contents of the following publication:

IBM 7340 Hypertape Drive, Model 1, Form G22-6634

Readers who intend to use IOCS with IBM 1301 Disk Storage should be familiar with the contents of the following publications:

IBM 1301 Disk Storage, Form G22-6595

IBM 1301 and 1302 Disk Storage, Models 1 and 2, with the IBM 7090, 7094, and 7094 Model II Data Processing Systems, Form A22-6785

Also useful is the publication *IBM 1301 Input/Output Control System for 1410 and 7000 Series Data Processing Systems*, Form J28-8064.

Readers who intend to use IOCS with IBM 7320 Drum Storage should be familiar with the contents of the following publication: *IBM 7320 Drum Storage with 7090 and 7094 Systems*, Form A22-6747.

The internal functioning of IOCS is explained in the publication *IBM 7090/7094 Input/Output Control System, Programming Systems Analysis Guide*, Form C28-6773.

This publication is a reprint of Form C28-6345-1, incorporating changes released in the following Technical Newsletters:

FORM NO.	PAGES	DATED
N28-0105	27, 28, 42, 43, 62-64, 77, 78	6/10/64
N28-0121	4, 47, 61, 84.1	7/31/64

The original publication and applicable Newsletters are not obsoleted.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM Corporation, Programming Systems Publications, Dept. D91, PO Box 390, Poughkeepsie, N. Y. 12602

Contents

The IBM 7090/7094 Input/Output Control System (IOCS) is a flexible program that provides automatic transmission of data to and from recording devices.

IOCS aids the programmer by simplifying the input/output aspects of his programs and by making data readily available for processing. The system is adaptable to a variety of input/output unit configurations and makes it easier to modify the coding in an existing program to fit a new configuration. These features were built into IOCS while retaining absolute reliability and a high degree of efficiency.

Once properly initialized, IOCS maintains a full supply of input data and records completed output records. IOCS also checks and prepares labels, detects and attempts to correct transmission errors, reports on the results of each transmission, allows checkpoints and restarts, and permits printing of on-line messages.

By using IOCS, the programmer can disassociate himself almost completely from input/output problems such as timing, overlap, and differences in the characteristics of recording devices. This permits him to concentrate on his primary task – the processing of data inside the computer.

IOCS also aids in the modification of existing programs when additional equipment or new devices are added to a system configuration. It requires considerably more effort to revise individual input/output instructions to fit a new configuration than it does to amend IOCS programs. Thus, if IOCS has been used, programs can be changed more rapidly to fit an expanded data processing system.

Machine Requirements

The minimum machine requirements for use of IOCS are an IBM 7090/7094 Data Processing System with:

1. Three IBM 729 Magnetic Tape Units or IBM 7340 Hypertape Drives.
2. Five other units, which may be any combination of IBM 729 Magnetic Tape Units, IBM 7340 Hypertape Drives, selected cylinders of IBM 1301 Disk Storage, or IBM 7320 Drum Storage Units.
3. One IBM 716 Printer.
4. One IBM 711 Card Reader if the System Library is on disk storage, drum storage, or Hypertape.
5. One IBM 7909 Data Channel if either IBM 1301 Disk Storage Units, IBM 7320 Drum Storage Units, or IBM 7340 Hypertape Drives, or any combination of these is used.
6. One IBM 7631 File Control with the Cylinder Mode feature if IBM 1301 Disk Storage or IBM 7320 Drum Storage is used.
7. One 7640 Hypertape Control if IBM 7340 Hypertape Drives are used.

These machine requirements are the same as those specified for use of the IBJOB Processor Monitor and the System Monitor (IBSYS).

Forms of the Input/Output Control System (IOCS)

Before a programmer can begin processing, he must provide IOCS with specific information in control cards and/or within his source program, depending on the form of IOCS he intends to use.

This information is used to reserve needed storage areas (buffers and buffer pools) and to set up control words that guide IOCS in doing its work. The interpretation of the programmer's specifications and the generation of control words and buffer pools has been designated as "preprocessing" and "initialization."

The two forms of IOCS available to the programmer (Full IOCS and Library IOCS) differ principally in the manner in which preprocessing and initialization are accomplished.

Full IOCS

Full IOCS is a complete subsystem operating directly under the IBSYS System Monitor. This form contains the following:

1. The Preprocessor, which handles the preparatory functions,
2. Operating subroutines, which accomplish specified input/output functions during execution of the object program,
3. A postprocessor, which handles housekeeping details at the end of a job.

Full IOCS is called directly by using a SEXECUTE IOCS control card. This control card causes the System Monitor to initiate loading of the Preprocessor. The Preprocessor then performs the housekeeping functions needed to establish file control blocks, performs tape assignments, sets up the configuration of IOCS requested for a particular run, and prepares for loading the object program.

Full IOCS is used in conjunction with the FORTRAN II Assembly Program (IBSFAP). It is sometimes called "IOCS with Preprocessor" or "Independent IOCS."

SHARE 7090 9PAC and IBM 709/7090 Commercial Translator Processor both contain modified forms of the Labels level of Full IOCS. The information on Full IOCS in this publication also applies to these two systems, including most calling sequences and error messages.

The 9PAC programmer uses Full IOCS calling sequences to reach desired IOCS routines. A list of Full IOCS routines that can be used with 9PAC is provided in the publication *IBM 7090 Programming Systems, SHARE 7090 9PAC, Part 3: The Reports Generator,* Form J28-6168.

In order to use the modified Full IOCS within Commercial Translator, the programmer must provide the necessary linkage through communication words in the first four subroutines in Commercial Translator. Use of IOCS with Commercial Translator is discussed in the publication *IBM 709/7090 Commercial Translator Processor,* Form J28-6169.

Library IOCS

Library IOCS has been incorporated into the IBJOB Subroutine Library and is used in conjunction with the IBJOB Processor Monitor and the IBJOB Loader.

Library IOCS is a relocatable form of IOCS that contains operating subroutines that are identical to those in Full IOCS. However, in Library IOCS, the IBJOB Loader performs the preprocessing tasks that are accomplished by the Preprocessor in Full IOCS. The user of Library IOCS processes data records in the same manner as the user of Full IOCS, except that he transfers to each IOCS routine by using a slightly different symbolic name.

The symbolic names of the routines in Library IOCS are distinguished by the fact that each begins with a period (.). The names of the routines in Full IOCS do not contain periods.

For example, there is a routine in both forms of IOCS for initiating a checkpoint. A programmer using Full IOCS initiates the checkpoint with the instruction:

```
TSX     CKPT,4
```

whereas, a programmer using Library IOCS would use the instruction:

```
TSX     CKPT,4
```

A full listing of the symbolic names of routines in both forms of IOCS can be found at the beginning of the section entitled "Processing with Both Forms of the Input/Output Control System (IOCS)."

FORTRAN IOCS

Within the IBJOB Library is an abbreviated form of IOCS, which is used by FORTRAN IV object programs. This restricted form is FORTRAN IOCS. The FORTRAN IV programmer has no direct access to these routines. They are utilized by input/output sequences generated by the FORTRAN IV Compiler.

FORTRAN IOCS contains the same routines as the Minimum level of Library IOCS, but many of the error-checking features and error messages have been deleted. By specifying FORTRAN IOCS, the FORTRAN IV programmer can reduce the amount of core storage occupied by IOCS routines. Unless FIOCS is specified in the $IBJOB control card, the Minimum level of Library IOCS will be used.

The following list indicates the facilities in Minimum IOCS that are not available in FORTRAN IOCS:

1. No secondary unit can be specified.
2. The DEFER and DEFERi options in the file mounting option of the $FILE card and FILE pseudo-operation cannot be specified. If either is specified, the Minimum level of Library IOCS is loaded.
3. No block sequencing or check summing can be used.
4. Most on-line operator and error messages have been deleted.
5. A list of files cannot be closed in a calling sequence. Files must be closed individually.
6. The .READR and .RELES routines are not provided even if FORTRAN IOCS is assembled with disk or drum storage capability.
7. No write checking can be performed on disk or drum storage files.

Figure 1 shows the relationship of the forms of IOCS within a partial drawing of components of the IBM 7090/7094 IBSYS Operating System.

Levels of the Input/Output Control System (IOCS)

Both forms of IOCS consist of a collection of generalized routines.

The programmer can specify any of four distinct packages of routines for sequential processing and can add Random capability to any of the four packages. Each of the four levels of IOCS — Input/Output Executor, Minimum, Basic, and Labels — provides a fixed number of IOCS routines.

The services and subroutines available at each level of sequential processing are as follows:

Input/Output Executor (IOEX): This level is a part of the System Monitor and provides trap supervision for both the object program and the Operating System. When the user specifies IOEX, he elects to code all of his input/output functions without using IOCS routines.

Minimum: This level provides trap supervision plus the IOCS routines for defining buffer pools, attaching files, opening and closing files, reading and writing of data records, and backspacing a record. Internal files cannot be used at this level.

Basic: This level includes all routines for sequential processing, except the routines associated with labels.

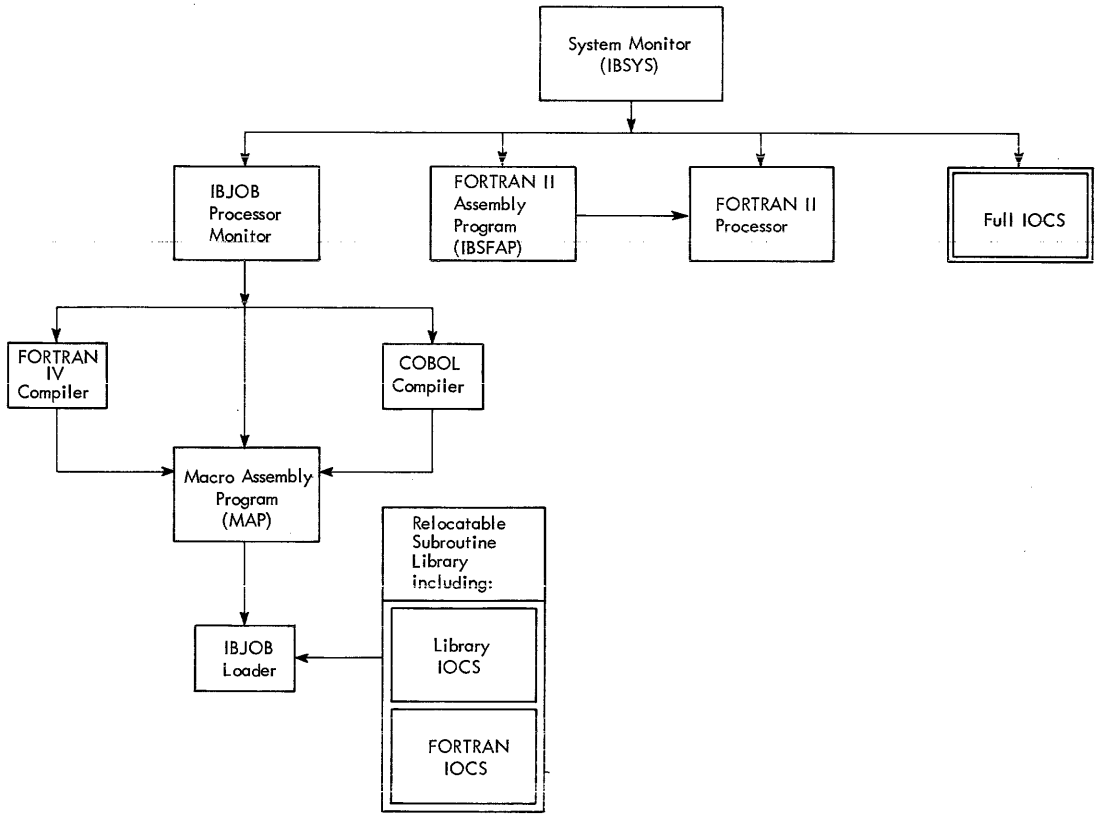


Figure 1. Relationship of Library IOCS, FORTRAN IOCS, and Full IOCS Within the IBM 7090/7094 IBSYS Operating System

Labels: This level contains all the IOCS sequential routines as well as routines for checking, creating, and writing standard IBM labels. Provision is made for processing nonstandard labels.

The IOCS routines for sequential processing apply to disk and drum storage as well as to magnetic tape units and unit record devices.

In using Library IOCS, the programmer can allow the Loader to determine which level is needed. The Loader examines the calling sequences written by the programmer and loads only that level of IOCS required to service the calling sequences.

Both forms of IOCS contain a set of routines for random processing of disk and drum storage files. IOEX must be in core storage when Random IOCS is used. However, Random IOCS does not use any of the other parts of sequential IOCS. Thus, Random IOCS can be used solely with IOEX, or can be combined with the Minimum, Basic, or Labels level of sequential IOCS.

The desired level of IOCS is specified in the SIBJOB control card (for Library IOCS) or the *JOB card (for Full IOCS). The manner in which the desired level is specified is discussed in sections of this publication on control cards.

Guide to Using This Publication

This publication has been organized for use in programming Library IOCS or Full IOCS.

The two forms differ primarily in (1) the required preparations for processing, and (2) control card formats and control information. A different section is provided in these two areas for the programmer using each form of IOCS.

It is recommended that the programmer interested in using Library IOCS read the major sections in the following order:

Basic Concepts
How the Library IOCS Programmer Prepares for Processing
IOCS Routines
IOCS Commands
Programming Examples
Any needed sections on labels, block sequence numbers and check sums, density and mode, checkpoints and restarts, and random processing
Control Card Information for Library IOCS

The programmer interested in using Full IOCS should read the major sections in this order:

Basic Concepts

How the Full IOCS Programmer Prepares for Processing
IOCS Routines
IOCS Commands
Programming Examples
Any needed sections on labels, block sequence numbers and check sums, density and mode, checkpoints and restarts, and random processing

Control and Loading Information for Full IOCS

The appendixes contain detailed information that will be helpful in gaining a more complete understanding of the system and in programming complex applications.

The Glossary near the end of this manual provides concise definitions of important terms used in this publication.

The following sections contain a discussion of terms and concepts necessary for understanding input/output activities and for grasping the full effect of subroutines and commands used in IOCS. Although some of these discussions are elementary, it is recommended that the sections be read to establish how each term is used in this publication.

Files

A *file* is a collection of related information, no matter where it is stored. It may be recorded on punched cards, magnetic tape, disk storage, drum storage, or some other recording medium.

The file does not consist of the input/output device itself. The file exists independently of the device and has its own logical beginning and end. It is important to maintain a clear distinction between the file and the input/output device.

Each device attached to a data processing system is usually assigned to a unit function, and is often referred to as *input/output unit.*

On a recording medium, a file resides as a series of *blocks,* or *physical records.* (This is shown in Figure 2.)

A block of information may be segmented into separate records. For example, a physical record may contain segments of information related to a number of different employees. Each segment is logically associated with one employee and is called a *logical record.* A block may consist of one or more logical records.

When a physical record is read into storage, it is held in a temporary storage area called a *buffer.* Output records are also retained in buffers until they can be recorded on an output unit. Each buffer contains exactly one physical record, although the record may not fill the buffer completely.

If the tape shown in Figure 2 were to be read into buffers, each buffer would contain a block consisting of three logical records.

Many data processing applications require that blocks of information be divided into separate logical records. Each logical record, or a portion of that record, is then processed individually. This method of breaking up blocks into logical records is called *deblocking.*

Once the information has been processed, the programmer may desire to conserve space on magnetic tape, disk storage, or drum storage by regrouping logical records into blocks before recording them. This process of grouping records in an output buffer is called *blocking.*

A file may extend beyond the physical limitations of the medium on which it is recorded. For example, a file may be contained on several reels of magnetic tape. When there is no more usable tape on a reel, the condition is called *end of reel.* However, the *end of file* does not occur until the last physical record associated with that file has been processed.

When a file extends beyond the recording capacity of one reel of magnetic tape, it is advantageous to switch from that unit to another unit without interruption. IOCS permits units to be coupled so that switching can be accomplished automatically.

Labels

A label is a physical record that identifies a file, or part of a file. Labels are recorded at the beginning and end of each segment of a file. The first label of each segment is the *header label,* and the second is the *trailer label.*

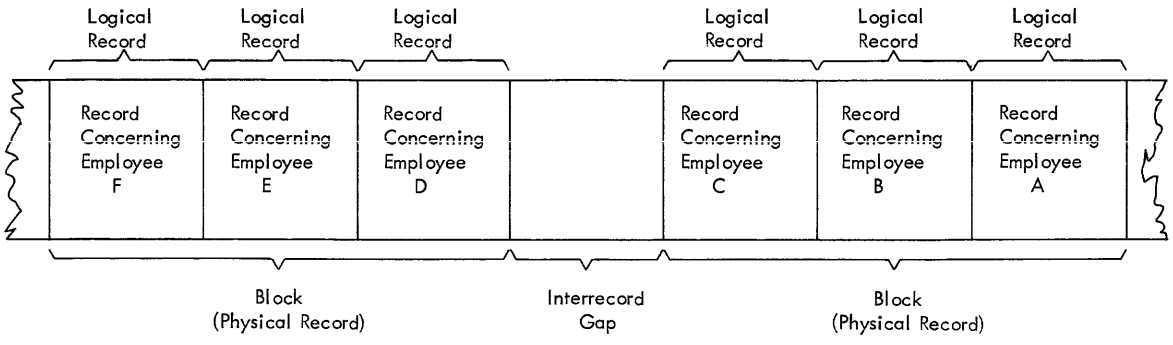


Figure 2. Logical Records Within Blocks on Magnetic Tape

If a file is contained on several reels, labels serve additional functions. In this case, labels are used to distinguish the various segments of the file. The header labels indicate the order in which the reels should be read, and the trailer label on each reel indicates whether or not end of file has been reached.

The trailer label on each reel before the last one is called an *end-of-reel trailer label.* The trailer label on the last reel is an *end-of-file trailer label.*

Movement of Data

Although IOCS provides many other services, its primary function is to move data into and out of the computer as specified by the programmer.

During processing, IOCS accomplishes the actual transmission of records from input unit to input buffer and from output buffer to output unit. In some cases, the data contained in an input buffer is processed directly within that buffer. At other times, the data may be moved from the input buffer to working storage and then from working storage to an output buffer.

NOTE: Throughout this publication, the actual transmission of data into and out of the computer is designated by the words read and write without quotation marks.

The movement of data inside the computer (from an input buffer to a work area, or from a work area to an output buffer) is accomplished with IOCS commands that are similar to IBM 7607 Data Channel commands. In using IOCS commands, a programmer tends to visualize consecutive buffers in the same manner in which he visualizes physical records on tape. For this reason, the concept of "reading" and "writing" to and from buffers is used.

NOTE: Throughout this publication, the words "read" and "write" in quotation marks are used to convey the idea of moving data from an input buffer to working storage, or from working storage to an output buffer.

Figure 3 is a schematic drawing that illustrates the distinction between the two stages of data movement. In Figure 3, stage A shows input data being read into input buffers. At stage B, the program is "reading" those records into a work area and processing them. Each record is then "written" into available output buffers at stage C, and finally written onto an output unit at stage D.

IOCS commands that actually move the contents of a buffer to a work area, or from a work area to a buffer, are called *transmitting commands.* Commands that provide information about contents of buffers, without actually moving any data, are called *non-transmitting commands.*

Buffer Pools and Overlap

In using IOCS, portions of core storage are structured as groups of buffers called buffer pools. The programmer can specify the number of buffers in each pool and the size of the buffers, but the buffers within a pool are always the same size.

Buffer pools are established in the following manner:

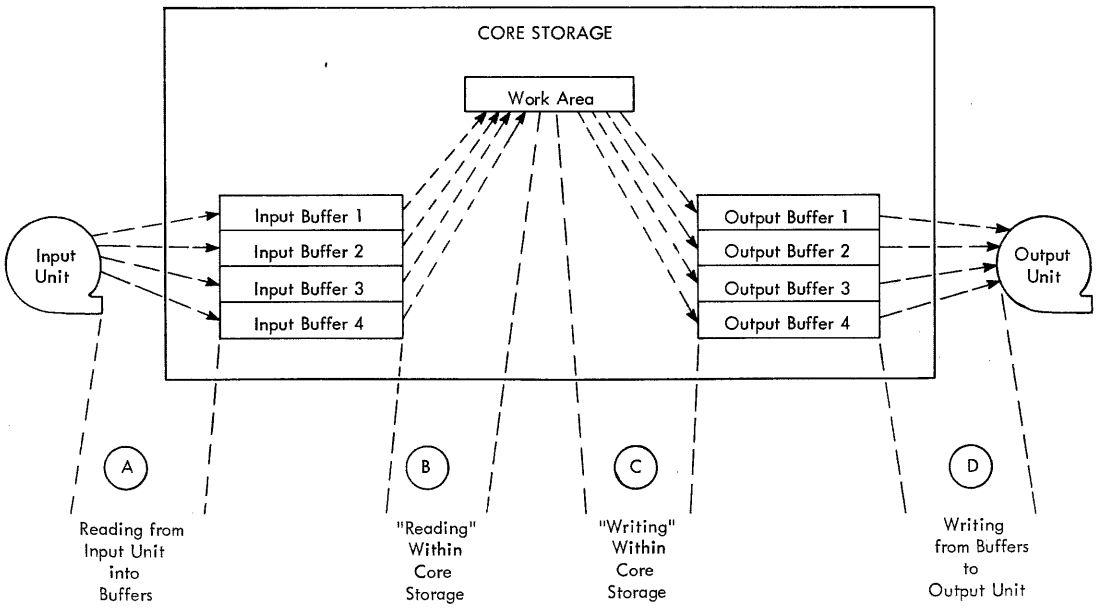


Figure 3. Stages of Data Movement When Transmitting Commands Are Used to Move Data Within Core Storage

1. Space is reserved for each of them.
2. Each pool is defined.
3. Files are attached to each pool.

The size of a pool is determined by the size of the buffers and by the number of buffers in the pool. In addition, the pool contains two pool control words and two control words for each buffer. Thus, if N equals the maximum number of data words a buffer is to hold, and M is the number of buffers in a pool, the size of the pool is $M(N+2)+2$.

Figure 4 is a schematic drawing of a buffer pool. The pool bears the symbolic name POOL and each buffer in the pool consists of N words. The first two words in the pool are *pool control words* and each buffer is preceded by two *buffer control words.*

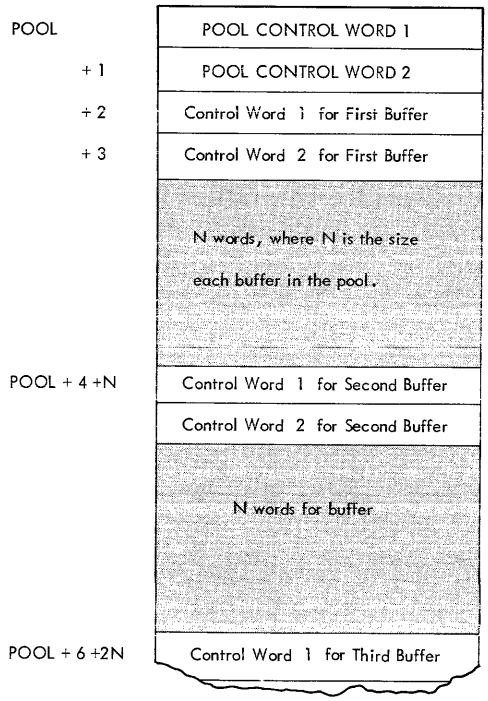


Figure 4. Structure of a Buffer Pool

The buffers in a buffer pool might be used by only one file. Frequently, however, several files share the buffers. This makes it possible for each buffer to be in use by one file during one part of the program and by another file at a different part of the program. Also, while two or more files are using a pool simultaneously, the available buffers are allocated to the files according to the frequency of physical transmission by each file.

For example, if 5 files are sharing a pool of 20 buffers, it is not necessarily true that 4 buffers would constantly be in use by each file. IOCS would permit the 20 buffers to be shared by the files according to increased or decreased demand by the individual files.

Buffer Cycles

A buffer within a pool may be used by IOCS to receive and hold input data at one moment, then used to hold output data at the next moment, then assigned again to an input file, then perhaps an output file, and so on. In this way, each buffer may be used at different times to hold input or output data.

The buffer always starts a cycle as an available buffer in the pool. As IOCS needs buffers, it picks up the location of each buffer and uses it to hold input or output data for a file. When the buffer is used for an input file, it can be thought of as an input buffer, although the input status is only temporary. In the same way, when the buffer is used for an output file, it can be thought of as an output buffer. In each instance, the buffer passes through several stages during processing.

Generally, the cycle for an input buffer includes the following stages:

1. At first the buffer is empty and is waiting to be filled with data.
2. The buffer is filled with data from an input unit.
3. The buffer waits for its data to be processed.
4. Then it is *in use* while the data is being processed.
5. Finally, the buffer is *released* and waits to be filled with new data.

The cycle for an output buffer includes similar stages, but the stages occur in a different order, as follows:

1. At first the buffer is empty and is waiting to be filled with data.
2. Then it is *in use*, being filled with processed information.
3. The buffer waits to have the data written on an output unit.
4. The data is written from the buffer and the buffer is *released* by IOCS.
5. The buffer waits to be filled with more data.

Control Words

Four types of control areas are used by IOCS to keep track of the status of buffers, buffer pools, files, and input/output units.

A programmer has little to do in establishing these areas or in changing any portion of them. IOCS provides for the contents of these areas to be established and maintained automatically.

The programmer should be aware of the control areas described in the following text and their relationship to each other.

Unit Control Blocks

A unit control block for each input/output unit attached to a computer system is located in the Nucleus of the System Monitor. Input/output activity for a file is accomplished by using this block.

File Control Blocks

For each file used in a program a file control block, consisting of 12 words, is established in core storage. It links the buffer pool used by the file to a unit control block.

For files on magnetic tape, the first six words in the file control block contain information about the characteristics of the file. Among other things, these six words contain the location of the unit control blocks associated with the file, the status of buffers in use by the file, and the location of a buffer pool control word. The other six words in the block contain labeling information.

For files on disk or drum storage, the first eight words are used for file and unit information, leaving only the last four words for labeling information.

The name assigned to the file by the programmer becomes the symbolic location of the first word of the file control block.

Appendix B contains a detailed diagram of the file control block.

Pool Control Words

The two pool control words at the beginning of each buffer pool provide information about the status of buffers for use by the file control blocks.

Buffer Control Words

The two buffer control words at the beginning of each buffer provide information about the status of each buffer while data is being processed.

During execution of a program, these control areas are continuously updated to reflect the current status of each unit and buffer area associated with a file.

Figure 5 shows the schematic relationship of these control areas to each other. In Figure 5, two files (A and B) are shown sharing a buffer pool. Four records from file A have already been read into buffers 4, 7, 2, and 5 (in that sequence); they are indicated by records A_n, A_{n+1}, A_{n+2}, and A_{n+3}, respectively. Three records are ready to be written out on file B; they are situated in buffers 3, 9, and 1, and are indicated by B_n, B_{n+1}, and B_{n+2}, respectively. Buffers 6 and 8 are empty.

The unit control block reflects the status of the unit in use by each file. The file control block for file A directs the program to the next buffer to be "read" into a work area, while the control words for other buffers being used by file A "chain" the buffers together in proper sequence for subsequent "readings." The file control block for file B provides a similar service for that file. Meanwhile, pool control words reflect the status of buffers in the pool.

More detailed information on the functions of buffer pool control words and buffer control words is contained in the publication *IBM 7090/7094 Input/Output Control System, Programming Systems Analysis Guide,* Form C28-6773.

NOTE: The programmer who intends to use Library IOCS should read the next section: "How the Library IOCS Programmer Prepares for Processing." The programmer who intends to use Full IOCS should skip the next section and should read the section: "How the Full IOCS Programmer Prepares for Processing."

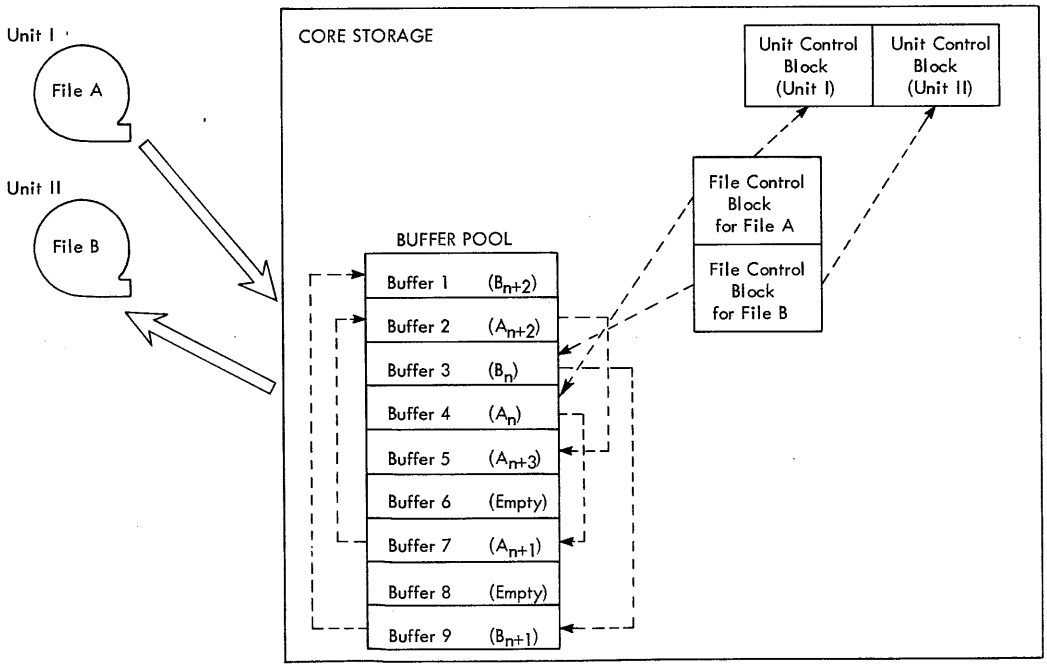


Figure 5. Schematic Relationship of Control Areas

The use of IOCS involves the following basic steps, divided into two major categories. These steps represent the broad phases of IOCS operation:

1. Preparation for Processing
   a. Files are described.
   b. Space in storage is assigned to buffer pools.
   c. Buffer pools are defined.
   d. Files are attached to each buffer pool.
2. Processing
   a. Files are opened. This starts the physical reading of records from an input unit and prepares an output file for output activity.
   b. Records are processed.
   c. Files are closed, ending input/output activity.

The incorporation of IOCS into the Relocatable Subroutine Library has reduced the amount of programming required to prepare for processing. In using Library IOCS, the programmer need only be concerned with the first preparatory step – the description of files. The IBJOB Loader reserves space for buffer pools and generates the calling sequences for defining pools and attaching files to them.

The programmer can control the assignment of files to buffer pools by using $POOL and $GROUP control cards. However, the use of these two cards is not required. The IBJOB Loader can perform all preparations for processing by using descriptions of files contained in the source program or in $FILE and $LABEL control cards.

File Description

The Macro Assembly Program provides two pseudo-operations that can be used to describe files within source programs. These are the FILE and LABEL pseudo-operations. The assembly program converts these pseudo-operations into $FILE and $LABEL control cards. If the file is unlabeled, only the FILE pseudo-operation need be used. Each labeled file, however, requires that a LABEL pseudo-operation for that file also be included in the source program.

The programmer may choose to describe his files at load time, rather than at assembly time. He may also wish to change certain options that were specified by FILE or LABEL pseudo-operations. In either case, he may use the $FILE and $LABEL cards.

File Types in Library IOCS

The Library IOCS programmer can use three major types of files: reserve file, internal file, or checkpoint file. Reserve and internal files can be designated as INPUT, OUTPUT, or INOUT. Each type is equivalent to a file type in Full IOCS, as shown in Figure 6.

The file types discussed in the following text are specified in the MAP language FILE pseudo-operation or in the $FILE control card.

RESERVE FILES

This type is not specified directly in a control card. The Loader automatically establishes all files, except internal and checkpoint files, as reserve-type files. The designation of INPUT, OUTPUT, or INOUT without specifying that the file is internal causes the Loader to classify the file as a reserve file. A reserve file must be opened before it can be processed. The specifications are as follows:

Input: This is an input file. Its characteristics are as follows:

a. It can be opened and closed as many times as desired.
b. It is neither rewound nor opened when attached.
c. Portions of its buffers can be "read" separately.

Output: This is a partial block output file. Its characteristics are as follows:

a. It can be opened and closed repeatedly during the program.
b. It is neither rewound nor opened when attached.
c. Portions of its buffers can be "written" separately.

Inout: This file can be an input file and a partial block output file at different times in a program. Initially, the file is set up as an input file. To change the file to an output file, the programmer must alter the configuration of bits 7 and 8 of Word 2 of the file control block for this file. The file type identification codes are shown in the following chart:

FILE CONTROL BLOCK WORD 2		
BIT 7	BIT 8	FILE TYPE
0	0	Input File
0	1	Output File

INTERNAL FILES (INT)

This is a file contained entirely within core storage. It can be given the characteristics of an INPUT, OUTPUT, or INOUT file, according to how it is to be used. An internal file must be opened before it is processed.

Types of Files		
Library IOCS		Equivalent File Type
Type	How Specified	In Full IOCS
	(Immediate Files cannot be specified in Library IOCS.)	Immediate
Reserve	(All files are reserve files unless internal or checkpoint is specified.)	Reserve
a. INPUT / INOUT †	File usage option in the FILE pseudo-operation	a. Input
b. OUTPUT / INOUT †	File usage option in the FILE pseudo-operation	b. Partial block output
INT	Unit assignment option in the FILE pseudo-operation	Internal
a. INPUT / INOUT †	File usage option in the FILE pseudo-operation	a. Input
b. OUTPUT / INOUT †	File usage option in the FILE pseudo-operation	b. Partial block output
CHECKPOINT (or CKPT)	File usage option in the FILE pseudo-operation	Checkpoint

† See discussion of INOUT option.

Figure 6. File Types in Library IOCS

CHECKPOINT FILES (CHECKPOINT OR CKPT)

This is a checkpoint file used for recording checkpoint information. A checkpoint file uses no buffers and is not attached to any buffer pool. However, it must be opened before it is used. A checkpoint file is not designated as input or output.

Functions of the IBJOB Loader

Using the descriptions provided in the control cards the IBJOB Loader performs the following preparatory functions:

Establishes File Control Blocks

The Loader reserves 12 words and sets up a file control block for each file described in a $FILE card or FILE pseudo-operation. The name of the file becomes the symbolic location of the first word of the file control block.

Assigns Space for Buffer Pools

The Loader assigns all core storage not used by the object program or by system programs for use as buffer pools. If $FILE cards or FILE pseudo-operations are not present or if the trap supervisor (IOEX) is specified on the $IBJOB control card, the Loader does not assign any storage as input/output buffers. Normally the size of the object program leaves enough space for the Loader to assign many buffers to each pool. Unless the $POOL card is used, all files with the same block size will be assigned to the same pool.

Generates .DEFIN and .ATTAC Calling Sequences

The .DEFIN calling sequence causes transfer to an IOCS routine that sets up the structure of a buffer pool. The .ATTAC calling sequence causes transfer to a routine that attaches a group of files to a specific pool. The Loader generates all of the needed .DEFIN and .ATTAC calling sequences and places them in front of the object program. The form and specific effect of these two calling sequences are described in the section "How the Full IOCS User Prepares for Processing."

Functions of $POOL and $GROUP Cards

By using the $POOL card, it is possible to assign files with different block sizes to the same pool. In this case,

the size of all buffers in the pool will equal the largest block size of a file in the pool. The $POOL card contains options that allow the programmer to specify block size (BLOCK) and buffer count (BUFCT). The buffer count in this card establishes the number of buffers to be assigned to the pool.

It may be desirable to group reserve-type files together into what are known as Reserve Groups. This is accomplished by using the $GROUP card. By establishing a Reserve Group, it is possible for a programmer to make more efficient use of the number of buffers he has available. Use of a $GROUP card allows a programmer to specify:

1. The number of buffers in the pool that are to be reserved for the use of the files in the Reserve Group (BUFCT), and
2. The number of files in that group that will be open at the same time (OPNCT). This allows the reserve files to share buffers more efficiently.

For example, if ten buffers have been reserved for two files in a Reserve Group and both files are open at the same time, five buffers are always available for use by each file. But if the programmer indicated that only one file in the Reserve Group would be open at a given time, each file would be allowed to use all ten buffers whenever the file was open. This would make more buffers available to each file when it was open and would increase the potential overlap.

Using OPNCT and BUFCT, IOCS internally controls the number of buffers it can allow each file in a Reserve Group to use at any given time.

Block Size and Activity Options

Two options available in the FILE pseudo-operation and the $FILE card are discussed briefly here because of their effect on the allocation of storage for buffer pools.

Block Size Option

The block size option is an integer (0000-9999) that specifies the maximum size of the physical records for a reserve or internal file. The block size can be omitted if the file is included in a $POOL or $GROUP card that contains a block size specification. If block size is not specified for a file in the FILE pseudo-operation, the Macro Assembly Program assumes a block size of 14 for BCD or MXBCD files and a block size of 256 for BIN or MXBIN files.

Activity Option

The activity option is an integer (00-99) that specifies relative activity of this file in comparison to other files. The programmer chooses a number from a low of 00 to a high of 99 to indicate the relative activity of the file. If there is a shortage of core storage, this option can be used to have a higher number of buffers assigned to the most active files.

Core Storage Assignment Without $POOL and $GROUP Cards

When $POOL and $GROUP cards are not used, the Loader makes input/output storage assignments as follows:

1. Creates a separate pool for each different blocking size encountered in the $FILE cards and assigns all files (input and output) that have the same block size to the same pool.
2. Reserves storage for each pool, using the following procedure:
   a. Gives the pool one buffer for each file. If available storage does not allow this, the Loader stops execution and prints an error message for the operator.
   b. Gives the pool one additional buffer for each file. If available storage does not allow this, the Loader forms a weighing factor for each pool. This weighing factor is formed by multiplying the desired number of additional buffers for the pool by the total of activity numbers (activity options in $FILE card) of all files assigned to the pool. The Loader then compares the pool weighing factors. The pool with the largest weighing factor then receives one additional buffer, if possible. If this is not possible, the weighing factor of that pool is made zero. If the assignment is possible, the weighing factor of the pool is reduced. The pool that now has the largest weighing factor is given an additional buffer, and this process continues until all weighing factors have been reduced to zero.
   c. Gives remaining storage, if any, to the pools in proportion to their output activity. The relative output activity of a pool is the ratio of the sum of output file activity numbers in the pool to the total of file activity numbers of all output files.

The amount of storage used by each buffer pool is:

BUFCT*(BUFSIZ+2) + 2

Core Storage Assignment with $POOL and $GROUP Cards

The $POOL and $GROUP cards override the normal Loader input/output storage assignments. The effect of these control cards is discussed in the following sections.

$POOL Card

A $POOL card causes all files mentioned on the card to be assigned to the same buffer pool. No other files (not

even those with block sizes equal to pool block size) will be assigned to the pool. If any of the files in the $POOL card also appear in a $GROUP card, all files in the $GROUP card are automatically assigned to the $POOL card pool.

If a buffer count (BUFCT) appears in the $POOL card, the pool will have exactly that number of buffers. The Loader checks the buffer count to ensure that it is at least as large as the number of nongrouped files in the pool, plus the buffer counts of all groups in the pool. If no buffer count appears in the $POOL cards, the Loader allocates buffers in the same manner as it would if no $POOL card were present.

If no block size (BLOCK) appears in the $POOL card, the pool block size will be equal to the largest block size found among the files assigned to that pool. The block size check option in the FILE pseudo-operation must have been used if files of different block size are specified in the $POOL card.

$GROUP Card

A $GROUP card causes the files mentioned in the card to be formed into a Reserve Group. If a buffer count (BUFCT) is specified, this count is used, and it must be at least equal to the open count (OPNCT). If BUFCT does not appear in the card, the Loader will assign extra buffers to the pool with which the group is associated in the manner described in step 2c of the section "Core Storage Assignment Without $POOL and $GROUP Cards." If OPNCT does not appear in the $GROUP card, the Loader assumes that the count is equal to the number of files in the group.

.DEFIN and .ATTAC Routines

Exceptional circumstances may prompt the Library IOCS programmer to define buffer pools and attach files to them in the same manner as the Full IOCS programmer. Library IOCS contains a .DEFIN routine and an .ATTAC routine that can be used for this purpose. These routines are equivalent to the DEFINE and ATTACH routines discussed in the next section. It is emphasized that use of .DEFIN and .ATTAC is exceptional.

NOTE: The Library IOCS programmer should skip the next section and should read the section "Processing with Both Forms of the Input/Output Control System (IOCS)."

Each step in the preparations for processing serves a specific function in setting up file control blocks and buffer pools and in linking them to input/output units.

In generalized form, the preparations for processing involve the following:

1. Files are described.
2. During assembly, space is reserved for each buffer pool.
3. Buffer pools are defined.
4. Files are attached to buffer pools.

Once these preparations have been completed, the programmer can open files and process data. Opening a file actually starts the physical reading of records from an input unit or prepares an output file for output activity. When a programmer is finished using a file, he closes it, ending input/output activity.

File Description

The Full IOCS programmer is required to describe each file in an *FILE control card. In addition, he must provide a list, or lists, of files within his source program.

The *FILE card allows such specifications as input/output units, file type, mode, density, and labeling conventions. This information is used to establish portions of the file control block.

Detailed information regarding the options in the *FILE card is provided in the section entitled "Control and Loading Information for Full IOCS."

File Types in Full IOCS

Four types of files can be specified by the Full IOCS programmer according to the manner in which each file will be used in the program. They are: Immediate, Reserve, Internal, and Checkpoint.

In addition, Full IOCS permits three subclassifications according to how data in each buffer are to be processed. They are: Input, Partial Block Output, and Total Block Output.

The file types and the manner in which they are specified are indicated in Figure 7. The restrictions on using each file type are described in the following text.

File Types in Full IOCS

Type	How Specified
Immediate	PZE entry in file list preceding Reserve files
a. Input	I in column 28 of *FILE card
b. Partial Block Output	P in column 28 of *FILE card
c. Total Block Output	T in column 28 of *FILE card
Reserve	SVN entry in file list
a. Input	I in column 28 of *FILE card
b. Partial Block Output	P in column 28 of *FILE card
Internal	SIX entry in file list and INT entry in columns 18–20 of *FILE card
a. Input	I in column 28 of *FILE card
b. Partial Block Output	P in column 28 of *FILE card
Checkpoint	C in column 28 of *FILE card

Figure 7. Types of Files in Full IOCS

IMMEDIATE FILES

An immediate file is an input/output file that can be processed only once during the course of a program. It is automatically rewound and opened when attached. Once this type of file has been closed, it cannot be reopened.

RESERVE FILES

A reserve file is an input/output file that can be opened and closed as many times as desired. It is neither rewound nor opened when attached.

A reserve file can be designated in the file list in such a way that a specified number of buffers will always be available for that file. The file list is provided by the programer within his source program. It is described in the section "Setting Up File Lists."

INTERNAL FILES

An internal file is a file that is kept entirely within core storage and has no input/output unit associated with it. Internal files have much the same character-

istics as reserve files, and the programmer processes them in a similar manner. An internal file must be opened before it is processed.

CHECKPOINT FILES

A checkpoint file uses no buffers and, consequently, is not attached to any buffer pool. It must, however, be opened before it is used.

A checkpoint is a point in a program at which the full contents of storage and the status of registers and indicators are recorded. This information is used later to restart the program from the point at which it was stopped. A full discussion of checkpoints and restarts is provided in the section "Checkpoints and Restarts."

File Subtypes in Full IOCS

The designation of a file as immediate, reserve, or internal provides only a partial description of how the file will be processed. One of the following subtypes must also be designated in Full IOCS in order to establish how IOCS will permit "reading" and "writing" to and from its buffers.

The buffer requirements cited in the descriptions of the subtypes are the absolute minimum and will not permit overlap. At least two buffers for each file are required to achieve overlap with input and partial block output files. At least three buffers for each input and partial block ouput file must be provided if one buffer is to be processed with a nontransmitting command. No overlap is possible with total block output files.

The three subtypes and their minimum buffer requirements are as follows:

INPUT FILES

This is the category for all input files. Each file requires at least one buffer. At least two buffers are required if a nontransmitting command is used to locate or skip words within a buffer.

PARTIAL BLOCK OUTPUT FILES

This type of output file permits the "writing" of any number of machine words, regardless of buffer size. The programmer can fill buffers in a piecemeal fashion, thus blocking his output records. Every partial block output file requires at least one buffer. At least two buffers are required if a nontransmitting command is used to locate or skip words within a buffer.

TOTAL BLOCK OUTPUT FILES

The total block output files are allowable with immediate files only. This type of output file requires that all words to be placed in a buffer be "written" using one command sequence.

Since this type of file does not withhold a buffer from the pool, except when the file is in use, all files of this type could use the same buffer at different times. Thus, all total block output files attached to the same buffer pool could have one buffer to serve all of them.

The total block output file is used infrequently. It can best be used for files in which records are "written" intermittently. For example, a total block output file might be used to record obsolete records encountered occasionally in a master file update run.

Functions of Reserve Groups

Full IOCS allows the programmer to group reserve files together into what are known as Reserve Groups. By establishing such a group, it is possible for a programmer to make more efficient use of the number of buffers he has available.

The distinguishing characteristic of a reserve file is that it can be opened and closed repeatedly during the program. When a reserve file is closed, that file uses no buffers because there is no input/output activity. The only time the file needs buffers is when it is open.

In designating a Reserve Group, the programmer specifies (1) the number of buffers in the pool that are to be reserved for the exclusive use of the files in the Reserve Group (*bufct*), and (2) the number of files in that group that will be open at the same time (*opnct*). This allows the reserve files to share buffers.

For example, if ten buffers have been reserved for two files in a Reserve Group and both files are open at the same time, five buffers are always available for use by each file. However, if the programmer indicated that only one file in the Reserve Group would be open at a given time, each file would be allowed to use all ten buffers whenever it was open. This would make more buffers available to each file when it was open and would increase the potential overlap.

Using *opnct* and *bufct*, IOCS internally controls the number of buffers it can allow each file in a Reserve Group to use at any given time.

A second function of the Reserve Group is to provide the programmer with a method of ensuring that a given number of buffers will always be available for a file. This is especially important if the program includes nontransmitting IOCS commands that withhold several buffers from a pool at the same time.

The use of the Reserve Group can help conserve storage and maximize overlap.

Preparations for Processing

The full IOCS programmer must program each of the steps in the following text before he opens files and begins processing data records.

Establishing Reference Locations to Full IOCS Subroutines

IOCS consists of a set of subroutines to which the programmer establishes linkage in calling sequences. Within his source program, the Full IOCS programmer must define the transfer point to each routine. A full listing of the relative locations of IOCS subroutines is provided in the section "Control Information for Full IOCS."

Reserving Space for File Control Blocks

Twelve consecutive words in storage must be reserved for the file control block for each file. The 12 words may be reserved with BSS pseudo-operations such as the following:

```
1          8          16
MASFIL     BSS        12        MASTER FILE
DETFIL     BSS        12        DETAIL FILE
```

where MASFIL and DETFIL are the names of the files. During assembly, they become the symbolic location of the first word of each file control block. The file control blocks for all files used in the program must be contiguous. The number of file control blocks to be generated by the Preprocessor must be specified in the *JOB card. The origin (in octal) of the first file control block can also be specified in the *JOB card.

Reserving Space for Buffer Pools

The formula used in determining the number of words required for each buffer pool is:

$$M(N+2)+2$$

where M is the number of buffers to be included in the pool, and N is the number of data words in each buffer. The formula includes space for the two pool control words for the pool and the two buffer control words for each buffer. If each physical record is to include a check sum/block sequence word, the value of N must be increased by 1.

To reserve space for a pool of eight buffers in which each buffer will contain 30 words, the following BSS pseudo-operation can be used:

```
1          8          16
POOL       BSS        8*30+8*2+2
```

The programmer can provide any number of buffers in the pool, but that number must be sufficient to meet the minimum requirements of the list of files to be attached to the pool.

Setting Up File Lists

Within his source program, the programmer must provide a list of files that will use the buffers in each buffer pool. The format of a file list is shown in Figure 8, where:

filist
  is the symbolic name of the list.

imfil1 through imfilx
  represent the names of immediate files.

rsfil1 through rsfilx
  represent the names of reserve files.

infil1 through infilx
  represent the names of internal files.

The rules for composing a file list are as follows:

1. The operation field preceding the name of each file must contain PZE.

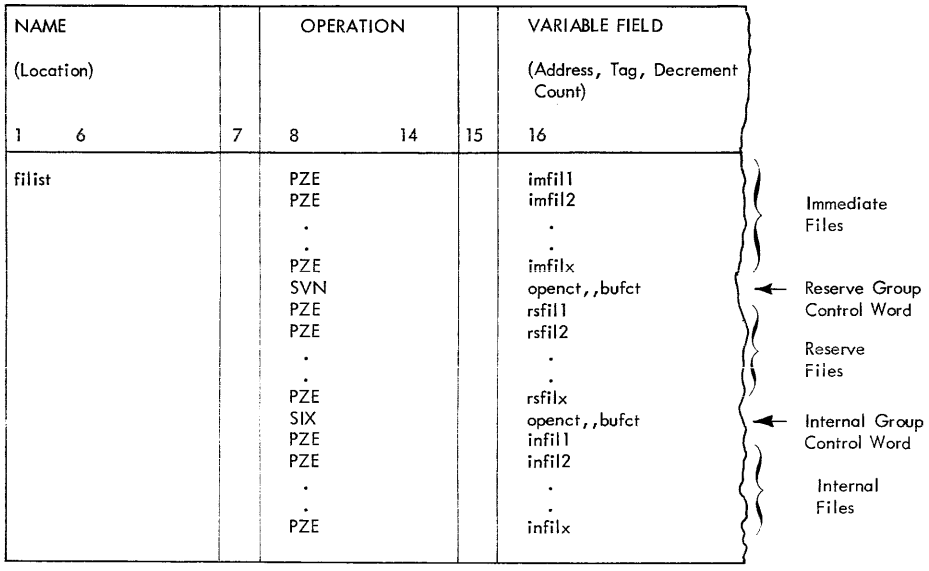


Figure 8. File List Format

2. All immediate files must be listed before reserve files and internal files. Immediate files have no control word, and they are listed consecutively before reserve files.

3. Reserve files are listed in groups which must be preceded by a Reserve Group Control Word in the following format:

```
1       8       16
        SVN     openct,,bufct
```

where:

SVN
is the prefix that distinguishes a Reserve Group Control Word (sometimes called the Seven Word).

openct
is the maximum number of files that will be open at the same time in the Reserve Group.

bufct
is the number of buffers to be reserved within the pool for the use of these reserve files.

A Reserve Group may contain only one reserve file.

4. Internal files are listed in groups which must be preceded by an Internal Group Control Word in the following format:

```
1       8       16
        SIX     openct,,bufct
```

where:

SIX
is the prefix that distinguishes an Internal Group Control Word (sometimes called the Six Word).

openct
is the maximum number of files that will be open at the same time in the Internal Group.

bufct
is the number of buffers to be reserved within the pool for the use of these internal files.

A file list used to attach several files to a pool simultaneously is usually preserved, since the same list can be used later when the files are closed.

Any file list that contains one or more Internal or Reserve Groups must be preserved until all such groups in the list have been closed. This is necessary because IOCS reflects the status of a group during processing by modifying the open count and buffer count portions of the group control words.

USE OF RESERVE GROUPS

Buffer count and open count allow the programmer to specify the exact number of buffers that will be withheld from the pool at any given time. With these specifications, IOCS automatically allows files to use a maximum number of buffers at all times during execution of the program.

Two examples of how these characters can benefit the program are discussed below.

Deferred Opening of a File: It might be known that a file will not be used until late in the program or might not be used at all if abnormal conditions are not encountered. By placing this file in a Reserve Group, buffers would remain free for use by other files in the group until such time as the special file were opened.

For example, assume that FILEA and FILEB are the respective names of two files that will never be open at the same time. They could be specified in a Reserve Group, as follows:

```
1       8       16
        SVN     1,,1
        PZE     FILEA
        PZE     FILEB
```

This signifies that only one file will be open at any time and that one buffer is to be reserved for both files.

Using a File for Both Input and Output: In some instances, a file is used for both input and output. It is not unusual, for example, to create an intermediate tape as output from one phase of a problem, and then to use this tape as input during a later phase.

One way to program this is to create separate file control blocks: one for the input phase, and the other for the output phase. The contents of the two file control blocks will be identical, except that one file is described as an input file and the other is described as an output file. If the input file were named IN and the output file were named OUT, they could be specified in the file list as a Reserve Group, as follows:

```
1       8       16
        SVN     1,,1
        PZE     IN
        PZE     OUT
```

Only one buffer will be withheld from the pool because IOCS recognizes that only one file at a time can be open. The file can be manipulated logically as two files because separate file control blocks have been established.

The OUT file would be opened and used during any output phase, while the IN file would be closed. The reverse would apply during an input phase.

Defining Buffer Pools (DEFINE)

The DEFINE routine in Full IOCS is used to set up the structure of each buffer pool. The form of the DEFINE calling sequence is as follows:

```
        TSX     DEFINE,4
        PZE     pool
        PZE     m,,n
```

where:

pool
is the symbolic location of the first word of the space reserved for this buffer pool.

m
is the number of buffers in the pool.

n
is the size of each buffer.

NOTE: The letters *m* and *n* should be the values that were used when space was reserved for the pool. This

is described in the section "Reserving Space for Buffer Pools."

Each buffer pool for which space has been reserved must be defined before the pool is to be used in the program. However, the pool may be defined at any point in a program.

JOINING POOLS (JOIN)

Two previously defined buffer pools can be joined at any point in the program by using the JOIN routine. The pools need not be at continguous locations.

The form of the JOIN calling sequence is:

```
TSX     JOIN,4
PZE     pool1,,pool2
```

This sequence causes the previously defined pools at symbolic locations *pool1* and *pool2* to be treated as one pool. *Pool1* becomes the symbolic location of the first control word of the enlarged pool.

The rules for using the JOIN calling sequence are:

1. The buffers of both pools must be the same size.
2. If *pool2* has been used by any IOCS routine other than DEFINE, *pool2* must be redefined before it is joined to *pool1*.
3. *Pool1* may have files attached to it when the pools are joined.
4. *Pool2* cannot subsequently appear in a calling sequence.

Attaching Files to Buffer Pools (ATTACH)

Every file, except a checkpoint file, must be attached to a buffer pool before it can be opened. Attaching files to pools and opening them is called file initialization. The calling sequence for opening files is discussed in the next section. The requirements for different file types are as follows:

1. Immediate files are opened automatically by IOCS when the programmer attaches them. An attempt to open an immediate file will be ignored.
2. Reserve and internal files must be attached and opened.
3. Checkpoint files are not attached, but they must be opened.

Files are attached to a pool on the basis of the file list that was established for that pool. All files in a list must be attached in one ATTACH calling sequence. The form of the calling sequence is:

```
TSX     ATTACH,4
PZE     pool
PZE     filist,,number
```

where:

pool
: is the symbolic location of the first pool control word. (The symbolic name is the same as that used to reserve space and to define the buffer pool.)

filist
: is the symbolic location of the first word of the list of files that is to be attached to the pool.

number
: is the number of files (in decimal) contained in the list, not counting the group control words.

RE-USING A BUFFER POOL

When all of the files attached to a buffer pool have been closed, the structure of the pool is essentially the same as it was when first defined. A new list of files can then be attached to it. Any Reserve or Internal Groups attached to the pool must also have been closed before a new list is attached.

IOCS Routines

The processing routines discussed in this section serve the same functions in Library IOCS and Full IOCS. For simplicity, the name of the Library IOCS is used in the text. The explanations, however, are equally applicable to Full IOCS. Figure 9 shows the symbolic names of the routines in both forms of IOCS and indicates the level at which they are available.

The formats of the calling sequences are displayed side-by-side. These formats are the same, except for the name of the routine. The format for Library IOCS appears on the left side of the column, and the format for Full IOCS appears on the right.

The forms of the calling sequences for files on IBM 729 Magnetic Tape Units, IBM 7340 Hypertape Drives, IBM 1301 Disk Storage Units, and IBM 7320 Drum Storage Units are identical, except where differences are stated in the explanations.

Opening Files

All files, except immediate files, must be opened before they can be processed. Opening a file prepares it for input/output activity.

SYMBOLIC NAMES OF IOCS ROUTINES

Level At Which Available	Library IOCS	Full IOCS	Function
IOEX	.MWR	.MWR	Printing on-line
Minimum	.ATTAC (Loader generates the calling	ATTACH	Attaching files to buffer pools
	.DEFIN sequences to these two routines.)	DEFINE	Defining buffer pools
	.OPEN	OPEN	Opening files
	.CLOSE	CLOSE	Closing files
	.READ	READ	"Reading" data words
	.WRITE	WRITE	"Writing" data words
	.BSR	BSR	Backspacing a record
(These two routines are available at the minimum level if IOCS has been assembled for disk storage.)	.READR	READR	Retaining a buffer
	.RELES	RELESE	Releasing a retained buffer
Basic	.BSF	BSF	Backspacing over one or more files
	.CKPT	CKPT	Initiating a checkpoint
	.COPY	COPY	Copying buffers
	.JOIN (Normally not used)	JOIN	Joining buffer pools
	.REW	REW	Rewinding a unit
	.STASH	STASH	Transferring buffers
	.WEF	WEF	Writing a file mark
Labels	(All automatic label checking and preparation routines)		
Random	.RANCL .RANDE .RANRE .RANRP	RANCLS RANDEF RANREQ RANRPL	(These routines are discussed in the section "Routines in Random IOCS.")
	.RANCA .RANFL	RANCAV RANFLG	(These flag words also are discussed in the section "Routines in Random IOCS.")

Figure 9. Symbolic Names of IOCS Routines

The .OPEN routine accomplishes the following:

1. Positions the unit to be used by the file as specified in the calling sequence
2. Performs label checking and preparation, if labels are specified
3. Initiates reading of records from an input unit
4. Prepares an output file for output activity

Opening Reserve Files

Each reserve file must be opened individually with the following calling sequence:

LIBRARY IOCS		FULL IOCS	
TSX	.OPEN,4	TSX	OPEN,4
pfx	file	pfx	file

where:

file
is the name of the file (the symbolic location of the first word of the file control block for this file).

pfx
controls rewind, label action, and file protection. The options are:

= PZE	Rewind before opening
= MZE	No rewind
= MON	No rewind and no label action
= PTW	(Hypertape) Rewind and file protect before opening
= SIX	(Hypertape) File protect but no rewind
= SVN	(Hypertape) File protect but no rewind or label action

Care should be taken in specifying rewind for a System Input Unit, System Output Unit, or System Peripheral Punch Unit because of the possibility of processing the wrong input file or of destroying an output file.

On opening an input file on an IBM 7340 Hypertape Drive, IOCS checks the decrement of the parameter statement of the calling sequence to determine whether the file should be read forward or backward. If reading backward is desired, a 1 must be placed in the decrement of the calling sequence, as follows:

LIBRARY IOCS		FULL IOCS	
TSX	.OPEN,4	TSX	OPEN,4
pfx	file,,1	pfx	file,,1

The records obtained by reading backward can be processed in the same manner as the records obtained by reading forward. The presence of the 1 in the calling sequence for a file on an IBM 729 Magnetic Tape Unit causes the tape to be rewound before processing is begun.

Opening Internal Files

Each internal file also must be opened before it is "read" or "written." The calling sequence for opening an internal file differs from the sequence used for opening a reserve file, in that the prefix of the parameter statement indicates the type of processing desired.

Thus, in the following sequence:

LIBRARY IOCS		FULL IOCS	
TSX	.OPEN,4	TSX	OPEN,4
pfx	file	pfx	file

the prefix specifies processing as follows:

pfx

= PZE	The file is prepared for "writing," and any information in the file is discarded. This is analogous to opening a partial block output file with rewind.
= MZE	The file is prepared for "writing," and any information within it is retained. This is analogous to opening a partial block output file without rewind.
= PON	The file is prepared for "reading," making any information within it available. When the file is "read," buffers that have been processed are released to the pool exactly as they are for reserve files. The contents of the buffers can no longer be secured. This mode of operation can be thought of as "destructive reading."
= MON	The file is prepared for "reading," making any information within it available for processing. When this prefix is used, buffers that have been processed are retained and can be processed again. This mode of operation can be thought of as "regenerative reading."

Opening Checkpoint Files

A checkpoint file must be opened, even though it is never attached to a buffer pool. IOCS keeps track of which checkpoint file was last opened and always uses that as the current checkpoint file. The calling sequence is as follows:

LIBRARY IOCS		FULL IOCS	
TSX	.OPEN,4	TSX	OPEN,4
pfx	ckfil	pfx	ckfil

where:

ckfil
is the name of the file (the symbolic location of the first word of the file control block).

pfx
has the same meaning as for reserve files.

Data Movement Routines

Data is moved within storage by using the .READ, .WRITE, .copy, and .STASH routines. IOCS provides two methods of processing data contained in input buffers. One method is to transmit the data from the buffers to working storage; the other is to process the data within the buffer by determining the position of the desired data.

The .READ and .WRITE calling sequences contain a list of one or more IOCS commands that control the "reading" and "writing" of data.

In the following explanations, IOCS commands are represented by:

IOxy A,,n

The *x* and *y* represent parameters that control the manner in which the command is executed, and determine the action IOCS will take when it has completed executing the commands; *A* is an address at which the command starts operating; and *n* is the number of words involved in the operation. The exact effect of individual commands will be discussed in the section "Input/Output Control System (IOCS) Commands."

"Reading" Buffers

A .READ calling sequence can be used to do the following:

1. Transmit words sequentially from input buffers to working storage
2. Locate the address of the next word within an input buffer
3. Skip words in an input buffer

The form of the calling sequence is as follows:

LIBRARY IOCS			FULL IOCS	
TSX	.READ,4		TSX	READ,4
PZE	file,,eob		PZE	file,,eob
PZE	eof,,err		PZE	eof,,err
IOxy	A,,n	IOCS commands	IOxy	A,,n
.	. ,,.		.	. ,,.
.	. ,,.		.	. ,,.

where:

file
is the name of the file (the symbolic location of the first word of the file control block).

eob
is the end-of-buffer switch. This may be 0 or an address. The effect of this switch is discussed in the section "Nontransmitting Commands."

eof
is the symbolic location to which the program should transfer when the end of this file is encountered.

Labeled File – For a labeled file, end of file is recognized from a LEOF trailer label.

Unlabeled File – For an unlabeled file, any file mark is recognized as end of file.

Internal File – End of file occurs on an internal file when all the buffers reserved for the Internal Group, as specified by BUFCT in the group control word, have been "read."

Disk or Drum File – For the end-of-file exit to function properly with a disk or drum storage file, the file must have been previously created by using IOCS or the System Editor.

err
is the location to which transfer will be made if any one of the following error conditions occurs:

Parity error that cannot be corrected
Check sum error (binary file)
Sequence error
Parity or cyclic check (disk or drum file)
Data compare check (disk or drum file)
Data check (Hypertape file)

The error condition is recognized at the first reference in an IOCS command to a buffer in which it occurs. If the error exit is taken, no words are "read" by the command at which the error is detected. The record can be processed as read by continuing to "read" the file.

The command list in a .READ calling sequence is terminated by the first IOCS terminating command (IOXT, IOXTN) or disconnecting command (IOCD, IOCDN) encountered.

At the completion of a .READ or .WRITE calling sequence, the AC and MQ contain a report, called a "history record," on the results of the sequence. This is discussed in the section "History Records." Appendix A indicates the words in the IOCS communication region from which history record information can also be secured.

Locating with Retention and Releasing a Retained Buffer

These two routines can be used only if IOCS has been assembled for IBM 1301 Disk Storage Unit – or IBM 7320 Drum Storage Unit – capability. These routines are used primarily for random processing with disk or drum storage. However, they can be used to locate and retain buffers during sequential processing. This might be desirable if a programmer intends to refer to the contents of a buffer repeatedly during processing.

To locate a buffer and retain it, the calling sequence is as follows:

LIBRARY IOCS		FULL IOCS	
TSX	.READR,4	TSX	READR,4
PZE	file,,eob	PZE	file,,eob
PZE	eof,,err	PZE	eof,,err
IORTN	**,**	IORTN	**,**

where *file, eob, eof,* and *err* have the same meanings as in the .READ calling sequence. When using this routine, the file from which the buffer is retained must be the only file attached to a buffer pool.

This routine allows retention of a buffer until it is released by the user's program, overriding the normal process whereby the buffer is returned to the pool after the next reference to the file.

A buffer located with retention must be released by using the following calling sequence:

LIBRARY IOCS		FULL IOCS	
TSX	.RELES,4	TSX	RELESE,4
PZE	lbuff,,file	PZE	lbuff,,file

where:

file
is the name of the file (the symbolic location of the first word of the file control block for this file).

lbuff
is the starting location of the buffer inserted into the IORTN command when the buffer was located and retained with a .READR calling sequence.

The use of these commands in random processing is discussed in the section "Routines in Random IOCS."

"Writing" Buffers

A .WRITE calling sequence can be used to do the following:

1. Transmit words from working storage to output buffers

2. Locate empty words within an output buffer into which processed data can be placed.

The form of the calling sequence is as follows:

```
LIBRARY IOCS                                  FULL IOCS
TSX     .WRITE,4                              TSX     WRITE,4
PZE     file,,eob                             PZE     file,,eob
IOxy    A,,n                                  IOxy    A,,n
.       . ,,.        IOCS                     .       . ,,.
.       . ,,.      commands                   .       . ,,.
.       . ,,.                                 .       . ,,.
```

where:

file
is the name of the file (the symbolic location of the first word of the file control block).

eob
is the end-of-buffer switch. This may be 0 or an address. The effect of this switch is discussed in the section "Nontransmitting Commands."

Internal Files – For an internal file, the end-of-buffer exit is taken when all the buffers indicated by *bufct* in the Internal Group Control Word have been filled.

The command list in a .WRITE calling sequence is terminated by the first IOCS terminating command (IOXT, IOXTN) or disconnecting command (IOCD, IOCDN) encountered.

At the completion of a .WRITE or .READ calling sequence, the AC and MQ contain a history record as explained in the section entitled "History Records." Appendix A indicates the words in the IOCS communications region from which history record information can also be secured.

Copying Input Buffers

The .COPY calling sequence can be used to transfer data from input file buffer(s) to output file buffer(s). Before words can be copied, they must have been located by using an IOCS nontransmitting command. No actual word transmission occurs. Instead, the input buffer(s) is appended to the output file buffer chain.

The form of the calling sequence is as follows:

```
LIBRARY IOCS                FULL IOCS
TSX     .COPY,4             TSX     COPY,4
PZE     file1,,file2        PZE     file1,,file2
```

where:

file1 and file2
are the names of the files (the symbolic locations of the first words of their respective file control blocks).

The rules for information transfer with this routine are:

1. Both must be in the same buffer pool.
2. All words in the same buffer with and preceding the first word located by a nontransmitting command are included in the output.
3. All words in the same buffer with, and subsequent to, the last word located are not included in the output. Furthermore, they will behave as if skipped when *file1* is "read" again.
4. If more than two buffers are involved, an intermediate buffer is included in the output, provided at least one word in it has been located by a nontransmitting command.

PROGRAM EXAMPLE USING .COPY

The programs shown in Figures 10 and 11 illustrate the use of the .COPY(COPY) routine. The programs copy the physical records from an input file (IN1) onto an output file (OUT1). The block size for both files is the same and both files use the same buffer pool.

The program is illustrated first as it would be programmed by a user of Library IOCS and then as it would be programmed in Full IOCS.

Library IOCS: The first two steps in this program are FILE pseudo-operations that describe the files. (This pseudo-operation is discussed in the section entitled "Control Card Information for Library IOCS.")

The .READ calling sequence ends with a nontransmitting command (IORTN **, ,**) that secures the address of the first word in each input buffer and locates all words in that buffer. Each buffer is then copied onto the output file by the .COPY calling sequence. (Nontransmitting commands are described in the section entitled "Input/Output Control System (IOCS) Commands.")

When end of file is detected on IN1, the program transfers to location EOF where both files are closed. If an uncorrectable error occurs, the program transfers to ERR where a dump is taken.

Full IOCS: Note that space is reserved for file control blocks and the buffer pool. The buffer pool is defined and a list of two immediate files is attached to the pool. Processing is the same as in the program shown in Figure 10.

```
1       8       16
IN1     FILE    INPUTFILE,UT1,INPUT,BLK=14
OUT1    FILE    OUTPUTFILE,UT2,OUTPUT,BLK=14
SAM     TSX     .OPEN,4
        PZE     IN1                 (OPEN BOTH FILES
        TSX     .OPEN,4             WITH REWIND)
        PZE     OUT1
IN      TSX     .READ,4             (LOCATE ONE RECORD
        PZE     IN1,,EOB            FROM FILE IN1)
        PZE     EOF,,ERR
        IORTN   **,,**
        TSX     .COPY,4             (COPY RECORD FROM
        PZE     IN1,,OUT1           IN1 TO OUT1)
        TRA     IN                  (GO READ NEXT RECORD)
EOF     TSX     .CLOSE,4
        PZE     IN1                 (CLOSE WITH
        TSX     .CLOSE,4            REWIND AND UNLOAD
        PZE     OUT1                OF BOTH FILES)
        CALL    EXIT
EOB     PZE     **                  (THIS EXIT CANNOT BE TAKEN)
ERR     TRA     SYSDMP              (BAD INPUT DATA)
        END
```

Figure 10. Use of .COPY Routine

```
1       8       16
START   TSX     DEFINE,4        (DEFINE THE POOL TO HAVE
        PZE     BUF             8 BUFFERS OF 14 WORDS)
        PZE     8,,14
        TSX     ATTACH,4        (ATTACH TO THE POOL AND
        PZE     BUF             OPEN THE TWO FILES IN THE
        PZE     LIST,,2         LIST)
IN      TSX     READ,4          (LOCATE ONE RECORD
        PZE     IN1,,EOB        FROM FILE IN1)
        PZE     EOF,,ERR
        IORTN   **,,**
        TSX     COPY,4          (COPY RECORD FROM
        PZE     IN1,,OUT1       IN1 TO OUT1)
        TRA     IN              (GO READ NEXT RECORD)
EOF     TSX     CLOSE,4         (CLOSE BOTH FILES)
        PZE     LIST,,2         WITH REWIND AND UNLOAD)
        TRA     IOCS
EOB     PZE     **              (THIS EXIT CANNOT BE TAKEN)
ERR     TRA     SYSDMP          (BAD INPUT DATA)
IN1     BSS     12              (FILE CONTROL BLOCK FOR INPUT FILE
OUT1    BSS     12              (FILE CONTROL BLOCK FOR OUTPUT FIL
BUF     BSS     8*14+8*2+2      (SPACE FOR POOL)
LIST    PZE     IN1             (IMMEDIATE FILES)
        PZE     OUT1
        END     START
```

Figure 11. Use of COPY Routine

Stashing Data

The .STASH routine is used to transfer data between an internal file and some other file.

The form of the calling sequence is as follows:

LIBRARY IOCS		FULL IOCS	
TSX	.STASH,4	TSX	STASH,4
PZE	file1,,file2	PZE	file1,,file2
PZE	nts	PZE	nts

where:

file1 and file2
are the symbolic locations of the file control blocks for an internal file and some other file.

nts
is a location to which the program will transfer if IOCS determines there is "Nothing to Stash" or "No Place to Stash," under circumstances described below.

The rules for using this routine are:

1. Both files must be in the same buffer pool.

2. All words in the same buffer with and preceding the first word located are included in the output.

3. All words in the same buffer with and subsequent to the last word located are not included in the output. Furthermore, they will behave as if skipped when *file1* reading is resumed.

4. If more than two buffers are involved, an intermediate buffer is included in the output, provided at least one word within it has been located by a non-transmitting command.

Four cases are permitted in the use of the .STASH routine. In the following discussions of these cases, the word "external" is used to mean "not an internal file."

1. *file1* is an internal output; *file2* is an external output.

The initial buffer is deleted from the chain of *file1* buffers, and is appended to the chain of *file2* buffers. The *nts* return is taken if there is no buffer in the *file1* chain that has been completely processed. This usage allows the programmer to "write" into an external file from the buffers in an internal file. For example, if all the buffers associated with the internal file are filled, the programmer can use this technique to spill one buffer at a time into an output file.

2. *file1* is external input; *file2* is an internal output.

As in the case of the .COPY routine, the buffer(s) that contains information located by the last read sequence on *file1* is appended to the *file2* buffer chain. The *nts* return occurs if, during the reassignment, the Internal Group is about to overflow its full quota of buffers (as indicated by *bufct* in the Internal Group control word). This usage allows the programmer, for example, to set aside some data from *file1* for later processing, without the necessity of again processing the entire file.

3. *file1* is an internal input; *file2* is an external output.

For this usage, the .STASH routine behaves exactly like the .COPY routine. It should be noted that buffers will be removed from the *file1* chain and thereby are lost to *file1*, even if the "regenerative" read mode is in use. The *nts* return cannot occur.

4. *file1* is an internal input; *file2* is an internal output.

This functions exactly as in item 2. However, buffers will be removed from the *file1* chain even if the "regenerative" read mode is in use. The *nts* return occurs if, during the reassignment, the Internal Group to which *file2* belongs is about to overflow its full quota of buffers (as indicated by *bufct* in the Internal Group Control Word). Note that even though *file1* and *file2* must be attached to the same pool, they need not belong to the same Internal Group.

Closing Files

When activity on an immediate or reserve file is to be terminated, it must be closed by means of the CLOSE routine. Closing a file ensures that the buffer or buffers associated with it are not unnecessarily withheld from use by other files in the buffer pool during the remainder of the program. At closing, all input/output activity on a file ceases.

One form of the calling sequence is as follows:

LIBRARY IOCS		FULL IOCS	
TSX	.CLOSE,4	TSX	CLOSE,4
pfx	file	pfx	file

where:

file
is the name of the file (the symbolic location of the first word of the file control block).

pfx
controls closing action. The options are:

= PZE	Close with rewind and unload (note that a file that has been rewound and unloaded cannot be referred to again)
= PTW	Close with rewind
= MZE	Close with no rewind
= MON	Close with no rewind, and without writing a file mark or a trailer label
= PON	(Hypertape) Close with rewind, unload, and file protection
= PTH	(Hypertape) Close with rewind and file protection
= SIX	(Hypertape) Close with unload, but no rewind
= SVN	(Hypertape) Close with unload and file protection, but no rewind

Any output buffer currently in use is truncated, and the file is unbuffered and written out. All buffers in use by an input file are released to the pool. (If the file is an immediate file, the number of available buffers in the pool is increased. If the file is a reserve file, the count is not increased until the Reserve Group in which the file is included is closed.)

For an output file, a close with a prefix other than MON causes the following:

1. Labeled file – A file mark, a trailer label, and another end-of-file mark to be written
2. Unlabeled file – A file mark to be written

When a file is closed, whether at the end-of-file position or not, it is always unbuffered, so that its physical position corresponds to the last logical usage of the file. However, in the case of an input file, information in a partially processed buffer may be lost if the file is closed and then reopened, because the tape will be positioned beyond the block that occupied this buffer.

Care should be taken in specifying rewind for a System Input Unit, System Output Unit, or System Peripheral Punch Unit because of the possibility of later processing the wrong input file or destroying an output file.

Closing a List of Files

A list of files or part of a list can be closed in the same calling sequence as follows:

LIBRARY IOCS		FULL IOCS	
TSX	.CLOSE,4	TSX	CLOSE,4
pfx	filist,,numfil	pfx	filist,,numfil

where:

filist
is the symbolic location of a list of files. (This list may be a different list from any list used to attach the files, or the same list may be used.)

numfil
is the number of files to be closed.

pfx
specifies the same options as listed above. The same option is used for all files in the list being closed.

If a Reserve Group Control Word appears in the list, the entire group of files which it controls must also appear in the list and be closed in the same calling sequence. The number of files in the group is included in *numfil*. When a Reserve Group is closed, the number of available buffers in the pool is increased by the number of buffers that had been reserved for the group. (These files cannot be referred to again.)

Files of more than one pool may appear in the closing list. A file that is rewound and unloaded may never be referred to again.

Closing an Internal File

The following calling sequence is used to terminate the current processing of an internal file:

LIBRARY IOCS		FULL IOCS	
TSX	.CLOSE,4	TSX	CLOSE,4
pfx	file	pfx	file

where:

file
is the name of the internal file (the symbolic location of the first word of the file control block).

pfx
has the following meanings:

= PZE	All buffers in use by the file are released to the pool, and the file may not be referred to again. This is a "destructive" close, analogous to a rewind and unload for a regular file.
= MZE	Reorganizes the structure of the file, causing it to appear as if it had just been written. The file may be opened again (with any of the available options). This may be thought of as a "regenerative" close.

Closing a Checkpoint File

No close is necessary for a checkpoint file. However, if the programmer wants to close the file, all of the previously stated options for reserve files are available for checkpoint files.

Taking a Checkpoint

A checkpoint is a reference point which can be used later to restart a program from exactly that point. A checkpoint can be taken by using the following calling sequence:

LIBRARY IOCS		FULL IOCS	
TSX	.CKPT,4	TSX	CKPT,4

Checkpoints and restarts are discussed in the section "Checkpoints and Restarts."

Nondata Routines

The four routines described in the following text provide for backspacing records and files, rewinding, and writing an end of file. IOCS performs the routines in a manner that automatically positions the input/output unit so that the operation is performed on the designated physical record or file. The operations as they apply to IBM 729 Magnetic Tape Units are simulated

on IBM 1301 Disk Storage Units and IBM 7320 Drum Storage Units. Specific information on how they apply to the read backward mode on 7340 Hypertape Drives is included in the explanations.

The term *truncated* applies in some of the explanations. When an output buffer is truncated, no further words can be "written" into it.

Backspacing a Record

The .BSR routine can be used to backspace a physical record. The form of the calling sequence is:

LIBRAY IOCS		FULL IOCS	
TSX	.BSR,4	TSX	BSR,4
PZE	file,,bof	PZE	file,,bof

where:

file
is the name of the file (the symbolic location of the first word of the file control block).

bof
is the location to which transfer will be made if beginning of file is encountered.

For an output file, the current buffer is truncated, and the file is unbuffered and written out. The unit is then backspaced over the last record "written." The block count is reduced by 1.

For an input file, the file is first unbuffered and the unit assigned to the file is backspaced over the last record "read." The block count is reduced by 1. (In read backward mode, the unit is physically forward-spaced one record.)

If the backspace passes over a file mark, the beginning-of-file exit (*bof*) is taken.

Backspacing a File

The .BSF routine can be used to backspace a specified number of physical files. The form of the calling sequence is as follows:

LIBRARY IOCS		FULL IOCS	
TSX	.BSF,4	TSX	BSF,4
PZE	file,,count	PZE	file,,count

where:

file
is the name of the file (the symbolic location of the first word of the file control block for this file).

count
is the number of files over which the programmer intends to backspace.

The file is first unbuffered, and *count* machine backspace file instructions are executed, followed by a forward space over the last file mark encountered. However, the forward space is suppressed if the tape reaches the beginning-of-tape position. A *count* of zero can be used to unbuffer the file.

In the read backward mode, the file is unbuffered, and *count* machine forward-space file orders are executed, followed by a backward space over the last file mark encountered. The backward space is suppressed if the Hypertape reaches the EWA position.

When the backspace file routine is used to backspace over one or more labeled files, the structure of such files must be kept clearly in mind. For example, after an end-of-file exit has been taken for a labeled input file, use of the backspace file routine will produce the results indicated in the following chart, depending on the count specified.

COUNT		TAPE POSITION ON EXIT
1		Beyond file mark following the trailer label.
2		In front of the trailer label.
3		In front of the first data block of the file.
4	a.	In front of the first header label for the file, if it is a single-reel file.
	b.	In front of the label of the current reel of a multi-reel file without checkpoints.
	c.	In front of the checkpoint record of a multi-reel file with checkpoints.
5		In front of the header label for the file, if a checkpoint was written on the file.

Writing a File Mark

The .WEF routine can be used to write a file mark on an unlabeled output file. The form of the calling sequence is as follows:

LIBRARY IOCS		FULL IOCS	
TSX	.WEF,4	TSX	WEF,4
PZE	file,,eot	PZE	file,,eot

where:

file
is the name of the file (the symbolic location of the first word of the file control block).

eot
is the location to which transfer will be made if end of tape is encountered.

For an output file, any buffer connected is truncated, the file is unbuffered and written out, and a file mark is written following it. If end of tape is encountered while writing the tape mark, the end-of-tape exit (*eot*) is taken.

The .WEF routine is ignored for an input file. Furthermore, the .WEF routine will not be executed on a labeled file, because it would result in an incorrect end-of-file condition for the file. For a labeled file, the .CLOSE routine should be used, because this routine will automatically ensure that the file is labeled properly.

Rewinding a File

The .REW routine can be used to rewind the current unit in use by an unlabeled file. The calling sequence is as follows:

LIBRARY IOCS		FULL IOCS	
TSX	.REW,4	TSX	REW,4
PZE	file,,x	PZE	file,,x

where:

file
is the name of the file (the symbolic location of the file control block for the file).

x = 0 for 729 magnetic tape, 1301 disk storage, and 7320 drum storage files.

= 1 (Hypertape only) If the 1 is present, the unit will be rewound and file-protected. If the 1 is not present, the unit will be rewound only.

Any output buffer currently in use is truncated, the file is unbuffered and written out, and a rewind occurs. If reel switching has occurred, only the current reel is rewound. Any attempt to rewind a labeled file will be ignored.

For an input file, an end of file that has been detected by IOCS, but not yet logically reached in processing, is canceled before the rewind is issued.

Printing Messages On-Line and Off-Line

The Input/Output Executor (IOEX) contains a system routine for printing messages on the channel A printer or on the System Output Unit. The name of this IOEX routine is (PROUT. Off-line messages are actually printed by a separate routine, SPOUT, which is called by (PROUT when an off-line message is specified.

SPOUT is stored in core storage locations SYSEND-199 through SYSEND. Therefore, if off-line messages are specified, SYSEND-200 is the end of usable core storage.

The programmer can reach the (PROUT routine by using the .MWR routine in IOCS. The calling sequence is as follows:

```
LIBRARY IOCS                    FULL IOCS
TSX     .MWR,4                  TSX     MWR,4
PZE     n                       pfx     n
pre     loc1,t,m+512*spr1       pre     loc1,t,m+512*spr1
pre     loc2,t,m+512*spr2       pre     loc2,t,m+512*spr2
...     ... ,,,. ....           ...     ... ,,,. ... ....
pre     locx,t,m+512*sprx       pre     locx,t,m+512*sprx
```

where:

pfx
indicates where the message is to be printed.

= PZE The message is to be printed on-line only. This is the only option that can be used with Library IOCS.

= MZE (In Full IOCS only) The message is to be printed on-line and recorded off-line.

= MON (In Full IOCS only) The message is to be recorded off-line only.

n
is the number of subsequent parameters in the calling sequence.

loc
is the symbolic location of the first word of the block of data to be printed.

t
is the index register tag, if any.

m
is the number of words to be printed starting at *loc,t.*

512*spr
controls page ejection and spacing between print lines.

The *m* consecutive words (of six BCD characters each) starting at location *loc,t* are placed in the line image for printing. Note that the maximum number of characters per line is 72 (m=12).

The *pre* is a prefix controlling the formation of the print lines, as follows:

pre

= PZE The image is taken to be complete and the line is printed.

= MZE This line is considered incomplete, and the *loc,t,m* of the next entry in the calling sequence is used to continue building the image, beginning with the next print position to the right.

If *pre*=PZE, the sense exit *spr* is activated after the line is printed. (*spr* appearing in a word with *pre* =MZE is ignored).

To activate an exit hub before printing the first line, an entry should be made in the calling sequence, in the following form:

```
PZE     **,,512*spr
```

This will print a blank line, followed by activation of the hub *spr.*

For off-line printing, *spr* is interpreted as follows:

=0 Normal (single) space.

=1 Eject to a new page.

=4 Double space.

These interpretations also apply for on-line printing if the SHARE standard sense exit hubs are used.

Example: Suppose two consecutive words at LOC1 contain the BCD text:

```
THIS IS ONE
```

and in the word at location LOC2 is the BCD text:

```
SAMPLE
```

The following calling sequence:

```
LIBRARY LOCS                    FULL IOCS
TSX     .MWR,4                  TSX     MWR,4
PZE     2                       PZE     2
PZE     LOC1,,2                 PZE     LOC1,,2
PZE     LOC2,,1+512*1           PZE     LOC2,,1+512*1
```

would result in printing of the two lines:

```
THIS IS ONE
SAMPLE
```

followed by a page eject.

Further information on the (PROUT routine is contained in the publication *IBM 7090/7094 IBSYS Operating System, System Monitor (IBSYS)*, Form C28-6248.

All text references to IOCS routines in this section are made to the name of the routine in Library IOCS. The text applies equally to the same routines in Full IOCS.

File Processing

IOCS commands are an integral part of the .READ and .WRITE calling sequences. The format of IOCS commands is identical to the format of commands used with the IBM 7607 Data Channel, and their interpretation is quite similar.

IOCS provides two methods of processing data within the computer. One method is actually to transmit the data between buffers and working storage by using transmitting commands. The other method is to use nontransmitting commands to determine the location of desired data in an input buffer, locate space in an output buffer, or skip over data words in a buffer.

The programmer may find it easier logically to use transmitting commands, but these commands require computer time to transmit data. Nontransmitting commands offer the speed advantage of processing data directly within the buffer.

Input Blocks and Buffers

It should be noted that the size of the buffers in a pool determines the size of blocks to be processed. If the size of a physical input record on a recording medium is larger than the buffers provided for that file, the physical record is truncated to buffer size. When a record is read from the input device into a buffer, data is read until the buffer is filled and the rest of the record is lost.

Output Blocks and Buffers

Each physical record recorded on an output unit consists only of the number of words "written" into a buffer before it is released for external transmission.

Buffer Truncating

In the following discussions, the term *truncate* is used. Certain IOCS commands will cause a buffer to be truncated, meaning that the remaining words in the buffer are disregarded.

When an input buffer is truncated, the buffer is disconnected from use and returns to the chain of available buffers in the buffer pool. It remains as a usable buffer in the pool, but its immediate contents are no longer accessible to the programmer. Any unprocessed words that remain in the buffer are discarded.

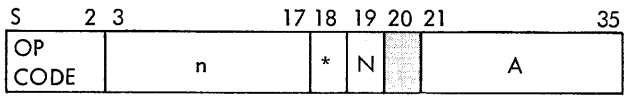
When an output buffer is truncated, it is disengaged and joins a chain of output buffers waiting to have their contents written on an output unit. Only the number of words inserted into the buffer before it was truncated will be written as an output record.

Thus, in the following explanations, disconnecting commands (D) indicate to IOCS that the programmer no longer wants the last buffer in use, and that the buffer should be truncated.

Appendix C contains a detailed description of the execution of each IOCS "read" and "write" command.

Format of a Command

The format of an IOCS command is as follows:



This is represented symbolically as follows:

```
1          8        16
           IOxy(N)(*) A,,n
```

where:

x
designates the type of control. It may be one of the following:
- C – Count control. The count supplied by the programmer in these commands specifies the number of words to be processed.
- R – Buffer (record) control. These commands process all (or the remainder) of a buffer.
- S – Special count control. These commands process under count control, except that special action is taken at the end of a buffer.

y
indicates the function to be performed at the completion of the command. It may be one of the following:
- P – Proceed to the next command in the command list.
- T – Terminate the command list.
- D – Terminate the command list and truncate the attached buffer.

N
if used, specifies that the command is nontransmitting. A nontransmitting command causes IOCS to provide an address within a buffer.
NOTE: N is assembled as index register 2.

*
is used to specify that the address, A, is indirect.
NOTE: The effective address is determined as though all the index registers contained zero.

A

is the address of the first word processed. In a transmitting command (not an IOxyN type), the address A is supplied by the programmer. It indicates the location to which input data should be transmitted, or the location of output data to be transmitted to one or more buffers.

If the command is a nontransmitting command (N is used), IOCS supplies the location of the next available word in the buffer. Normally, IOCS inserts the location of that word into the address portion of the command. However, if the nontransmitting command is indirectly addressed, the location of the next available word is inserted into the address of the word specified in the address of the IOCS command.

n

is the number of words to be processed. This is normally supplied by the programmer. However, if a buffer control command (R) is used for "reading" input data, the count of the number of words "read" by the command is inserted into the decrement of the command.

NOTE: In the discussions that follow, ** is used to indicate that the address or count in a command is inserted by IOCS during processing, rather than being provided by the programmer.

Transmitting Commands

The transmitting commands move data words either (1) from an input buffer (or buffers) to working storage, or (2) from working storage to an output buffer (or buffers). During processing, buffers associated with one file are chained together and presented in sequence for processing. This permits the programmer to treat each file, in general, as a continuous string of words, with the string extending from one buffer to the next.

Thus, by using count control commands (except IOCD), a programmer can process across buffers as if there were no division between them. If 15-word buffers were in use by one file, a series of count control commands could process 10 words each. The second command would start with the eleventh word in the first buffer and process through the fifth word in the next buffer. IOCS would connect and disconnect the buffers automatically.

A command list in a .READ or .WRITE calling sequence can consist of one or more commands. The effect of each IOCS command depends in part on the number of words "read" or "written" by preceding commands in the command list.

Count Control Commands

The count control transmitting commands are as follows:

IOCP
IOCT
IOCD

FORMAT AND EFFECT

The format of a count control transmitting command is as follows:

```
1               8               16
                IOCy            A,,n
```

"Reading": *n* words are transmitted from one or more buffers connected to the file into consecutive locations starting with *A*. Transition from buffer to buffer is automatic. IOCD, in addition, truncates the buffer in use when the *n* count is fulfilled.

"Writing": *n* words are transmitted from consecutive locations beginning with *A* into one or more buffers connected to the file. Transition from buffer to buffer is automatic. IOCD, in addition, truncates the buffer in use when the *n* count is fulfilled.

EXAMPLES

1. Using the command:

```
IOCP        AREA,,10
```

"Reading": Ten words are transmitted from an input buffer (or buffers) into AREA through AREA+9, and IOCS proceeds to the next command.

"Writing": Ten words are transmitted to an output buffer (or buffers) from locations AREA through AREA+9.

2. Using the command:

```
IOCD        AREA,,20
```

"Reading": Suppose each input buffer contained 30 words. The first 20 words would be transmitted into AREA through AREA+19. The buffer would be truncated, and any unprocessed words in it would be lost.

If each buffer contained six words, the contents of three buffers and two words from the fourth buffer would be transmitted. The fourth buffer would be truncated.

Buffer Control Commands

The buffer control transmitting commands are as follows:

IORP
IORT

FORMAT AND EFFECT

"Reading": When "reading," the format of a buffer control command is as follows:

```
1               8               16
                IORy            A,,**
```

Words are transmitted from the input buffer to consecutive locations starting with *A*, until either the end of the buffer or the end of the physical record in the buffer is reached (whichever occurs first). IOCS inserts the count of the number of words transmitted into the decrement portion of the IORY command.

"Writing": The format when "writing" is as follows:

```
1               8               16
                IORy            A,,n
```

Words are transmitted from consecutive locations starting with *A* until the count *n* is reduced to zero or

the end of a buffer is reached (whichever occurs first). If the count is satisfied before reaching end of buffer, only a partial buffer will be written.

EXAMPLES

1. Using the command:

```
IORT    AREA,,**
```

"Reading": If 10 words had previously been "read" from a 20-word input buffer, the remaining 10 words would be "read" and a count of 10 would replace the **. If no words had been read from the 20-word buffer, all 20 words would be "read" and a count of 20 would replace the **.

In both cases, the buffer would be released.

2. Using the command:

```
IORT    AREA,,15
```

"Writing": If 5 words had been placed in a 20-word buffer by a previous command, 15 words would be "written" into the buffer. If 10 words had been placed in the buffer by a previous command, only 10 words would be transmitted, thus filling the buffer, and the buffer would be released for transmission to an output unit.

In both cases, the command list would be terminated.

Special Count Control Commands

The special count control commands are as follows:

```
IOSP
IOST
```

FORMAT AND EFFECT

The form of a special count control transmitting command is as follows:

```
1       8       16
        IOSy    A,,n
```

"Reading": *n* words are transmitted from the input buffer to consecutive locations starting at *A*, until the end of buffer is reached or the count *n* is reduced to zero (whichever occurs first). A buffer is never released by an IOSY command.

"Writing": If the buffer being used is capable of holding *n* additional words, the *n* words are transmitted to this buffer. If the buffer cannot hold *n* additional words, the buffer is truncated and the *n* words are placed in the next buffer.

EXAMPLES

Using the command:

```
IOSP    AREA,,10
```

"Reading": If there are 10 words remaining in the buffer, the 10 words are transmitted into AREA through AREA+9.

If only 5 words are left in the buffer, only 5 words are transmitted because end of buffer is reached. The fact that only 5 words were transmitted could be determined by examining the "history record."

"Writing": Assume the file is using 15-word buffers, and no words have been transmitted to the current buffer by a previous command. In this case, the 10 words are transmitted into the current buffer.

Assume 15-word buffers and 8 words have already been transmitted to the current buffer. In this case, the current buffer is truncated, and the 10 words are transmitted to the next buffer. The remaining 5 words could be filled by a later IOCS command.

Transfer and Continue Command List

The form of this command is:

```
1       8       16
        TCH     A
```

The command list continues at A. The normal exit from an IOCS routine is the location after the first TCH command in the list.

EXAMPLE

Assume the following command list had been used to "read" 40 words from a buffer into four areas in working storage:

```
IOLIST  IOCP    AREA1,,10
        IOCP    AREA2,,10
        IOCP    AREA3,,10
        IOCT    AREA4,,10
```

Later in the program, the programmer could "write" the 40 words from the same areas by using the sequence:

```
        TSX     .WRITE,4
        PZE     file,,eob
        TCH     IOLIST
```

Ten words would be "written" from each area into a buffer. The exit from the command list would be at the instruction following the TCH command.

Examples of Other Usage

1. An input file is attached to a buffer pool containing buffers of 20 words and the following command is used:

```
IOCD    WKAREA,,24
```

Twenty-four words are 'read" into WKAREA through WKAREA+23, and both buffers are released. The history record in the AC would be PZE WKAREA+24,,16.

Suppose the command were as follows:

```
IOCT    WKAREA,,24
```

In this case, 24 words would be 'read" into the same locations, but only the first buffer would be released. The history record in the AC would be PZE WKAREA+24,,16. The next reference to the file in a .READ

calling sequence would begin operation on the fifth word in the second buffer.

2. The following command could be used to "read" variable-length records from an input file:

```
IORT      RECORD,,**
```

This command would transmit the next record from an input buffer into locations starting at RECORD. The actual length of the record that was transmitted would be inserted into the decrement portion of the command.

The area in storage beginning with RECORD would have to be as large as the input buffer. Otherwise, storage locations beyond that area might be destroyed when the record was transmitted.

3. Some records are written as "self-loading" variable-length records. The first word of each self-loading record is itself an IOCS command that can be used to read properly the record as input.

The following sequence could be used to process an input file containing self-loading records:

```
IOCP      *+1,,1
PZE       **,,**
```

The first entry in the sequence tells IOCS to pick up the first word in the record and to place it in the location following the IOCP command. By the time IOCS reaches the next location, the command from the record has been inserted there and IOCS proceeds to execute it.

Nontransmitting Commands

Nontransmitting commands provide the IOCS programmer with a means of finding desired locations within input or output buffers, or a means of skipping over words in a buffer. Use of these commands make it possible to process input data directly within a buffer or to form an output record within a buffer.

The nontransmitting commands are distinguished by the presence of the N following the usual form of a transmitting command.

```
IOCP      A,,10 is a transmitting command
IOCPN     **,,10 is a nontransmitting command
```

The nontransmitting commands can be used to perform two functions: locating and skipping.

Locating: This is the process of using nontransmitting commands to find an address in an input or output buffer. IOCS inserts that address into the address portion of the command itself. Or, if indirect addressing is used, the located address is inserted into the effective address. Locating can be used to do the following:

1. Determine the location of the first word or the next unprocessed word in an input buffer.

2. Determine the location of the next available word in an output buffer or find space for a specified number of words in an output buffer.

Skipping: This is the process of using nontransmitting commands to do the following:

1. Skip over any number of words in an input buffer

2. Inform IOCS of the actual number of words already placed in an output buffer by using an IOSYN type command.

End-of-Buffer Switch

The *eob* entry in each .READ and .WRITE calling sequence is called the end-of-buffer switch. This switch is interrogated by IOCS each time the end of buffer is reached, regardless of whether a transmitting or nontransmitting command is being executed. The *eob* switch indicates to IOCS whether the buffer should be retained until the next reference to the file is made in an IOCS routine, or whether it should be immediately truncated and released for input/output activity.

If *eob* = 0, the buffer is truncated and IOCS automatically positions another buffer for use. Command execution continues without interruption.

If *eob*≠0, all information located by the nontransmitting command will be retained until the next reference to the file is made in any IOCS routine.

The *eob* switch also indicates to IOCS whether a nontransmitting command is intended to locate or skip information. The programmer uses the *eob* switch to specify either locating or skipping, as follows:

Skipping: If *eob* = 0, the nontransmitting command is executed as a skip.

Locating: If *eob*≠0, the command is interpreted as an attempt to locate information.

Rules for Using Nontransmitting Commands

The rules governing the use of nontransmitting commands are as follows:

1. No single nontransmitting command can locate words in more than one buffer.

2. Input data or output buffer space located by a single IOCS command must be in sequential words. Therefore, when locating with a count control nontransmitting command, the execution of the command is discontinued when end of buffer is encountered. Transfer is made to the location specified in *eob* and no further commands in the command list are executed. An end-of-buffer condition occurs when all words in a buffer have been used and the command word count has not been satisfied.

3. If "reading" and *eob*≠0, words located are available for processing until the file is next referenced by an IOCS routine. Hence, the buffer in which the words were located is retained until then.

4. If "writing" and *eob*≠0, the space located for output words is considered filled when the file is next

referred to by any IOCS routine, Hence, it is not written until then.

5. A sequence of commands can be used to locate words in more than one buffer if sufficient buffers have been reserved for the file by defining a Reserve Group which consists of that file only.

6. Nontransmitting commands may be freely intermixed with transmitting commands in any command sequence. However, since the *eob* switch is set on or off by each entry into the .READ or .WRITE routines, skipping and locating cannot be done by the same sequence.

Count Control Nontransmitting Commands

The count control nontransmitting commands are as follows:

IOCPN
IOCTN
IOCDN

FORMAT AND EFFECT

The form of a count control nontransmitting command is as follows:

```
1           8           16
            IOCyN       **,,n
```

"Reading": The next *n* words of the file are located or skipped, and the location of the first of these is inserted into the address portion of the command (replaces the **). In addition, and IOCDN command will cause the buffer to be truncated.

"Writing": The ** is replaced by the location of the next available word in an output buffer in use by the file. IOCS will assume that *n* words have been placed into the *n* locations before the next reference to that file. In addition, IOCDN will cause the buffer to be written upon the next reference to that file.

EXAMPLES

1. Suppose the buffers for an input file contained 20 words, and no words had been located or skipped by a previous command.

In this example, the following command list is used:

```
IOCPN     **,,10      (eob ≠ 0)
IOCTN     **,,5
```

The location of the first word in the buffer replaces the ** in the first command, and the location of the eleventh word is inserted in the second command. The buffer is retained so that the remaining five words, which have not been located, could be used by another command.

2. Suppose the buffers in use by an output file are defined as having only 10 words each, and the following command list is used in a .WRITE calling sequence:

```
IOCPN     **,,5       (eob ≠ 0)
IOCTN     **,,10
```

Five words are located by the first command and five words are located by the second command. Execution of the second command is interrupted by end of buffer and exit would be made to the address indicated by the *eob* entry. (One command cannot locate words in more than one buffer.) The buffer is not released until the next reference is made to the file. Since the *eob* exit was taken, the history record in the AC would contain the following:

$$\text{PZE} \quad \begin{pmatrix}\text{buffer}\\ \text{address}\end{pmatrix} + 10,, - \begin{pmatrix}\text{the location following}\\ \text{the IOCTN command}\end{pmatrix}$$

Buffer Control Nontransmitting Commands

The buffer control nontransmitting commands are as follows:

IORPN
IORTN

FORMAT AND EFFECT

"Reading": When "reading," the format is as follows:

```
IORyN     **,,**
```

The location of the next unprocessed data word in an input buffer replaces the ** in the command, and the count (*n*) of the remaining words in that buffer is inserted into the decrement portion of the command. The next reference to the command will cause the *n* located words to be bypassed.

"Writing": When "writing," the format is as follows:

```
IORyN     **,,n
```

The location of the next available word in an output buffer replaces the ** in the address of the command. IOCS reduces the number of available words in the buffer by *n* words, unless the end of buffer is reached first. The next reference to the file causes the contents of the buffer to be written as an output record. The programmer can examine the history record to determine if less than *n* words were located or skipped.

EXAMPLES

1. Suppose an entire record from a given input file is to be located (that is, a programmer wants to obtain the address of the first word of that record). The following command could be used to find that address and to determine the number of words in the record.

```
IORTN     **,,**
```

Assume that the first word in the record is at an address that can be represented by the word RECORD, and that the record contains 60 words. When an exit is taken from the read routine, the preceding command is changed to the following:

```
IORTN     RECORD,,60
```

2. The buffer pool to which a file is attached contains 14-word buffers. This example uses the following command:

```
IORTN     **,,20      (eob = 0)
```

Fourteen words are skipped and the buffer is released. The history record in the AC is as follows:

```
PZE  (buffer address)  +14,,0
```

3. Consider Example 2 when each of the following commands is used:

```
IOCTN     **,,20      (eob = 0)
```

Twenty words are skipped and the first buffer is released.

```
IOCDN     **,,20      (eob = 0)
```

Twenty words are skipped and both buffers are released.

Special Count Control Nontransmitting Commands

The special count control nontransmitting commands are as follows:

IOSPN
IOSTN

FORMAT AND EFFECT

The format of a special count control nontransmitting command is as follows:

```
IOSyN     **,,n
```

"Reading": Unprocessed words in the input buffer are bypassed until either *n* words have been skipped or located or the end of the buffer is reached (whichever occurs first). The location of the first word located is inserted into the address portion of the command. The actual number of words skipped or located is reflected in the history record. The IOCS record of the number of available words in the buffer (the available word locator) is reduced by the number of words skipped or located.

If end of buffer is encountered first, the end-of-buffer exit is not taken. In addition, an IOSYN command will never release a buffer.

"Writing": If the output buffer is capable of holding *n* additional words, the location of the first available word replaces the ** in the address of the command. Otherwise, that buffer is truncated and the location of the first available word in the next buffer replaces the **.

NOTE: When used for "writing," an IOSYN command never adjusts the available word locator (a portion of buffer control word 1 which reflects the number of available words in the buffer while the buffer is in use). Hence, when an IOSYN command is used for "writing," an IOCYN or IORYN command must be executed afterward to adjust the available word locator by the actual number of words placed in the buffer.

EXAMPLES

1. If the input buffers for a file contain 15 words and the following sequence is used:

```
IOSTN     **,,20      (eob = 0)
```

Fifteen words are skipped and the buffers are not released.

2. Suppose a programmer desires to locate space in an output buffer in which to place *n* words. The programmer can use the following command:

```
IOSTN     **,,n       (eob ≠ 0)
```

IOCS checks the buffer currently in use to determine whether enough space remains in the buffer to accept *n* words. If not, IOCS truncates that buffer and proceeds to the next buffer.

The address of the first word of the next available buffer is inserted into the IOCS command. The programmer could then use this address to transmit the *n* number of words into the located space. Because the command was an IOSYN (the only command type which does not advance the available word locator), the programmer must subsequently inform IOCS of the actual number of words placed in the located area by means of a skipping command such as:

```
IOCTN     **,,n       (eob = 0)
```

where *n* is the actual number of words placed in the buffer. The above technique can be used to create a variable length output record.

Examples of Other Usage

1. Suppose the next word of a file is to be placed into the accumulator. Then, the following command could be given:

```
IOCTN*    CLAI,,1
```

where CLAI is the symbolic location of a particular CLA instruction.

IOCS locates the address of the next word. Then, under control of the indirect addressing specified in the IOCS command, it places that address in the address portion of the CLA instruction.

The CLA instruction is modified as follows:

Before the read sequence, it would be:

```
CLAI      CLA       **
```

After the read sequence, it would be:

```
CLAI      CLA       ADDR
```

where ADDR is the location of the desired word. Note that the effective address computation is performed as though all of the index registers contain zero.

2. If a file is being used in such a way as to withhold more than one buffer at a time from a pool, it must be treated as a Reserve Group (by itself) and the buffer count (BUFCT) must be equal to at least the number of buffers that will be held in use at one time.

For example, suppose that a file whose file control block is located at FILE has been produced by a card-to-tape operation. Suppose, further, that five physical records (five card images on tape) are to be processed at one time, and that five input buffers containing this information must be retained during the processing. The location of the first word in each buffer is to be transmitted indirectly by using the following .READ calling sequence and command list:

```
TSX     .READ,4
PZE     FILE,,EOB
PZE     EOF,,ERR
IORPN*  RCD1,,**
IORPN*  RCD2,,**
IORPN*  RCD3,,**
IORPN*  RCD4,,**
IORTN*  RCD5,,**
```

Since eob≠0, the nontransmitting commands locate the initial address of each card buffer. Since indirect addressing is used, the address of the first word in each buffer is inserted into the address portions of locations RCD1, RCD2, RCD3, RCD4, and RCD5, respectively. If the instructions at each of these effective addresses were as follows:

```
RCDx    AXC     **,4
```

then, the nth word in any of the buffers could be referenced by first executing the instruction at location RCDx and then executing an instruction such as:

```
        CLA     n,4
```

This procedure will require the use of no fewer than five buffers by the file at all times. Actually, IOCS would try to use at least ten buffers. At each entry to the .READ routine, when the five buffers just processed are released to the pool, maximum overlap occurs if the next five buffers have already been filled and are waiting to be located. If, however, ten buffers are not available (either because there are not ten in the pool, or because other files are using them), the system can still operate properly as long as there are at least five buffers available to the file. This condition can be guaranteed only by the following methods:

In Library IOCS: By making this file the only file named in a $GROUP card and by specifying that BUFCT is five, the programmer can assure that at least five buffers are always available for this file. BUFCT should be ten or more to achieve overlap.

In Full IOCS: The file can be established as a Reserve Group of one file by making the following entry in the file list:

```
SVN     1,,m
PZE     FILE
```

with *m* (the buffer count) not less than five.

It should be noted that if the programmer has erred and specified only three buffers, as follows:

```
SVN     1,,3
PZE     FILE
```

then the end-of-buffer exit would have been taken after execution of the third IORPN*. This would occur even if, at that particular time, a buffer were available in the pool for use by another file. IOCS control forces this action, since otherwise some file might later be unable to obtain a buffer. In this usage, *eob* is to be interpreted as "end of available buffers."

History Records

When the last IOCS command in a .READ or .WRITE calling sequence is completed, or when an error condition interrupts the execution of a command sequence, a record of action is produced in the AC and MQ.

This record of action is called the history record. It provides the programmer with information about the last IOCS command that was executed.

The count of the remaining words in the buffer can be used to create future IOCS commands or to make other logical decisions. The investigation of the last word "read" or "written" may be necessary to determine what the command actually did, as in the case of an IOST command when "reading," where the count specified may not have been satisfied because the end of buffer was encountered. Similarly, an IORY command, when used for "writing," may not "write" the specified number of words if it encounters an end-of-buffer condition. The command list can also be interrupted by encountering an end of file on an input file.

The history records are as follows:

Normal Exit

At each normal exit from a .READ OR .WRITE routine, the record is in the AC.

AC

S	2 3 — 17		21 — 35
	Number of usable words remaining in the buffer that contained the last word "read", or the number of unused words in the buffer containing the last word "written"		1 plus the location of the last word "read" or "written", if in the transmitting mode

Error Exit

An unexpected exit from the routine will be taken if any of the following conditions are encountered:

1. "Reading" – End of buffer, end of file, block sequence error, check sum error, or a parity error which cannot be corrected
2. "Writing" – End of buffer

The history record provides the following information in the AC and MQ:

AC

S	2 3 – 17	18 20	21 – 35
	The 2s complement of the following quantity: 1 plus the location of the command being executed when the condition was encountered		1 plus the location of the last word "read" or "written", if in the transmitting mode

MQ

S	1	2	3 – 17	21 – 35
Parity error or end of available buffers	Check sum error or end-of-buffer error	Block sequence error	The 2s complement of the location of the TSX to the IOCS routine	The 2s complement of the location of the normal return from the IOCS routine

When a parity, check sum, or block sequence error occurs during "reading," the prefix of the MQ will contain the following:

BITS	MEANINGS
110	check sum and parity errors have occurred
101	sequence and parity errors have occurred
100	a parity error occurred which could not be corrected
010	a check sum error has occurred
001	a sequence error has occurred

The order in which these errors is detected is: check sum, block sequence, and parity. If all three checks are being made on a file, the occurrence of only a parity error is probably a false parity error, because the check sum is correct. However, the user must realize that check sums are not a foolproof check.

When an end-of-buffer exit occurs during either "reading" or "writing," the prefix of the MQ is one of the following:

BITS	MEANINGS
100	all available buffers are in use
010	the end-of-buffer condition was encountered during execution of an IOCYN command

Each of the two sample programs in this section is illustrated first as it would be programmed in Library IOCS and then as it would be programmed in Full IOCS. The comments fields explain the effects of the instructions.

Example 1

In this example, an output file (OUT1) is generated by writing 100 records from the same storage area (RECORD). The first word of each record is a record number that is increased by 1 before the next record is written. OUT1 is then read back as IN1 and duplicated to another output file (OUT3).

An unusual deblocking and blocking technique is employed in the command sequences to illustrate the effects of IOCS commands and indirect addressing.

On "reading" from IN1, 10-word segments are alternately transmitted or located. On "writing" to OUT3, indirect addressing is used to "write" the first 40 words of each record and the last 10 words are transmitted directly from ZOLEC+20.

In the example, encountering end of buffer or an uncorrectible error would cause a transfer to a system dump. The programs are shown in Figure 12 and 13.

```
1       8       16
$EXECUTE        IBJOB
$IBJOB          GO,MINIMUM
$IBMAP  DKNAM   100
IN1     FILE    GENERATEDFILE,UT1,INPUT,BLK=50
OUT1    FILE    GENERATEFILE,UT1,OUTPUT,BLK=50
OUT3    FILE    DUPLICATEFILE,UT2,OUTPUT,BLK=50
SAM     TSX     .OPEN,4             (OPEN OUT1)
        PZE     OUT1
        AXT     100,1               (SET INDEX REGISTER FOR LOOP)
        TSX     .WRITE,4            (WRITE A RECORD OF 50 WORDS
        PZE     OUT1                TO OUT1)
        IORT    RECORD,,50
10A     CLA     =1                  (CHANGE VALUE OF FIRST WORD
        ADD     RECORD              OF RECORD BY ONE)
        STO     RECORD
        TIX     *-6,1,1
        TSX     .CLOSE,4            (CLOSE OUT1 WITH REWIND
        PTW     OUT1                AND WEF)
        TSX     .OPEN,4
        PZE     IN1
        TSX     .OPEN,4             (OPEN IN1 AND OUT3)
        PZE     OUT3
LEC     TSX     .READ,4             (READ FROM IN1)
        PZE     IN1,,EOB            (IF EOB, GO TO EOB)
        PZE     EOF,,ERR
IO      IOCP    ZOLEC,,10           (TRANSMIT FIRST 10 WORDS TO ZOLEC)
        IOCPN   **,,10              (LOCATE NEXT 10 WORDS)
        IOCP    ZOLEC+10,,10        (TRANSMIT 10 MORE WORDS)
        IOCPN   **,,10              (LOCATE NEXT 10 WORDS)
        IORT    ZOLEC+20,,10        (TRANSMIT LAST 10 WORDS)
        TSX     .WRITE,4
        PZE     OUT3
        IOCP*   IO,,10              (WRITE 10 WORDS AT A TIME
        IOCP*   IO+1,,10            INDIRECTLY)
        IOCP*   IO+2,,10
        IOCP*   IO+3,,10
        IOCT    ZOLEC+20,,10        (LAST 10 WORDS DIRECTLY)
        TRA     LEC                 (GO TO LEC)
EOF     TSX     .CLOSE,4            (CLOSE IN1 AND OUT3)
        PTW     IN1
        TSX     .CLOSE,4
        PTW     OUT3
        CALL    EXIT
EOB     TRA     SYSDMP              (CALL DUMP)
ERR     TRA     SYSDMP
RECORD  PZE     0                   (RECORD=50 WORDS)
        DUP     1,49
        BCI     1,
ZOLEC   BSS     30
        END
  (END-OF-FILE CARD)
$IBSYS
```

Figure 12. Example 1 Using Library IOCS

```
1       8       16
$EXECUTE        IBSFAP
        *FAP
        COUNT   100
        ABS
        SST                         (SYMBOL TABLE)
DEBUT   BOOL    13000
        ORG     DEBUT
IOCS    EQU     SYSORG
DEFINE  EQU     IOCS+4              (ESTABLISH RELATIVE LOCATIONS
ATTACH  EQU     IOCS+8              OF IOCS TRANSFER VECTORS)
OPEN    EQU     IOCS+12
READ    EQU     IOCS+14
WRITE   EQU     IOCS+16
CLOSE   EQU     IOCS+10
IN1     BSS     12                  (FILE CONTROL BLOCKS)
OUT1    BSS     12
OUT3    BSS     12
BEGIN   TSX     DEFINE,4            (DEFINE POOL AS A BUFFER
        PZE     POOL                POOL CONTAINING 10 BUFFERS,
        PZE     10,,50              EACH BUFFER=50 WORDS)
        TSX     ATTACH,4            (ATTACH 3 FILES
        PZE     POOL                IN LOCATION LIST
        PZE     LIST,,3             TO POOL)
        TSX     OPEN,4              (OPEN OUT1)
        PZE     OUT1
        AXT     100,1               (SET INDEX REGISTER FOR LOOP)
        TSX     WRITE,4             (WRITE A RECORD
        PZE     OUT1                OF 50 WORDS FROM
        IORT    RECORD,,50          AREA RECORD)
10A     CLA     =1
        ADD     RECORD
        STO     RECORD              (RECORD=RECORD+1)
        TIX     *-6,1,1
        TSX     CLOSE,4
        PTW     OUT1                (EOF AND REWIND ON OUT1)
        TSX     OPEN,4              (OPEN IN1 AND OUT3)
        PZE     IN1
        TSX     OPEN,4
        PZE     OUT3
LEC     TSX     READ,4              (READ AND LOCATE FROM IN1)
        PZE     IN1,,EOB            (IF EOB, GO TO EOB)
        PZE     EOF,,ERR
IO      IOCP    ZOLEC,,10           (TRANSMIT 10 WORDS TO ZOLEC)
        IOCPN   **,,10              (LOCATE NEXT 10 WORDS)
        IOCP    ZOLEC+10,,10        (TRANSMIT 10 MORE WORDS)
        IOCPN   **,,10              (LOCATE NEXT 10 WORDS)
        IOCD    ZOLEC+20,,10        (TRANSMIT LAST 10 WORDS)
        TSX     WRITE,4
        PZE     OUT3
        IOCP*   IO,,10              (WRITE 10 WORDS
        IOCP*   IO+1,,10            AT A TIME INDIRECTLY)
        IOCP*   IO+2,,10
        IOCP*   IO+3,,10
        IOCT    ZOLEC+20,,10        (LAST 10 WORDS DIRECTLY)
        TRA     LEC                 (GO TO LEC)
EOF     TSX     CLOSE,4             (CLOSE IN1 AND OUT3)
        PTW     IN1
        TSX     CLOSE,4
        PTW     OUT3
        TRA     IOCS                (GO TO IOCS)
EOB     TRA     SYSDMP              (CALL DUMP)
ERR     TRA     SYSDMP
LIST    SVN     2,,3                (FILE LIST OF ONE
        PZE     IN1                 RESERVE GROUP)
        PZE     OUT1
        PZE     OUT3
RECORD  PZE     0                   (RECORD=50 WORDS)
        DUP     1,49
        BCI     1,
ZOLEC   BSS     30
POOL    BSS     602                 (RESERVE 602 WORDS FOR POOL)
        END     BEGIN
$IBSYS
```

Figure 13. Example 1 Using Full IOCS

EXECUTION DECK

Figure 14 shows the *JOB card and the *FILE cards that would be included in the deck for execution of the assembled program shown in Figure 13.

```
7     13                        35        44        55
*JOB  MIN IOCS PROGRAM          3         13000     MINIMUM

7      15 17        28 30                          55
*FILE  1  *UT1      I  HD                          GENERATED FILE
*FILE  2  *UT1      P  HD                          GENERATE FILE
*FILE  3  *UT2      P  HD                          DUPLICATE FILE
```

Figure 14. Execution Deck

Example 2

The following example illustrates use of an internal file and the .STASH routine.

An output file (OUT1) is generated in the same manner as in the preceding example. This file is then converted to an input file by changing the configuration of bits 7 and 8 of Word 2 of the file control block. Records from this file are stashed into an internal file (INT2) until the 20 buffers provided for that file are full.

When the internal file is full, the remaining records on OUT1 are duplicated onto another output file (OUT4). The programs are shown in Figures 15 and 16.

```
1       8        16
$EXECUTE         IBJOB
$IBJCB           GO,BASIC
$PCOL            'GENERATEDFILE','INTERNALFILE'
$GROUP           BUFCT=20,'INTERNALFILE'
$IBMAP  DKNAM    150
OUT1    FILE     GENERATEDFILE,UT1,OUTPUT,BLK=50
OUT4    FILE     SPILLFILE,UT2,OUTPUT,BLK=50
INT2    FILE     INTERNALFILE,INT,OUTPUT,BLK=50
RB      TSX      .OPEN,4              (OPEN FILE TO BE GENERATED)
        PZE      OUT1
        AXT      100,1
        TSX      .WRITE,4
        PZE      OUT1                 (WRITE 50-WORD RECORDS)
        IORT     RECORD,,50
        CLA      =1
        ADD      RECORD               (INCREASE RECORD NUMBER)
        STO      RECORD
        TIX      *-6,1,1              (WRITE 100 RECORDS)
        TSX      .CLOSE,4
        PTW      OUT1                 (CLOSE GENERATED FILE)
        LDI      OUT1+1
        RIL      001000               (SET TO INPUT)
        STI      OUT1+1
        TSX      .OPEN,4              (OPEN GENERATED FILE)
        PZE      OUT1
        TSX      .OPEN,4              (OPEN INTERNAL FILE)
        PZE      INT2
LESE    TSX      .READ,4
        PZE      OUT1,,IOX            (LOCATE NEXT RECORD
        PZE      EOF,,S1              FROM GENERATED FILE)
IO      IORTN    **,,50
NTS2    TRA      **1                  (NTS+1, IF NTS EXIT TAKEN)
        TSX      .STASH,4
        PZE      OUT1,,INT2
        PZE      NTS
        TRA      LESE                 (CONTINUE READING)
NTS     STL      NTS2
        TRA      *+1                  (TRA **4 IF FILE OPENED)
        TSX      .OPEN,4
        PZE      OUT4                 (OPEN SPILL FILE)
        STL      *-3
        TSX      .WRITE,4
        PZE      OUT4                 (WRITE SPILL FILE)
        IOCD*    IO,,50
        TRA      LESE                 (CONTINUE READING)
EOF     TSX      .CLOSE,4             (CLOSE INTERNAL FILE)
        PZE      INT2
        TSX      .CLOSE,4             (CLOSE SPILL FILE)
        PZE      OUT4
        TSX      .CLOSE,4             (CLOSE GENERATED FILE)
        PZE      OUT1
        CALL     EXIT
S1      TRA      SYSDMP               (FILE READ ERROR)
IOX     TRA      SYSDMP               (EOB EXIT CANNOT OCCUR)
RECORD  PZE      0                    (RECORD NUMBER)
        DUP      1,49
        BCI      1,                   (49 BLANK WORDS)
        END
  (END-OF-FILE CARD)
$IBSYS
```

Figure 15. Example 2 Using Library IOCS

```
1       8        16
$EXECUTE         IBSFAP
         *FAP
         COUNT   150
         ABS
         SST
IOCS     EQU     SYSORG
DEFINE   EQU     IOCS+4
ATTACH   EQU     IOCS+8               (DEFINE ENTRY POINTS
CLOSE    EQU     IOCS+10              TO IOCS SUBROUTINES)
OPEN     EQU     IOCS+12
READ     EQU     IOCS+14
WRITE    EQU     IOCS+16
STASH    EQU     IOCS+30
BJ       BOOL    14000
         ORG     BJ
OUT1     BSS     12                   (FCB - GENERATED FILE)
INT2     BSS     12                   (FCB - INTERNAL FILE)
OUT4     BSS     12                   (FCB - SPILL FILE)
START    TSX     DEFINE,4             (SET UP BUFFER POOL
         PZE     POOL                 OF 30 BUFFERS,
         PZE     30,,50               50 WORDS PER BUFFER)
         TSX     ATTACH,4
         PZE     POOL                 (ATTACH FILES TO POOL)
         PZE     LIST,,3
         TSX     OPEN,4               (OPEN FILE TO BE GENERATED)
         PZE     OUT1
         AXT     100,1
         TSX     WRITE,4
         PZE     OUT1                 (WRITE 50-WORD RECORDS)
         IORT    RECORD,,50
         CLA     =1
         ADD     RECORD               (INCREASE RECORD NUMBER)
         STO     RECORD
         TIX     *-6,1,1              (WRITE 100 RECORDS)
         TSX     CLOSE,4
         PTW     OUT1                 (CLOSE GENERATED FILE)
         LDI     OUT1+1
         RIL     001000               (SET TO INPUT)
         STI     OUT1+1
         TSX     OPEN,4               (OPEN GENERATED FILE
         PZE     OUT1                 AS INPUT)
         TSX     OPEN,4
         PZE     INT2                 (OPEN INTERNAL FILE)
LESE     TSX     READ,4
         PZE     OUT1,,IOX            (READ GENERATED FILE)
         PZE     EOF,,S1
IO       IORTN   **,,50               (LOCATE 50 WORDS)
NTS2     TRA     **1                  (NTS+1 IF NTS EXIT TAKEN)
         TSX     STASH,4              (STASH GENERATED FILE
         PZE     OUT1,,INT2           TO INTERNAL FILE)
         PZE     NTS
         TRA     LESE                 (CONTINUE TO STASH)
NTS      STL     NTS2
         TRA     **1                  (**4 AFTER FILE OPENED)
         TSX     OPEN,4               (OPEN SPILL FILE)
         PZE     OUT4
         STL     *-3
         TSX     WRITE,4
         PZE     OUT4                 (WRITE SPILL FILE)
         IOCD*   IO,,50
         TRA     LESE                 (CONTINUE READING)
EOF      TSX     CLOSE,4              (CLOSE INTERNAL FILE)
         PZE     INT2
         TSX     CLOSE,4              (CLOSE SPILL FILE)
         PZE     OUT4
         TSX     CLOSE,4              (CLOSE GENERATED FILE)
         PZE     OUT1
         TRA     IOCS                 (EOJ)
S1       TRA     SYSDMP               (READ ERROR EXIT)
IOX      TRA     SYSDMP               (EOB CANNOT OCCUR)
PCCL     BSS     1562
RECORD   PZE     0
         DUP     1,49
         BCI     1,                   (RECORD+2 -- RECORD+49)
LIST     SVN     2,,2                 (FILE LIST)
         PZE     OUT1
         PZE     OUT4
         SIX     1,,20                (20 BUFFERS FOR
         PZE     INT2                 INTERNAL FILE)
         END     START
$IBSYS
```

Figure 16. Assembly Deck

EXECUTION DECK

Figure 17 shows the *JOB card and the *FILE cards that would be included in the deck for execution of the assembled program shown in Figure 16:

```
7     13                        35        44        55
*JOB  BASIC IOCS PROGRAM        3         14000     BASIC

7      15 17        28 30                          55
*FILE  1  *UT1      P  HD                          GENERATED FILE
*FILE  2  *INT      P                              INTERNAL FILE
*FILE  3  *UT2      P  HD                          SPILL FILE
```

Figure 17. Execution Deck

The LABELS level of IOCS contains the internal routines for processing tape, disk, and drum labels. When IBM standard labels are used, these routines require no programming in the source program. They will function automatically when labeling action is required.

If nonstandard labels are used, or if files are unlabeled, the programmer must follow procedures outlined later in this section.

If labeled files are specified, IOCS will automatically provide for tape reel switching (except for backward reading), and will provide file identification messages as the file is prepared for processing.

Figure 18 shows the format of a labeled file on magnetic tape. The header label is followed by a file mark. The file may contain an optional checkpoint record. If it does, this record is followed by another file mark and then by the actual blocks of data. A file mark separates the last data block from the trailer label, and another file mark follows the trailer label.

Labels are subject to the following conditions:

1. Labeling is not available for files processed on any on-line card equipment.
2. Header and trailer labels are always written in the BCD mode.
3. Tape density requirements, applicable only to 729 magnetic tape unit files, are as follows:
   a. Header and trailer labels may be high or low density.
   b. The file mark following a header label is in the same density as the header label.
   c. The optional checkpoint record and the file mark that follows it are in the same density as the file.
   d. The file itself may be either high or low density.
   e. The file mark preceding a trailer label, the trailer label, and the file mark following the trailer label are in the same density as the file.

IBM Standard Labels

The label routines in IOCS are designed to process IBM standard label formats. Special programming is required when nonstandard labels are used.

The IBM standard label formats are described in the following text.

84-Character Header Labels

This is the standard format for header labels on IBM 729 Magnetic Tape Unit files, IBM 1301 Disk Storage Unit files, and IBM 7320 Drum Storage Unit files. The format for the 84-character standard header label is shown in Figure 19.

In this standard header label, creation date is the date the file was written, and retention days is the number of days the file is to be retained. If an attempt is made to use this reel before creation date plus retention days has been reached, IOCS signals the error by printing a message.

Positions 7-24 and 42 in the standard 84-character header label are not applicable to disk – or drum – storage files.

84-Character Trailer Labels

This is the standard format for trailer labels on IBM 729 Magnetic Tape Unit files, IBM 1301 Disk Storage Unit files, and IBM 7320 Drum Storage Unit files. The format for the 84-character standard trailer label is shown in Figure 20.

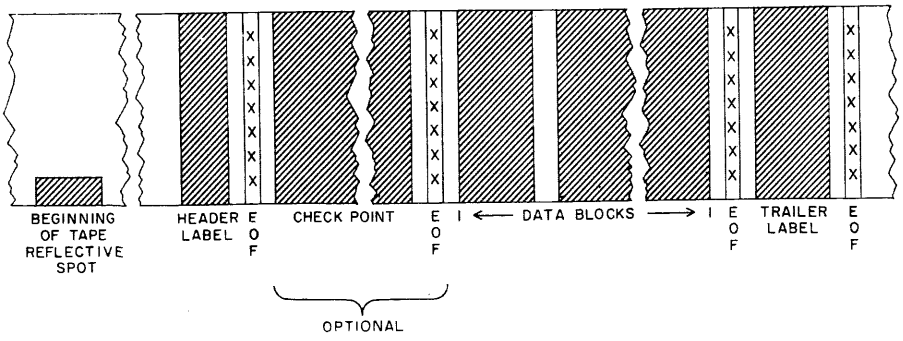


Figure 18. Format of a Labeled Tape File

Field	BCD Position(s)	Field Name	Description
1	1-5	Label Identifier*	1HDRb to indicate a header label.
	6		Blank.
2	7		Blank.
	8-12	Reel Serial Number	A five-character identification code assigned to the reel when it enters the installation. This number normally appears on the outer surface of the reel for visual identification.
3	13		Blank.
	14-18	File Serial Number*	This field is the same as the reel serial number of the first or only reel of the file.
4	19		Blank.
	20-23	Reel Sequence Number*	A four-digit number (0000-9999) that is the order of this reel within the file.
	24		Blank.
5	25-30	Creation Date*	The year and day of the year on which the file was created. The year occupies the first two positions (00-99) and the day occupies the last three positions; e.g., December 30, 1963 would be entered as 63b364.
6	31-32		Blank.
	33-36	Retention Days*	The number of days after the creation date (0000-9999) that this file is to be retained.
7	37		Blank.
	38	Density Indicator	This character indicates the density of the file as: 0 - Low Density 1 - High Density
8	39	File Mode**	This character indicates the mode of the file as: 0 - BCD 1 - Binary 2 - Mixed Mode
9	40	Check Sum Indicator**	This character indicates the presence (1) or absence (0) of check sums.
10	41	Block Sequence Indicator**	This character indicates the presence (1) or absence (0) of block sequencing.
11	42	Checkpoint Indicator**	This character indicates the presence (1) or absence (0) of a checkpoint following the label on the file.
12	43-60	File Identification*	An 18-character field that identifies the file.
13	61-72	For Optional Use***	Not used, but reserved for 709/7090/7094 sort compatibility.
14	73-84	For Optional Use***	Positions 73-84 may be employed as an area for nonstandard label data at the programmer's option.

* This field is verified by IOCS.
** This field overrides file control block specifications.
*** This field is not interpreted by IOCS.

Figure 19. Format of the IBM Standard 84-Character Header Label

Field	BCD Position(s)	Field Name	Description
1	1-5	Label Identifier*	1EORb to indicate an end-of-reel trailer label or 1EOFb to indicate an end-of-file trailer label.
2	6-7		Blank.
	8-12	Block Count*	This field specifies the number of blocks written on this reel of the file (excluding labels, checkpoints, and file marks). The block count is given in trailer labels only.
3	13		Blank.
	14-18	Unit Control Word	This is the location in storage of the first word of the unit control block which specifies the unit on which this file was prepared.
4	19-84	For Optional Use***	Positions 19-84 may be employed as an area for nonstandard label data at the programmer's option.

* This field is verified by IOCS.
*** This field is not interpreted by IOCS.

Figure 20. Format of the IBM Standard 84-Character Trailer Label

In the 84-character standard trailer label, the following actions will be taken for the indicated fields:

Label Identifier: 1EORb causes reel-switching to occur automatically during reading. No end-of-file indication is given the programmer when an end-of-reel trailer label is encountered. 1EOFb causes an end-of-file exit from the routine. If the file is referred to again, another end-of-file indication is given.

Block Count: The block count is checked for every segment of an input file (reel, disk, or drum). If it differs from the number of blocks actually read in, IOCS reports a sequence error. In this case, it should be noted that a sequence error for the reel can occur even though each block is not being checked for sequence. During writing, an end-of-reel trailer label is prepared by IOCS when the end of reel is encountered. An end-of-file trailer label is prepared when the file is closed.

120-Character Header and Trailer Labels

This is the standard format for both header and trailer labels for files on IBM 7340 Hypertape Drives. The format of the standard 120-character header or trailer label is shown in Figure 21. The actions indicated above for Label Identifier and Block Count are also taken for these trailer labels.

Blank Reels

Use of the LABELS level of IOCS requires that all files on magnetic tape be labeled before processing is begun. A temporary header label should be placed on each otherwise blank reel by a utility program or by a card-to-tape operation. The format of required header labels on blank reels is shown in Figure 22.

NOTE: Blank labels are not required for disk and drum output files.

Field	BCD Position(s)	Field Name	Description
1	1-5 6	Label Identifier*	1HDRb to indicate a header label, 1EORb to indicate an end-of-reel trailer label, or 1EOFb to indicate an end-of-file trailer label. Blank.
2	7-10	Retention Period*	Number of days (0001-9999) this file is to be retained after the creation date. It should be 0000 for files where the expiration date is not applicable.
3	11-15	Creation Date*	The year and the day of the year on which the file was created. The year occupies the first two positions (00-99), and the day of the year occupies the last three positions; e.g., December 30, 1963 would be entered as 63364.
4	16-25	File Identification*	A 10-character field that identifies the file uniquely.
5	26-30	File Serial Number*	This field is the same as the reel serial number of the first or only reel of the file.
6	31-35 36	Reel Serial Number	A 5-character identification code assigned to the reel when it enters the installation. This number normally appears on the outer surface of the reel for visual identification. Blank.
7	37-40 41	Reel Sequence Number*	A four-digit number (0001-9999) that is the order of this reel within the file. Blank.
8	42-44	Reserved	This field is reserved for future IBM use.
9	45	Density Indicator***	This indicator is 0 for Hypertape labels.
10	46	Check Sum Indicator**	This character indicates the presence (1) or absence (0) of check sums.
11	47	Block Sequence Indicator**	This character indicates the presence (1) or absence (0) of block sequencing.
12	48	Recording Mode Indicator**	This character indicates mode: 1 - Binary 2 - BCD 4 - Mixed Mode
13	49	Recording Technique Indicator***	This field specifies the number of bits recorded as one byte: 8 (but only six of the eight bits are used as data to be processed.)
14	50	Data Processing Technique Indicator***	This field specifies the number of bits to be processed as one byte: 6.
15	51-54	Creating System	This field specifies the system that created the file (e.g., 7090 or 7094).
16	55	Record Format***	This character indicates the record format of this file.
17	56-60	Record Length***	For fixed-length records, this field specifies the number of characters in each data record; for variable-length records, it specifies the number of characters in the largest possible data record in this file.
18	61-65	Block Size***	For fixed-length records, this field specifies the number of logical data records in each block; for variable-length records, it specifies the number of characters in the largest possible block in the file.
19	66	Checkpoint Indicator**	This character indicates the presence (1) or absence (0) of checkpoints in the file.
20	67-72	Block Count*	This field specifies the number of blocks written on this reel of the file (excluding labels, checkpoints, and file marks). The block count is given in trailer labels only.
21-26	73-100	Reserved	These fields are reserved for future IBM use.
27	101-120	For Optional Use***	Positions 101-120 may be employed as an area for additional label data, at the programmer's option.

* This field is verified by IOCS.
** This field overrides file control block specifications.
*** This field is not interpreted by IOCS.

Figure 21. Format of the IBM Standard 120-Character Header and Trailer Labels

Labeling and Label Checking

Input File Header Labels

When the file is opened, and at the beginning of each new reel, if it is a multireel file, the label is checked to ensure that the correct reel is being processed. The following conditions must be met for an input label to be valid:

1. The file serial number given in a control card or pseudo-operation must agree with the file serial number in the label. If the file serial number is omitted from the $LABEL card, the *FILE card, or the LABEL pseudo-operation, this test is not performed.

2. The reel sequence number in the label must agree with the reel sequence number given in a control card or pseudo-operation, unless reel-switching has occurred. If reel-switching has occurred, IOCS will have advanced the reel sequence number stored in the file control block. In Library IOCS, if the reel sequence number is omitted from the $LABEL card or the LABEL pseudo-operation, this test is not performed.

Field	BCD Position(s)	Field Name	Description
1	1-6	Label Identifier	1BLANK to identify the reel as a blank reel available for use.
2	7 8-12	Reel Serial Number	Blank. A five-character identification code assigned to the reel when it enters the installation. This number normally appears on the outer surface of the reel for visual identification.
3	13-84 (729, 1301, or 7360) or 13-120(7340)		Arbitrary - IOCS does not use these fields on a 1BLANK header label.

Figure 22. Format of Required Labels on Blank Reels

3. The file identification given in a control card or pseudo-operation must agree with the file identification in the label. If no file identification is given, this test is bypassed.

If any of the above checks fail, and the label search option has not been specified, a message is printed indicating that a label error has occurred and the label in error is printed. A halt will then occur. By sense switch control, the operator may accept the reel as valid, in which case he may proceed. Otherwise, he must mount the correct reel to be processed.

If the label search option has been specified, a message containing the file name and a search skip list is printed. Searching occurs until the correct label is found or the operator intervenes by using a sense switch.

Additional Input File Header Labels

The system will automatically bypass all records between the header label and the next file mark. The system provides no automatic way to create these records, and this feature is provided for compatibility with existing labeling schemes. If these records are to be processed, the file should be treated as an unlabeled multireel file. The programmer can then process the label records with his own routines.

Input File Trailer Labels

Each labeled file is assumed to be a multireel file. The occurrence of an end-of-reel trailer label causes tape switching to the next reel, which is checked before processing is continued. The occurrence of an end-of-file trailer label or, in fact, any record other than an end-of-reel trailer where a trailer label is expected, will cause an end-of-file exit to be taken from the read routine. The block count, which appears in every trailer written by the system, is checked. A sequence error exit is taken if the block count does not agree with the number of blocks read.

Output File Header Labels

Every reel upon which a labeled output file is to be prepared must have a label written on it. This label may be a 1BLANK type or a 1HDR type on which creation date plus retention days has been reached. If these conditions are not found, a message is printed indicating that a label error has occurred. The label is then printed and a stop occurs. The condition may be ignored by the operator, in which case a dummy label is generated. The file serial number is set to ***** for a reel that does not contain a 1BLANK or 1HDR header label.

Output File Trailer Labels

If, during the course of writing an output reel, the end-of-tape reflective spot is sensed, a file mark, end-of-reel trailer label, and another file mark will be written. The reel will then be rewound and unloaded, and a message will be printed advising the operator to remove that reel.

If the file is assigned two different units, reel-switching occurs and processing continues. Otherwise, a delay will occur until the present reel is removed and a new blank reel with a temporary label is mounted.

Disk and Drum File Labels

It is emphasized that protection cannot be guaranteed to files located on a disk or drum by signifying a retention period. The HA2 identifier should be written on those cylinders which are to contain the IOCS-written file, before IOCS is used. This identifier can be written by using the System Editor or a utility program. The specifications for disk and drum editing can be found in the publication *IBM 7090/7094 IBSYS Operating System: System Monitor (IBSYS)*, Form C28-6248.

The functions and procedures of label facilities will be simulated. Blank labels are not required for new output files.

Nonstandard Labels

Nonstandard labels are those that do not conform to the IBM standard formats.

Nonstandard labels on IBM 729 Magnetic Tape Unit files, 1301 Disk Storage files, and 7320 Drum Storage files can be processed either by treating the labels as files consisting of a single record and processing each label individually within the main program, or by using IOCS to transfer to a nonstandard label routine provided by the programmer.

IOCS does not permit nonstandard label formats or labeling procedures to be used with files on Hypertape.

Nonstandard Labels for 729 Tape, 1301 Disk, and 7320 Drum Files

If the nonstandard labels on IBM 729 Magnetic Tape Unit files, disk storage files, or drum storage files differ radically from the standard format, the programmer must replace the IOCS labeling procedures with routines of his own. For instance, IOCS is unable to handle labels in which the first field (label identifier) is not in the standard form to indicate whether the label is a header label , an end-of-reel trailer label, or an end-of-file trailer label.

When such nonstandard labels are used, the programmer must specify that the files are unlabeled and must treat labels, as well as data segments, as separate files. Each header label would be treated as a separate file, data records would be a file, and each trailer label would be a file. The programmer must provide routines within his main program to process the labels in the manner he desires. IOCS would not be able to distinguish labels from data records, and IOCS label procedures would not apply.

If, however, the nonstandard labels do not differ significantly from the standard format, they can be handled within IOCS by using a nonstandard label routine provided by the programmer. When IOCS is to check or prepare a label, it transfers to the programmer's nonstandard label routine to accomplish the required labeling action.

In this case, the programmer must provide IOCS with the location of the first word of his nonstandard label routine. This routine would actually consist of five routines for checking and preparing header and trailer labels. The first instructions in the nonstandard label routine would be a set of transfer instructions that would direct IOCS to the programmer's own labeling routines when label action is required.

The set of transfer instructions should be set up as follows:

```
mylbls      TRA     lbrtn1
(+1)        TRA     lbrtn2
(+2)        TRA     lbrtn3
(+3)        TRA     lbrtn4
(+4)        TRA     lbrtn5
```

where:

mylbls
is the symbolic location of the first word of the programmer's nonstandard label package.

lbrtn1
is the symbolic location of the programmer's routine to check input file header labels.

lbrtn2
is the symbolic location of the programmer's routine to check input file trailer labels.

lbrtn3
is the symbolic location of the programmer's routine to check the label on tape to be used for output.

lbrtn4
is the symbolic location of the programmer's routine to prepare output header labels.

lbrtn5
is the symbolic location of the programmer's routine to prepare output trailer labels.

The programmer must provide IOCS with the location of the first word of the nonstandard label routine according to the form of IOCS he is using, as follows:

1. In Library IOCS—The external name of the first word of the nonstandard label routine is specified in the nonstandard label option field (NSLBL) of the FILE pseudo-operation.

2. In Full IOCS—The location of the first word of the nonstandard label routine is specified in the decrement field of the entry for that file in the list of files attached to a buffer pool. It is done in the following manner:

```
filist      SVN     1,,4
            PZE     file1
            PZE     file2,,mylbls
```

where:

file1
is the symbolic name of a file for which IOCS standard label routines are to be used.

file2
is the symbolic name of a file with nonstandard labels.

mylbls
is the symbolic location of the first word of the nonstandard label routine for that file.

When *filist* is attached to a buffer pool with a .ATTAC routine, IOCS automatically handles placement of *mylbls* for proper transfer to the nonstandard routine.

When nonstandard label routines are provided, the following label procedures occur:

1. Header and trailer labels are read and written by IOCS.

2. All file marks associated with the labels are read and written by IOCS.

3. The standard 14-word area at symbolic location .LAREA (LAREA in Full IOCS), within the IOCS communications region, will be used for holding, checking, and forming label images. (Appendix A contains the relative location of .LAREA and LAREA to the origin of IOCS.)

4. All system messages and actions will apply.

5. At each TSX to the nonstandard routine, index register 2 will contain the 2s complement of the location of the first word of the file control block for the file being processed.

In his nonstandard routine the programmer must save and restore the contents of all index registers and sense indicators.

Assume that the programmer has chosen MYLBLS as the symbolic location of the first word of his nonstandard labels routine. IOCS makes the following TSX transfers when labeling action is needed and the routine must contain the indicated return transfer for the specified condition:

1. Checking input file header label
   a. IOCS starts this function by means of TSX MYLBLS,1.
   b. The routine must return to IOCS by means of the following:
      TRA 1,1 if the label is correct.
      TRA 2,1 if the label is invalid.

Bit 35 of the seventh word of the label area must be 1 upon either return if a checkpoint record is to be skipped.

2. Checking input file trailer label
   a. IOCS starts this function by means of TSX MYLBLS+1,1.
   b. The routine must return to IOCS by means of
      TRA 1,1.
      Upon return, the sign of the MQ must be:
      + if the label is an end-of-reel trailer label. Reel switching will occur.
      − if the label is an end-of-file trailer label. The end-of-file will occur.
3. Checking label on tape to be used for output
   a. IOCS starts this function by means of TSX MYLBLS+2,1.
   b. The routine returns to IOCS by means of the following:
      TRA 1,1 if the reel may be used.
      TRA 2,1 if creation date plus retention days has not been reached.
      TRA 3,1 if the reel cannot be used.
4. Preparing output header label
   a. IOCS starts this function by means of TSX MYLBLS+3,1.
   b. The routine must return to IOCS by means of
      TRA 1,1.
5. Preparing output trailer label
   a. IOCS starts this function by means of TSX MYLBLS+4,1. Upon entry to this routine, the location .LAREA will contain the following:
      1EORbb if an end-of-reel trailer label is to be prepared.
      1EOFbb if an end-of-file trailer label is to be prepared.
   b. The routine must return to IOCS by means of
      TRA 1,1.

Provision for Additional Information in Hypertape Labels

NOTE: In the following discussion, the Full IOCS programmer should substitute LAREA wherever .LAREA appears and LAREA1 wherever .AREA1 appears. The relative locations of these words are indicated in Appendix A.

IOCS makes no provision for nonstandard 120-character labels.

However, label field 27 (BCD positions 101-120) may be used for additional information. If the programmer wants to use this label field for additional information, he must provide his own routine for processing the information.

After checking or creating a label, IOCS transfers to the programmer's routine, which can then check or insert the additional information into the last two BCD positions of .AREA1+2 and into .AREA1+3 through .AREA1+5. In addition, if the tape being created is to be used on a different computer (e.g., the IBM 7074), the appropriate information may be placed into .LAREA+9 and into the first five BCD positions of .LAREA+10. The sixth BCD position of .LAREA+10 must not be destroyed.

The programmer notifies IOCS of the location of his Hypertape additional information routine, as follows:

1. In Library IOCS—By providing the external name of the first word of the routine in the NSLBL field of the FILE pseudo-operation.
2. In Full IOCS—By providing the symbolic location of the first word of the routine in the decrement of the appropriate file entry in the list of files attached to a buffer pool. The symbolic location of the routine is specified in the same manner as the symbolic location of the first word of a nonstandard label routine.

The following functions will be performed if the additional information routine is specified:

1. All header and trailer labels will be read, written, and checked by IOCS.
2. All file marks associated with labels will also be read and written by IOCS.
3. Two label areas within the IOCS communications region will be used. They are the 14-word area at .LAREA and the six-word area at .AREA1.
4. At each TSX to the programmer's additional label information routine, index register 2 will contain the 2s complement of the location of the file control block for the file on which the label action is being performed.

The programmer need not save the contents of index registers or sense indicators in his routine. The return from the routine must be TRA 1,4.

Assume that the programmer has designated the first word of his additional label information routine as ADDLBL. IOCS would make the following transfers for the indicated label actions:

1. Checking input file header label — IOCS starts this function by means of TSX ADDLBL,4.

2. Checking input file trailer label – IOCS starts this function by means of TSX ADDLBL+1,4.

3. Preparing output file header label – IOCS starts this function by means of TSX ADDLBL+2,4.

4. Preparing output file trailer label – IOCS starts this function by means of TSX ADDLBL+3,4.

Unlabeled File Procedures

Although the handling of labeled files is entirely automatic, IOCS is equally capable of processing unlabeled files. The difference is that multireel file handling is only semiautomatic for unlabeled files, because the occurrence of the file mark can have several meanings.

Single Reel Unlabeled Files

An unlabeled file that is contained on one reel of tape may have any number of file marks. During reading, the detection of each file mark temporarily suspends buffering on that reel. When a file mark is detected by the read routine, the end-of-file exit is taken. If the file is again referred to by a .READ calling sequence, buffering will continue until the next file mark is encountered. It is the programmer's responsibility, in this case, to determine which file mark signifies the actual end of the file. No reel switching is possible for input files of this type.

If end of tape is encountered while writing a supposedly single reel unlabeled file, a reel switch occurs and processing continues. For some files, such as peripheral output files, no harm occurs from this action. However, if the file is to be processed by IOCS at a later time, the file is not describable. (It is a multireel file with more than one file mark on a reel, a situation which is not allowable.)

Multireel Unlabeled Files

A multireel unlabeled file may have only one file mark —the one that signifies the end of the reel. This file mark is written automatically when the end-of-tape reflective spot is sensed while writing. Reel switching occurs after the file mark is written.

When a file mark is detected while reading a multireel unlabeled file, all buffering is suspended until the end of file is reached. An end-of-file exit is taken once per reel for each file mark encountered and, if the file is again referred to by a read operation, reel switching occurs and buffering is resumed. For this type of file, the programmer must have a recognizable data record to indicate that the end of file has been reached.

Multifile Reels

Several files that appear on the same reel may be processed automatically by closing each processed file without rewinding and then opening the subsequent file. IOCS does not automatically rewind any reel, except at reel switching. Hence, a single reel may contain several complete files and the first portion of a multireel file. Labeled and unlabeled files may be mixed upon the reel, provided some precautions are observed in the usage of IOCS routines. For example, the backspace file routine with a count greater than 1 may not result in positioning the read-write head at the beginning of data of a given file, because of the occurrence of file marks for labels, trailers, and checkpoint records. It is also unlikely that the open routine with label search would find the correct file.

Density Considerations

Density can only be changed on 729 tapes. If a reel is to be involved in a restart, IOCS allows a tape density change, if any, only after the first file mark on the reel. The reason for this restriction is that the restart routine must know the density of every block and file mark to be skipped over in the process of tape positioning. Arbitrary changes in density are not permitted because there is no way IOCS can detect the changes.

Tape densities of a file and its header label are specified in a control card or pseudo-operation and are stored in the file control block. The density of a checkpoint record on a labeled file is the same as that of the main body of the file. In the case of a mult-file reel, a density change is permitted only after the header label of the first file and the remainder of the reel must be the same density.

A standard density option is provided for labeled files. If this option is indicated in a control card or pseudo-operation, the density of the label will be that specified for the file. Furthermore, when an output file is prepared, the density of the label on the reel to be checked for validity is always the same, low or high, according to installation choice. It should be emphasized that if the standard option is not used, the density of the header label of the blank reel must agree with the density specified for the label of the file to be written.

The backspace file routine cannot be used on a reel with mixed densities because difficulty will be encountered in backspacing over files of different densities. The backspace file routine also cannot be used to reposition the read-write head in front of the data of a labeled file when the label and the body of the file are in different densities.

NOTE: It is strongly recommended that density changes be avoided.

Mode Considerations

The body of a file normally contains information recorded in only one mode: binary or BCD. However, IOCS is capable of processing a mixed-mode file if

1. The file is described as being a mixed-mode file in a control card or pseudo-operation, and
2. One of the standard look-ahead words, described in Appendix F, is attached to the end of each physical record of the file.

When reading a mixed-mode file which satisfies these two conditions, IOCS examines each look-ahead word to determine the mode of the next physical record. No indication is given to the programmer as to what that mode is.

In order to enable IOCS to write a mixed-mode file, the user must place the proper look-ahead words in the output buffer. Since these words cannot be properly placed by programs written in the FORTRAN IV or COBOL languages, mixed-mode files can be written with compiled programs only when output is handled by subroutines written in the MAP language. IOCS will always write the first physical record of a file in the mode specified on the $FILE or *FILE card.

Block Sequence Numbers and Check Sums

Every file control block carries a block sequence number for the current reel of the file. For a labeled file, this sequence number is always written in the trailer label at the end of each reel and is checked later, when the trailer label is read.

For a binary file, a block sequence word can be appended to the end of each block, regardless of whether the file is labeled or unlabeled. This word contains the sequence number of the block in its address and may also contain an 18-bit folded check sum for the block in the left half of the word.

The check sum is formed by computing a logical sum of the entire data in a block, excluding its block sequence word, and then logically adding the left and right halves of this sum. This computation requires the execution of at least two instructions for each data word, and hence may be costly to the program. On the other hand, forming and checking a block sequence number is accomplished automatically by IOCS in a few programming steps for each block, and it is quite helpful in detecting certain machine malfunctions, such as tape shift-register trouble.

The proper entries must be made in control cards if the programmer desires formation of block sequence words and examination of check sums. Block sequencing must be specified if the check sum option is to be used.

In order for a block sequence word to be generated and checked, each buffer requires an additional word. The programmer must be sure that this additional word is available, as follows:

In Library IOCS: By including the additional word when block size is specified in a control card or pseudo-operation.

In Full IOCS: By including the additional word when space is reserved for buffer pools and when the pools are defined with the DEFINE routine.

However, generation and checking of this word is accomplished internally, and the extra word is not evident during processing of data.

Checkpoints and Restarts

A checkpoint may be initiated by one of three conditions:

1. Beginning of reel on a labeled file.
2. Transfer to the checkpoint routine (.CKPT or CKPT) within a program.
3. Operator intervention by use of the sense switch assigned to checkpoint control. Sense switch 2 is the standard sense switch for checkpoint control.

In the first condition, a checkpoint is written either after the label on an output file or on the checkpoint file, depending on which file has been specified in a control card or pseudo-operation. In the second and third conditions, the checkpoint is always written on the checkpoint file.

The checkpoint written by IOCS contains all of the information necessary to restart the program from the point at which the checkpoint was taken.

When a checkpoint is executed, a checkpoint identification and restart code is printed and all input/output activity ceases. The checkpoint identification code is a sequence number that is increased by 1 each time a checkpoint record is written. The contents of all index registers, the condition of sense indicators and sense lights, the settings of sense switches, and the contents of core storage are recorded.

A checkpoint record of two blocks is written, followed by a file mark. The first block contains information later used by IOCS to provide mounting instructions and to reposition tapes. The status of sense switches is also recorded in this block. The second block contains the contents of core storage and the conditions of registers and indicators.

If a checkpoint is initiated by use of the checkpoint routine, and either a checkpoint file was not specified earlier in a control card or pseudo-operation or the checkpoint file was not opened, a transfer back to the main program will occur immediately.

Restarting a program from a checkpoint must be initiated by the operator. It cannot be initiated by the stored program. Once initiated, the restart routine uses the information in the checkpoint record to print messages to assist the operator in mounting tapes and setting sense switches. The information is also used to restore core storage and all indicators and registers to their status when the checkpoint was taken.

The execution of the program is resumed from the point in the program at which the checkpoint record was written.

Restarts can be initiated by using the $EXECUTE RESTART control card. This card passes control to the Restart Program described in the publication *IBM 7090/7094 IBSYS Operating System, Operator's Guide,* Form C28-6355.

In Full IOCS, restarts can also be initiated by using the *RESTART Preprocessor control card described in the section "Control and Loading Information for Full IOCS."

Interchanging Hypertape Cartridges with Other IBM Data Processing Systems

If the programmer is concerned with the need for interchanging 7340 Hypertape cartridges with other IBM data processing systems (7074, 7080, or another 7090/7094) the following tape formats should be used:

1. Fixed-length records with fixed blocking factor – Tape records as well as logical data records must be a multiple of 30 characters in length. All data records must have a terminating record mark. The multiple of 30 characters includes the terminating record mark.
2. Variable-length records with variable blocking factor – Every logical data record must be a multiple of 30 characters in length. The multiple of 30 characters includes a terminating record mark and a five-character field, located at the immediate start of each data record. This length field is right-justified, with leading zeros included when necessary.

Further requirements for Hypertape exchange among 7000 series systems are:

1. All tape information must be in BCD form
2. The delta character must not be written on tape.
3. All data must be in the unpacked mode.

Sequential Processing Using Disk and Drum Storage Units

Sequential processing of a file on 1301 Disk Storage or 7320 Drum Storage requires several special considerations.

The format track must be written so that each track contains one physical record of 465 words for disk or 524 words for drum. The HA1 for each track must be the same as the seek address, the HA2 must be one word, and the record address must be one word.

IOCS will read or write one full track for every physical buffer less than or equal to 465 words for disk or 524 words for drum. Additional tracks will be used in sequence if the physical buffer is greater than 465 (disk) or 524 (drum) words. IOCS will *not* pack or block physical records on a track. Therefore it is suggested that a buffer size of 465 (disk) or 524 (drum) be used with output files for maximum efficiency.

If the disk or drum storage file is an input file, it must have been created by IOCS. This is required because IOCS simulates logical definitions, such as files, records, load point, and end of tape, by placing special

flags in the record address of each track when the file is created. Hence, the record address of each track is reserved for use by IOCS.

All operations will be in six-bit mode.

Checkpoints will not be recorded on a disk or drum even if they are specified in a control card or on a label.

Before a disk or drum unit can be used for output, the HA2 addresses must previously have been written using the System Editor or a utility program for those cylinders to be used by the file. This HA2 identifier must be specified in the LABLE pseudo-operation, $LABEL card, or *FILE card. A no-record-found indication by the IBM 7631 File Control unit because of improper matching of HA2 will terminate a job with a core storage dump.

For simplicity, Library IOCS names for the random processing routines are used in the following text. The explanations, however, are equally applicable to the same routines in Full IOCS. Calling sequences are illustrated side-by-side.

For reference purposes, the symbolic names of the routines in both forms of IOCS are as follows:

LIBRARY IOCS			FULL IOCS
.RANCL			RANCLS
.RANDE			RANDEF
.RANRE			RANREQ
.RANRP			RANRPL
.RANCA	} Flag	{	RANCAV
.RANFL	} Words	{	RANFLG

The Library IOCS programmer does not specify Random IOCS in a control card. The IBJOB Loader determines whether the random routines are needed and loads them if they are required.

The Full IOCS programmer must specify Random IOCS by punching the characters RAND in columns 63-66 of the *JOB card.

Random IOCS is a modular program that uses IOEX to make it compatible with the System Monitor, under which it operates. Random IOCS does not use any of the parts of sequential IOCS, and the sequential routines need not be in core storage while Random IOCS is being used. The programmer using IOCS may want to use sequential IOCS for other purposes; it is therefore possible to combine Random IOCS with any one of the configurations of sequential IOCS (except FIOCS) when choosing the IOCS routines to be loaded with the object program.

Random IOCS is designed to facilitate random processing. In random processing, the data files being processed are not necessarily in sequence with respect to one another. One file is read sequentially, and the records of this file indicate the locations of the records of the other files which must be processed at the same time. All of the files that are in a different order from the file which is being read sequentially must be on either an IBM 1301 Disk Storage unit or IBM 7320 Drum Storage unit.

In order to simplify terminology, records from the file being read sequentially will be referred to as *alpha records*. Records from disk or drum storage will be referred to as *beta records*. Alpha records may be obtained by the use of sequential IOCS, or they may be read through IOEX by a routine written by the programmer. The order in which alpha records are read and the method by which they are obtained do not affect the operations of Random IOCS.

The beta records usually form some type of master file and have a permanent arrangement on disk or drum storage. Random IOCS is not affected by the way in which the beta file was originally written. Most beta files can be loaded by a utility routine, and are essentially sequential.

A FILE card is not needed in either Library IOCS or Full IOCS for the file containing the beta records.

Structure of Random IOCS

Random IOCS is designed to obtain the set of one or more beta records which correspond to a given alpha record, and to schedule the execution of the routine, written by the programmer, which processes these records. The programmer's main program reads an alpha record and informs IOCS of its location, the locations of the beta records to which it corresponds, and the location of the routine (or routines) to process them. IOCS then reads in the beta records and releases control to the processing routine when a read operation is complete. There is ordinarily a lapse of time between the request for a beta record and its delivery. Random IOCS allows this time to be used by the main program for computation, including further requests for beta records.

In order to allow several requests to be made while IOCS is reading beta records, a file (*queue*) of such requests is maintained. Whenever trapping occurs after the reading of a beta record, Random IOCS performs only the necessary updating of the queue. The programmer's processing routine is not executed at trap time.

When the main program makes a request, Random IOCS records the request and examines the queue to determine whether or not a beta record has been read since the previous request. If a beta record has been read, a processing routine for this beta record is entered. If there is no beta record waiting to be processed, Random IOCS returns control to the main program. It does not retain control and wait for the completion of a beta activity unless specifically requested to do so.

The order in which processing takes place depends on the order in which alpha records are obtained, on the constraints imposed on processing by the various types of beta records and alpha records, and on the time necessary to read a given beta record.

Alpha records may specify either of two types of processing. If the alpha records are independent of one another, processing is said to be *full-random.* However, an alpha record may force processing of all alpha records which have preceded it. This is called *force-sequential processing.*

Beta records are of two types: *mutually independent* and *mutually dependent.* If the records are mutually independent, processing can occur as each record of a set becomes available. If a group of beta records within a set are designated mutually dependent, processing can occur only when all of the group is present. Only one group of mutually dependent beta records is allowed within the set of beta records corresponding to a given alpha record, but the group may include the entire set.

By inspecting information sent from the processing routine, Random IOCS determines whether to release the buffer in which a beta record is held, or, first, to write it back on the unit from which it came, subsequently releasing the buffer.

Figure 23 is a diagram of the structure of Random IOCS.

Use of Random IOCS

In order to make use of the facilities of Random IOCS, the programmer must do the following:

1. Reserve space for both the disk or drum record holding area (*drha*) and the request entry queue (*queue*); they will be defined by Random IOCS.
2. Write a main program which reads alpha records, derives from them the addresses of the beta records required, and requests that Random IOCS read them.
3. Write one or more processing routines which use the information supplied by Random IOCS to process the alpha and beta records.
4. Write an error routine to take action in case the beta record requested cannot be found.

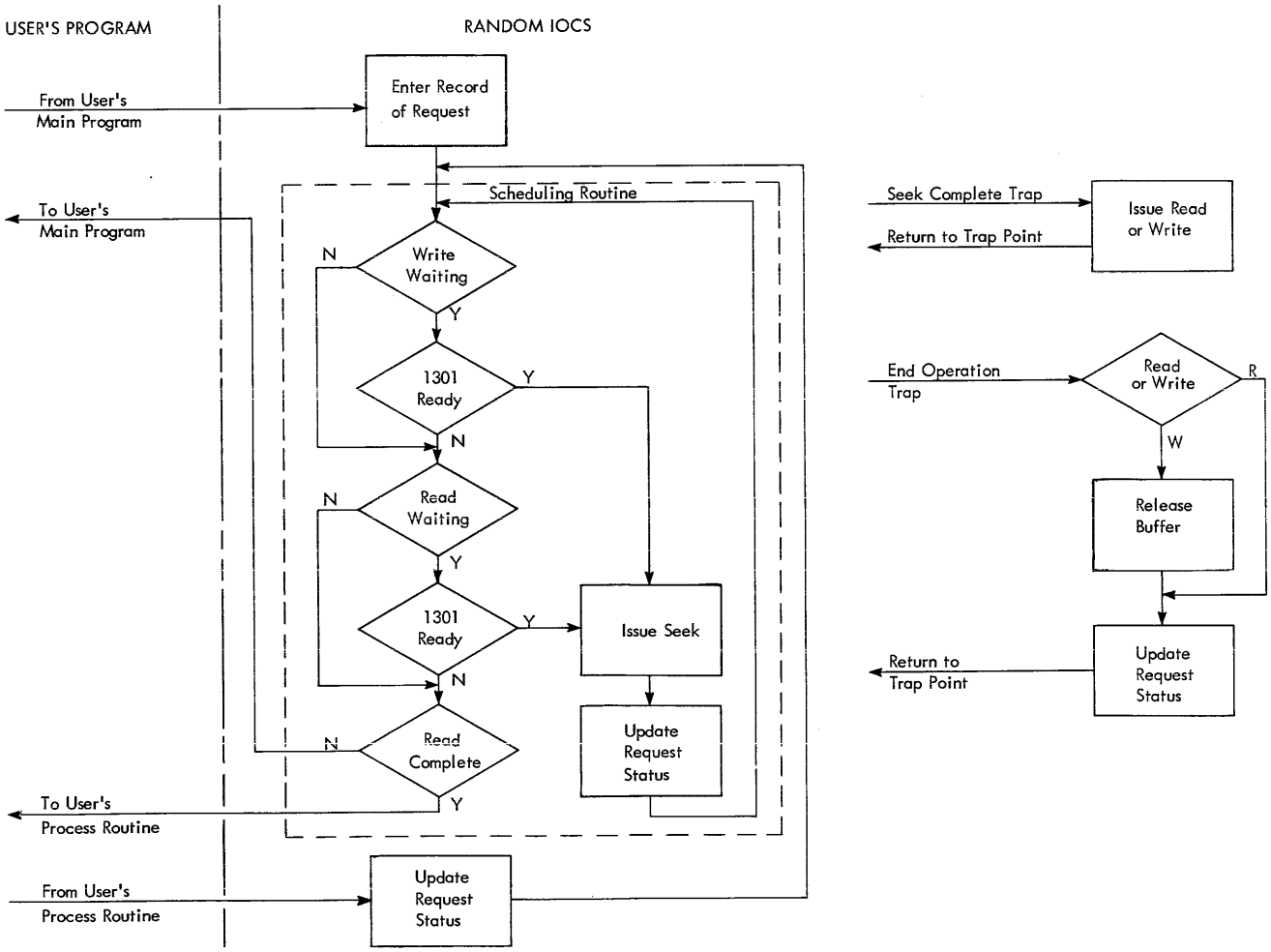


Figure 23. Diagram of the Structure of Random IOCS

Reserving Buffer Areas

To use Random IOCS, the Library IOCS and Full IOCS programmers must reserve a disk/drum record holding area (*drha*) of N(R+1) +1 words. R is the length of a single beta record, and N is the number of beta record buffers to be used. Thus, if the programmer wants five beta record buffers, and the beta records with which he will be working are each ten words long, he must reserve a disk/drum record holding area 56 words long.

The programmer must also reserve a request entry queue (*queue*) of s*7+1 words. S is the maximum number of beta record requests which random IOCS can be processing at any given time. These areas need not be contiguous. If M is the number of alpha record buffers defined by the programmer and m is the maximum number of beta records called by one alpha record, the recommended relationship between the variables N,S,M, and m is as follows:

S = N = m*M

Random IOCS can operate if S is smaller than m*M, or if N is smaller than S (or both), but at decreased efficiency. S and N must be at least equal to m. The number M should be at least twice as large as the number of 1301 and 7320 modules referred to by the alpha file, preferably more.

Routines Added to Sequential IOCS

Since the programmer may want to use sequential IOCS to read alpha records, two subroutines are provided in sequential IOCS.

Locate with Retention (.READR)

Alpha records may be read with a locate-with-retention routine, using the following calling sequence:

LIBRARY IOCS		FULL IOCS	
TSX	.READR,4	TSX	READR,4
PZE	file,,eob	PZE	file,,eob
PZE	eof,,err	PZE	eof,,err
IORTN	**,**	IORTN	**,**

where:

file
is the symbolic name of the alpha file (the symbolic location of the first word of the file control block).

eof
is the location of the programmer's end-of-file routine.

eob
is the location of the programmer's end-of-available-buffers routine.

err
is the location of the programmer's routine for unrecoverable input/output errors.

This routine allows retention of a buffer until it is released by the programmer's program, overriding the normal process whereby the buffer is returned to the pool after the next reference to the file. This feature is necessary to allow retention of the alpha record until it can be processed. The alpha file or files should be the only files attached to their buffer pool.

Release Retained Buffer (.RELES)

After the alpha record which it holds has been processed, a buffer can be released by using the following calling sequence:

LIBRARY IOCS		FULL IOCS	
TSX	.RELES,4	TSX	RELESE,4
PZE	locbuf,,file	PZE	locbuf,,file

where:

locbuf
is the location of the buffer. This location is inserted into the address portion of the IORTN command during execution of the locate-with-retention routine.

file
is the symbolic name of the alpha file (the symbolic location of the first word of the file control block).

Routines in Random IOCS

.RANDE Calling Sequence

The first entry to Random IOCS defines the buffer and queue areas in the locations reserved by the programmer for that purpose. The .RANDE calling sequence defines both N beta record buffers, of effective size R, in the drha; and S entry request buffers in the queue area. The format of the calling sequence is as follows:

LIBRARY IOCS		FULL IOCS	
TSX	.RANDE,4	TSX	RANDEF,4
PZE	queue,,drha	PZE	queue,,drha
PZE	,,S	PZE	,,S
PZE	R,,N	PZE	R,,N

where:

queue
is the location of the first word in the request entry queue area reserved by the programmer.

drha
is the location of the first word in the disk/drum record holding area reserved by the programmer.

.RANRE Calling Sequence

For each alpha record, the programmer enters IOCS to obtain the necessary beta records. In the calling sequence for this entry, the programmer specifies the operation and gives the address of each beta record that corresponds to the alpha record. The following calling sequence causes a buffer to be reserved for each beta record being requested. Unless otherwise specified, the beta record is read into the reserved buffer.

LIBRARY IOCS			FULL IOCS	
TSX	.RANRE,4		TSX	RANREQ,4
pfx	alpha,t1,m		pfx	alpha,t1,m
pfx	erret		pfx	erret
pfx	a,t2,proces	These three words are supplied for each beta record.	pfx	a,t2,proces
pfx	phys addr		pfx	phys addr
verif	addr		verif	addr
...	.....		...	.....
...	.....		...	.....
...	.....		...	.....

where:

Word 1:

pfx = PZE
for full-random processing.

pfx = MZE
for force-sequential processing.

alpha
is the alpha record buffer address.

t1 $\neq$ 0
to retain control if .RANCA is set.

m
is the number of beta records requested in this calling sequence.

Word 2:

pfx = PZE
for no write check when writing.

pfx = MZE
for write check when writing.

erret
Random IOCS will transfer to this location when a beta record address fails to be verified (no record found). (See the section "Error Handling Procedures.")

Word 3:

sign bit
=0 if a read is requested.
=1 if the user intends to write a new record; the reservation of a buffer is necessary, but no read is needed.

bit 2
=0 for an independent beta record.
=1 for a mutually dependent beta record.

process
is the location of the user's processing routine for this beta record.

t2: bit 18
=0 for a binary beta record.
=1 for a BCD beta record.

bit 19
=0 for single record mode.
=1 for full track with addresses.

a
is the location of a word (possibly a System Unit Function Table entry) which contains in its address portion the location of the unit control block of the module addressed.

Word 4:

pfx = PZE
if the beta record physical address is in binary.

pfx = MZE
if the beta record physical address is in BCD (four characters).

phys addr
is the record physical address, right-justified.

Word 5:

verif addr
is the record verification address in BCD (six characters).

Analogous information to that given in words 3, 4, and 5 must be given for each beta record requested.

Although alpha is normally the alpha record buffer address, it is used to identify a given entry through .RANRE. Therefore, if the programmer finds it necessary to distinguish several entries to .RANRE for a given alpha record, alpha must identify the different entries for him. Alpha is not interpreted by Random IOCS. It is sent directly to the processing routine upon completion of the necessary read operation.

Flag Words

(Two flag words in Random IOCS are discussed in the following text. .RANFL and .RANCA are the names of these words in Library IOCS. The discussion applies equally to the flag words RANFLG and RANCAV in Full IOCS.)

Upon completion of a read, if the beta record just read is independent or completes a group of mutually dependent beta records, IOCS sets a flag in a location indicated by the word .RANFL, making it nonzero, and control is then returned through IOEX to the point at which the machine was trapped. The programmer's processing routine is not executed at trap time.

.RANFL indicates to the main program whether or not IOCS has a beta record, or records, ready for processing. This allows IOCS to return control to the main program after a .RANRE entry, without waiting for the completion of the read operation. As soon as the flag is set, the main program can detect it by testing .RANFL. .RANFL must be indirectly addressed. When the flag is set, the main program can re-enter .RANRE to have the processing routine executed. A regular entry through .RANRE may not be possible, as indicated later in text. In this case, the programmer can enter through .RANRE with m=0, using the short calling sequence. With either entry, the word pointed to by .RANFL is cleared.

An entry to .RANRE with m=0 is made using a short version of the .RANRE calling sequence. Only the transfer instruction and word 1 are used. No error return, no processing routine entry address, and no reference to beta records are expected. After such an entry has been made, .RANRE returns to the main program by TRA 2,4.

When Random IOCS lacks spaces in which to record the information provided in the .RANRE calling sequence, the information is left within the calling sequence. If this occurs, the location pointed to by .RANCA is made nonzero. The programmer should always test .RANCA before entering .RANRE to request another beta record.

If the programmer can perform no useful computation while .RANCA is nonzero, he can avoid the necessity of testing .RANCA by using a calling sequence in which

the tag of word 1 is nonzero. This causes Random IOCS to retain control until .RANCA becomes zero.

Calling Sequence to Processing Routine

When the main program has entered Random IOCS, and IOCS finds that either an independent record or a group of mutually dependent records has been read since the last call, it enters the programmer's processing routine through the following calling sequence:

```
TSX     proces,4
PZE     req,,alpha
PZE     diskbi,,i
...     ......
...     ......
PZE     diskbi,,i
MZE     diskbi,,i
```

where:

Word 1:

req
is the request identification.

alpha
is the alpha record buffer address.

Word 2 and succeeding words:

diskbi
is the address of the buffer holding the i^{th} beta record of the set.

i
is the order within the set of the beta record being delivered.

The prefix MZE on the last word indicates the end of the calling sequence.

If the programmer wishes to suppress the normal process of writing the beta records back on disk or drum, he should make nonzero the tag portion of the words (in the calling sequence to the processing routine) that correspond to the records which are not to be written. The programmer must return to Random IOCS by TRA 1,2. IOCS then releases the buffers of the beta records which are not to be returned to disk or drum storage and begins rewriting the others. Each buffer is released after the beta record which it contains has been written.

On entry to the processing routine, the calling sequence might be the following:

```
TSX     proces,4
PZE     req,,alpha
PZE     diskb3,,3
MZE     diskb5,,5
```

To prevent the writing of record 5, the processing routine could modify the calling sequence to the following form:

```
TSX     proces,4
PZE     req,,alpha
PZE     diskb3,,3
MZE     diskb5,4,5
```

.RANRP Calling Sequence

It is possible for the programmer to obtain a beta record which is not the one he needs. (This is described in the discussion of "Chaining Method" in the publication *IBM 1301 IOCS for 1410 and 7000 Series Data Processing Systems*, Form J28-8064-1.) The algorithm used to derive the physical and verification addresses of the beta records may yield the same results for two or more different inputs. Since only one such record can be written in the space designated, the other records are linked in a chain in which the record in the actual physical address location points to another record with the same derived physical address, and this points to another, until all of the records have been included. If the first record given is not the one required, the programmer should extract from the record the link address pointing to the next record in the chain and re-enter Random IOCS through the following calling sequence:

```
LIBRARY IOCS            FULL IOCS
TSX     .RANRP,4        TSX     RANRPL,4
PZE     req,,i          PZE     req,,i
PZE     a               PZE     a
pfx     phys addr       pfx     phys addr
verif   addr            verif   addr
```

where:

Word 1:

req
is the request identifier.

i
is the order within the set of the beta record being replaced.

Word 2:

a
is the location of a word (possibly a System Unit Function Table entry) which contains in its address portion the location of the unit control block of the module addressed.

Word 3:

pfx = PZE
if the physical address is in binary.

pfx = MZE
if the physical address is in BCD.

phys addr
is the beta record physical address, right-justified.

Word 4:

verif addr
is the beta record verification address in BCD (six characters).

Random IOCS replaces the old beta record disk or drum address with the new address. Later, when the new record is received, IOCS goes back to the processing routine, ignoring the earlier entry for that record (or group).

The .RANRP routine may be entered only by the programmer's processing and error routines. When the entry is made by the processing routine, .RANRP notes that a beta record is being replaced, changes the request to agree with the information in the calling sequence, and returns to the processing routine. Several beta records may be replaced before the processing routine returns control to Random IOCS. Since the records have not been processed, their buffers are not released.

When .RANRP is entered by the error routine (this is discussed in the section "Error Handling Procedures"), it modifies the request and then gives control either to the main program or to a processing routine, depending on the status of the various read operations in progress.

.RANCL Calling Sequence

The final entry to Random IOCS is made to the closing sequence, .RANCL, using the following calling sequence:

```
LIBRARY IOCS              FULL IOCS
TSX     .RANCL,4          TSX     RANCLS,4
```

The programmer enters the random close routine when he reaches an end of file on his input alpha file, when he wants to complete the processing of a group of alpha records, or when he wants to change the parameters of the process by an entry to .RANDE. IOCS retains control, continually scanning the queue for completed beta requests, until all pending requests have been processed. When all activity has ceased, IOCS returns to the main program by TRA 1,4. After this, the programmer can enter IOCS again at any time to process a new collection of alpha records. It is not necessary to repeat the entry to .RANDE unless different parameters are to be provided.

Error Handling Procedures

If no beta record is found, Random IOCS transfers to the programmer's error routine by using the following calling sequence:

```
TSX     erret,4
PZE     alpha
PZE     req,,i
Return  1
Return  2
```

The programmer can then take one of three courses of action:

1. He can replace the record address in the request, using an entry to .RANRP. No return to 3, 4 or to 4, 4 is necessary in this case.

2. He can reject the unobtained beta record, and ask Random IOCS to go to the processing routine with the rest of the request, if any. For this action, the error routine should use Return 1 (TRA 3,4).

3. He can nullify the complete alpha request. This is done by using Return 2 (TRA 4,4).

If an error other than the no-record-found type occurs, the following message will be printed out:

PERSISTENT ERROR – BETA (verif addr)

indicating the request which resulted in the failure.

If this error is a failure to write properly on disk or drum, the following message will also be printed:

DATA CHECK WHILE WRITING ON DISK

In any case, the entire contents of the request queue will be printed in the following format:

```
STATUS OF REQUESTS FOLLOWS
ALPHA              BETA              STATUS
(octal location)   (verif addr)      (status code)
```

Under STATUS, Random IOCS will print one of the following six code words:

1. RDWAIT – The request was just entered.
2. RDISSD – The beta record is in the process of being read.
3. PRWAIT – The beta record is in core storage, waiting to be processed.
4. PRISSD – The beta record is being processed.
5. WRWAIT – The beta record is waiting to be written out.
6. WRISSD – The beta record is in the process of being written out.

If there are no requests in the queue, the following message will be printed:

NO REQUESTS WAITING

If the main program requests beta records by way of .RANRE while the word pointed to by .RANCA is nonzero, the following message is printed:

BETA REQUESTS UNSERVICED

If the process routine returns to Random IOCS with a request identifier that IOCS does not recognize, the following message is printed:

INVALID REQ IDENTIFIER

All error messages will be written on the on-line printer (SYSPRT). If any error other than the no-beta-record-found type occurs, the job will be terminated with a core storage dump.

Sample Program Using Random IOCS

In the sample program shown in Figures 24 and 25, the third word of each alpha record is placed into the first word of the corresponding beta buffer that is not 616161616161.

If an error exit from .RANRE is taken, the alpha record is written out onto another file.

The first word of the alpha record is the desired disk (or drum) verification address. (The first four BCD characters are the track address.)

```
1       8       16
$EXECUTE        IBJOB
$IBJOB          GO,MINIMUM
$POOL           'ERRORFILE'
$IBMAP  DKNAM   150
AB55    FILE    INPUTFILE,UT1,INPUT,BLK=14
AB56    FILE    ERRORFILE,UT2,OUTPUT,BLK=14
SAM     SAVE
        TSX     .OPEN,4
        PZE     AB55
        TSX     .RANDE,4
        PZE     AQUEUE,,ADRHA       (GO TO
        PZE     ,,4                 RANDOM IOCS
        PZE     14,,4               PREPROCESSOR)
        CLA     SYSUNI
        ADM     AB10
        STA     AB03-1
        AXT     4,2
AB01    TSX     .READR,4
        PZE     AB55,,AB58          (ALPHA,,EOB)
        PZE     AB61,,AB60          (EOF,,ERR)
        IORTN   **,,14
        CAL     *-1                 (PICK UP ADDRESS OF ALPHA BUFFER)
        PAC     0,1                 (-L(BUFFER))
        STA     AB02
        STA     ATEST+4,2
        CLA     0,1                 (FIRST WORD OF ALPHA RECORD)
        SSM                         (BETA RECORD IN BCD)
        STO     AB03
        CLA     1,1                 (TTTTRR)
        STO     AB03+1
        TSX     .RANRE,4
AB02    PZE     **,,1               (ALPHA RECORD ADDRESS)
        PZE     AERRET
        PON     **,4,APROCS
AB03    MZE     **                  (PHYSICAL ADDRESS)
        PZE     **                  (VERIFY ADDRESS)
AB22    NZT*    -RANCA              (BETA BUFFER QUEUE FILLED?)
        TRA     *+5                 (NO)
        NZT*    -RANFL              (YES, ANY ACTIVITY TO BE PROCESSED?)
        TRA     *-1                 (NO, WAIT HERE)
        TSX     .RANRE,4            (YES, MAKE SHORT ENTRY
        PZE     0,7,0               TO RANDOM)
        AXT     4,2
        NZT     ATEST+4,2
        TRA     AB01
        TIX     *-2,2,1
        TRA     AB22+4
AB61    TSX     .CLOSE,4            (CLOSE ALPHA FILE)
        PTW     AB55
        TSX     .RANCL,4
        NZT     AB04
        RETURN  SAM
        TSX     .CLOSE,4
        PTW     AB56
        RETURN  SAM
*           ERROR EXIT FROM RANREQ
AERRET  SXA     AB41,4
        ZET     AB04
        TRA     AB05                (NOT FIRST ERROR)
        STL     AB04
        TSX     .OPEN,4
        PZE     AB56
        XEC     AB41
AB05    CLA     1,4
        STA     *+4
        STA     *+5                 (ADDRESS OF ALPHA)
        TSX     .WRITE,4
        PZE     AB56,,AB59          (WRITE ALPHA RECORD)
        IOCD    **,,14
        TSX     .RELES,4
        PZE     **,,AB55            (RELEASE ALPHA BUFFER)
        AXT     4,2
        CAL     ATEST+4,2
        LAS*    *-3
        TRA     **2
        TRA     **2
        TIX     *-3,2,1
        STZ     ATEST+4,2
AB41    AXT     **,4
        TRA     4,4                 (IGNORE REQUEST)
*               PROCESSING ROUTINE
*
*THE ALPHA RECORD NUMBER IS PLACED IN THE FIRST AVAILABLE WORD
*OF THE BETA RECORD
*
APROCS  SXA     AB29,2
        TXI     **+1,4,-1
        PXA     0,4
        PAC     0,2                 (SAVE LOCATION CONTAINING
        SXA     AB07,2              ALPHA BUFFER ADDRESS)
        AXT     0,1
        CLA     1,4                 (DISKBI,,I)
        STA     **+1
AB06    CLA     **,1                (ADDRESS OF BETA BUFFER)
        CAS     AB17
        TRA     **2
        TRA     AB07                (FOUND UNUSED DATA WORD)
        TXI     **+1,1,-1
        TXH     AB06,1,-14
        AXT     -2,1                (PLACE IN THIRD WORD FAIL OTHERWISE
AB07    CLA     **                  REQ,,ALPHA)
        PDC     0,2
        CLA     3,2                 (PICK UP THIRD WORD OF ALPHA RECORD)
        STO*    AB06
        CLA*    AB07
        ARS     18                  (ALPHA BUFFER ADDRESS TO 21-35)
        ANA     AMASK2              (ADDRESS OF BUFFER WE HAVE PROCESSED)
        AXT     4,2
        CAS     ATEST+4,2           (FIND CURRENT ALPHA BUFFER)
        TRA     **2
        TRA     **2
        TIX     *-3,2,1
        STA     **2
        TSX     .RELES,4            (RELEASE ALPHA BUFFER)
        PZE     **,,AB55
        STZ     ATEST+4,2
AB29    AXT     **,2
        TRA     1,2
*           CONSTANTS
AMASK2  OCT     77777
ADRHA   BSS     4*14+4+1
AQUEUE  BSS     4*7+1
AB04    PZE     **                  (ERROR ENTRY SWITCH)
ATEST   PZE     **
        PZE     **
        PZE     **
        PZE     **
AB10    PZE     SYSUT3
AB17    OCT     616161616161
AB58    TRA     SYSDMF              (EOB EXIT)
AB59    TRA     SYSDMP
AB60    TRA     SYSDMP              (ERROR EXIT)
        END
  (END-OF-FILE CARD)
$IBSYS
```

Figure 24. Library IOCS Random Example

```
1       8       16
$EXECUTE        IBSFAP
        *FAP
        COUNT   200
*   SAMPLE PROBLEM TO DEMONSTRATE 7090/94 RANDOM IOCS
        ABS
        SST
IOCS    EQU     SYSORG
DEFINE  EQU     IOCS+4          (LOCATIONS
ATTACH  EQU     IOCS+8          OF IOCS
CLOSE   EQU     IOCS+10         ROUTINES)
OPEN    EQU     IOCS+12
WRITE   EQU     IOCS+16
READR   EQU     IOCS+32
RELESE  EQU     IOCS+34
RANDEF  EQU     IOCS+76
RANREQ  EQU     IOCS+77         (LOCATION OF
RANCLS  EQU     IOCS+78         RANDOM IOCS
RANFLG  EQU     IOCS+80         ROUTINES)
RANCAV  EQU     IOCS+81
START   BOOL    15000
        ORG     START
*
AB55    BSS     12              (FILE CONTROL BLOCK ORIGIN)
AB56    BSS     12
AB00    TSX     DEFINE,4
        PZE     AB51            (DEFINE, ATTACH AND OPEN
        PZE     4,,4            THE INPUT FILE
        TSX     ATTACH,4        CONTAINING
        PZE     AB51            ALPHA RECORDS)
        PZE     AB53,,1
        TSX     OPEN,4
        PZE     AB55
        TSX     RANDEF,4
        PZE     AQUEUE,,ADRHA   (GO TO
        PZE     ,,4             RANDOM IOCS
        PZE     14,,4           PREPROCESSOR)
        AXT     4,2
AB01    TSX     READR,4
        PZE     AB55,,AB58      (ALPHA,,EOB)
        PZE     AB61,,AB60      (EOF,,ERR)
        IORTN   **,,14
        CAL     *-1             (PICK UP ADDRESS OF ALPHA BUFFER)
        PAC     0,1             (-L(BUFFER))
        STA     AB02
        STA     ATEST+4,2
        CLA     0,1             (FIRST WORD OF ALPHA RECORD)
        SSM                     (BETA RECORD IN BCD)
        STO     AB03
        CLA     1,1             (TTTTRR)
        STO     AB03+1
        TSX     RANREQ,4
AB02    PZE     **,,1           (ALPHA RECORD ADDRESS)
        PZE     AERRET
        PON     SYSUT3,4,APROCS
AB03    MZE     **              (PHYSICAL ADDRESS)
        PZE     **              (VERIFY ADDRESS)
AB22    NZT*    RANCAV          (BETA BUFFER QUEUES FILLED?)
        TRA     **5             (NO)
        NZT*    RANFLG          (YES, ANY ACTIVITY TO BE PROCESSED?)
        TRA     *-1             (NO, WAIT HERE)
        TSX     RANREQ,4        (YES, MAKE SHORT ENTRY
        PZE     0,7,0           TO RANDOM)
        AXT     4,2
        NZT     ATEST+4,2
        TRA     AB01
        TIX     *-2,2,1
        TRA     AB22+4
AB61    TSX     CLOSE,4         (CLOSE ALPHA FILE)
        PTW     AB55
        TSX     RANCLS,4
        NZT     AB04
        TRA     IOCS
        TSX     CLOSE,4
        PTW     AB56
        TRA     IOCS            (END OF JOB)
*           ERROR EXIT FROM RANREQ
AERRET  SXA     AB41,4
        ZET     AB04
        TRA     AB05            (NOT FIRST ERROR)
        STL     AB04
        TSX     DEFINE,4
        PZE     AB08
        PZE     3,,14           (INITIALIZE OUTPUT)
        TSX     ATTACH,4        (ERROR FILE)
        PZE     AB08
        PZE     AB09,,1
        TSX     OPEN,4
        PZE     AB56
        XEC     AB41
AB05    CLA     1,4
        STA     **+4
        STA     **+5            (ADDRESS OF ALPHA)
        TSX     WRITE,4
        PZE     AB56,,AB59      (WRITE ALPHA)
        IOCD    **,,14
        TSX     RELESE,4        (RELEASE ALPHA BUFFER)
        PZE     **,,AB55
        AXT     4,2
        CAL     ATEST+4,2
        LAS*    *-3
        TRA     **2
        TRA     **2
        TIX     *-3,2,1
        STZ     ATEST+4,2
AB41    AXT     **,4
        TRA     4,4             (IGNORE REQUEST)
*               PROCESSING ROUTINE
*
*THE ALPHA RECORD NUMBER IS PLACED IN THE FIRST AVAILABLE WORD
*OF THE BETA RECORD
*
APROCS  SXA     AB29,2
        TXI     **1,4,-1
        PXA     0,4
        PAC     0,2             (SAVE LOCATION CONTAINING
        SXA     AB07,2          ALPHA BUFFER ADDRESS)
        AXT     0,1
        CLA     1,4             (DISKBI,,I)
        STA     **1
AB06    CLA     **,1            (ADDRESS OF BETA BUFFER)
        CAS     AB17
        TRA     **2
        TRA     AB07            (FOUND UNUSED DATA WORD)
        TXI     **1,1,-1
        TXH     AB06,1,-14
        AXT     -2,1            (PLACE IN THIRD WORD IF FAIL OTHERWISE
AB07    CLA     **              REQ,,ALPHA)
        PDC     0,2
        CLA     3,2             (PICK UP THIRD WORD OF ALPHA RECORD)
        STO*    AB06
        CLA*    AB07
        ARS     18              (ALPHA BUFFER ADDRESS TO 21-35)
        ANA     AMASK2          (ADDRESS OF BUFFERS WE HAVE PROCESSED)
        AXT     4,2
        CAS     ATEST+4,2       (FIND CURRENT ALPHA BUFFER)
        TRA     **2
        TRA     **2
        TIX     *-3,2,1
        STA     **2
        TSX     RELESE,4        (RELEASE ALPHA BUFFER)
        PZE     **,,AB55
        STZ     ATEST+4,2
AB29    AXT     **,2
*               CONSTANTS
AMASK2  OCT     77777
ADRHA   BSS     4*14+4+1
AQUEUE  BSS     4*7+1
AB04    PZE     **              (ERROR ENTRY SWITCH)
ATEST   PZE     **
        PZE     **
        PZE     **
        PZE     **
AB17    OCT     616161616161
AB08    BSS     3*14+3*2+2      (ERROR FILE POOL)
AB51    BSS     4*14+4*2+2      (INPUT POOL)
AB09    SVN     1,,1
        PZE     AB56
AB53    SVN     1,,4
        PZE     AB55            (INPUT TAPE)
AB58    TRA     SYSDMP          (EOB EXIT)
AB59    TRA     SYSDMP
AB60    TRA     SYSDMP          (ERROR EXIT)
        END     AB00
$IBSYS
```

Figure 25. Full IOCS Random Example

This section contains detailed descriptions of the options and formats of control cards used with Library IOCS.

The level of IOCS desired by the Library IOCS programmer can be specified in the $IBJOB card. If the programmer fails to specify a level, or fails to specify a high enough level, the IBJOB loader loads the level of IOCS needed to service the routines used in the program.

The other control cards and pseudo-operations described in this section are used for file and label specifications and for grouping files within buffer pools. The following notations are used to describe the fields of control cards and pseudo-operations:

1. Information enclosed by brackets [ ] may be included or omitted, by choice of the programmer. If the programmer does not indicate a choice, the IBJOB Processor Monitor or the Macro Assembly Program assumes the underlined standard option.
2. Information enclosed by braces { } within brackets gives the field options from which the programmer can choose.
3. Upper-case (all capitals) terms must be present in that form, if used.
4. Lower-case terms represent typical quantities or expressions whose values must be supplied by the programmer.
5. Options may be specified in the variable field in any sequence, unless otherwise stated.
6. Commas are used to separate options when options are present. If options are absent, it is not necessary to indicate their absence by using a string of commas, unless otherwise stated.

Format of $IBJOB Card

The $IBJOB card must be the first control card read by the IBJOB Processor Monitor for a given application. The options that can be specified in this card describe the manner in which an application is to be processed.

Of special interest to the Library IOCS programmer is the IOCS options field of the $IBJOB card. In this field, the programmer can specify the IOCS configuration (IOEX, FORTRAN, Minimum, Basic, or Labels) to be used with his program.

The format of the $IBJOB card is as follows:

1	16
$IBJOB	options, ...,...,...

Columns 1-6 must contain the identifying name of the control card: $IBJOB

The options in the variable field (columns 16-72) are as follows:

Execution Options:

$$\left[\left\{\begin{matrix}\underline{\text{GO}}\\ \text{NOGO}\end{matrix}\right\}\right]$$

GO specifies that the object program is to be executed after it is loaded.

NOGO specifies that the object program is not to be executed, even if it is loaded. If NOGO is specified, the object program is loaded only when LOGIC, DLOGIC, or MAP is specified in the $IBJOB card.

Logic Options:

$$\left[,\left\{\begin{matrix}\underline{\text{NOLOGIC}}\\ \text{DLOGIC}\\ \text{LOGIC}\end{matrix}\right\}\right]$$

LOGIC specifies that a cross-reference table of the program sections and the system subroutines required for execution be written on the System Output Unit. The origin and length of each program section and subroutine and the buffer assignments are also given.

DLOGIC specifies that a cross-reference table of the program sections and the origin and length of each program section is to be written on the System Output Unit. The system subroutines and buffer assignments are not given.

NOLOGIC specifies that a cross-reference table is not wanted.

MAP Options:

$$\left[,\left\{\begin{matrix}\underline{\text{NOMAP}}\\ \text{MAP}\end{matrix}\right\}\right]$$

MAP specifies that a core storage map, giving the origin and amount of storage used by the IBSYS Operating System, the object program, and the input/output buffers, is written on the System Output Unit. The file list and buffer pool organization are also given.

NOMAP specifies that a core storage map is not wanted.

File List Options:

$$\left[,\left\{\begin{matrix}\underline{\text{NOFILES}}\\ \text{FILES}\end{matrix}\right\}\right]$$

FILES specifies that a list of input/output unit assignments and mounting instructions to the operator are to

be printed on-line and written off-line on the System Output Unit.

NOFILES specifies that the list is to be printed on-line, but is not to be written on the System Output Unit.

IOCS Options:

[, (IOEX / FIOCS / MINIMUM / BASIC / LABELS)]

The IOCS options are as follows:

IOEX specifies that the Input/Output Executor is to be used for trap supervision. The only IOCS routine available is the .MWR routine for on-line printing.

The FIOCS specification should be used only by the FORTRAN IV programmer. It specifies that the reduced form of Minimum IOCS is to be loaded for use by the FORTRAN IV object program. If the FORTRAN IV programmer does not specify FIOCS, the Minimum level of IOCS will be loaded.

MINIMUM specifies that the Minimum level of IOCS is to be loaded with the object program. Internal files cannot be used at the Minimum level. This level contains the .ATTAC and .DEFIN routines utilized by calling sequences generated by the Loader. This level also contains the following IOCS routines:

.OPEN
.CLOSE
.READ
.WRITE
.BSR

If IOCS has been assembled for disk and/or drum storage capability, this level also contains the following two routines:

.READR
.RELES

BASIC specifies that the Basic level of IOCS is to be loaded with the object program. Besides the routines in Minimum IOCS, this level contains the following routines:

.BSF
.CKPT
.COPY
.JOIN (not normally used)
.REW
.STASH
.WEF

LABELS specifies that the Labels level of IOCS is to be loaded with the object program. This level contains all of the routines in the Minimum and Basic levels plus all label checking and label preparation routines.

The above specifications may be ignored by the Loader at load time. If the object program requires a higher level of IOCS than is specified, the higher level will be loaded.

Random IOCS is loaded with the object program if the program contains a reference to a Random IOCS routine.

Loading Options:

[, (SOURCE / NOSOURCE)]

SOURCE specifies that this application involves at least one compilation or assembly.

NOSOURCE specifies that this application involves only relocatable binary object program decks which will be loaded from System Input Unit 1. If neither GO, MAP, nor LOGIC has been specified in the $IBJOB card, the decks will not be loaded.

Overlay Options:

[, (NOFLOW / FLOW)]

FLOW specifies that execution of the object program is to be terminated when a link contains a call that causes it to be overlaid or when reference is made to a control section that is not in core storage at the time.

NOFLOW specifies that execution is to be allowed when the conditions mentioned in FLOW occur.

Formats of FILE Pseudo-Operation and $FILE Card

FILE Pseudo-Operation

The FILE pseudo-operation is written in the MAP language source program. It provides detailed specifications about a file for use by IOCS.

The Macro Assembly Program converts the FILE pseudo-operation into a $FILE card, which is later included with the object deck when that deck becomes input for the IBJOB Loader.

The fields of the FILE pseudo-operation are as follows:

The name field (positions 1-6) must contain the internal name of the file. This is the name the programmer has used to refer to the file within his source program. This name becomes the symbolic location of the first word of the file control block.

The operation field (positions 8-11) must contain the four letters FILE.

The variable field (positions 16-72) contains one or more options that describe the file.

If the variable field is too long for a single line of the program coding sheet, it can be continued on the next line by using the ETC pseudo-operation in the operation field of the next line.

The options are as follows:

External Name of the File: The external name is an alphameric literal of up to 18 BCD characters excluding blank, comma, slash, quotation mark, and asterisk. The external name is the name used by other programs

to refer to this file. The external name, if present, must be the first subfield in the variable field. If an external file name is not present, the variable field must begin with a comma in column 16. When no external name is entered, the six-character name field (left-justified with trailing blanks) is regarded as the external name. If the name field in positions 1-6 is also absent, MAP issues a level 4 error message.

The external name is also the name that is printed out whenever IOCS prints a message concerning this file.

The order of subsequent entries in the variable field is arbitrary.

Unit Assignment Options: Two symbolic units may be specified for each file. For tape units, the primary unit is the first one to be used and the secondary unit is to be used as a reel-switching alternate. Unit assignment specifications are described in the section "Unit Assignment Under the IBJOB Processor."

File Mounting Options:

[,{MOUNT / READY / DEFER}] and/or [,{MOUNTi / READYi / DEFERi}]

These options govern the on-line message to the operator indicating the impending use of an input/output unit. The first form of the option (MOUNT, READY, DEFER) applies to both the primary unit and secondary unit assigned to a file. The second form applies to the primary unit when i=1 and to the secondary unit when i=2.

Note that two standard options are indicated. One is for units assigned to system unit functions (READY), and the other is for nonsystem units (MOUNT).

The effects of the file mounting options are as follows:

MOUNT	A message is printed before execution, and a stop occurs for the required operator action. This is the standard option for nonsystem units.
READY	A message is printed before execution, but no stop occurs. System units are normally given the READY option if a mounting option is not specified.
DEFER	A message and operator stop are deferred until the file is opened.

If both a primary unit and secondary unit are specified for a file, and only one file mounting option of the first form is entered, this option applies to both units. For example,

```
MOUNT
```

will cause the MOUNT action for both units.

If the i form of the option is used, it will override any general option specified for the unit i. For example,

```
MOUNT,DEFER2
```

will cause the MOUNT action for the primary unit and the DEFER action for the secondary unit.

Operator File List Options:

[,{LIST / NOLIST}]

LIST	This file will appear in the operator's mounting instructions.
NOLIST	No message will be printed, unless the DEFER option has been specified.

File Usage Options:

[,{INPUT / OUTPUT / INOUT / CHECKPOINT (or CKPT)}]

INPUT	The file is an input file.
OUTPUT	The file is an output file.
INOUT	The file may be either an input or an output file. Initially, the file is set up as an input file. The programmer is responsible for altering the appropriate bits in the file control block before he uses the file as an output file.
CHECK-POINT (or CKPT)	The file is a checkpoint file.

Block-Size Option:

```
[,BLOCK=xxxx (or BLK=xxxx)]
```

xxxx is an integer (0000-9999) that specifies the block size for this file. The block size may be omitted from the FILE pseudo-operation if the file is included in a pool or group where the block size can be determined. If block sequence and/or check sum options are specified, the block size number must be increased by 1 to allow for the block sequence/check sum word. If the block size option is omitted, the Macro Assembly Program assumes a block size of 14 for BCD or MXBCD files and a block size of 256 for BIN or MXBIN files.

Activity Option:

```
[,ACT=xx]
```

xx is an integer (00-99) that specifies the relative activity of this file with respect to other files. If this field is omitted, the Loader assumes an activity value of 1 for the file. The activity value is used in determining the number of input/output buffers assigned to each buffer pool in the object program. The highest activity value is 99.

Reel-Switching Options for Unlabeled Files:

[,{ONEREEL / MULTIREEL (or REELS)}]

ONEREEL	Reel switching should not occur.
MULTIREEL (or REELS)	Reel switching will occur. Every output file will switch reels if an end-of-tape condition occurs.

Reel-Switching Options for Labeled Files:

[,{NOSEARCH / SEARCH}]

NOSEARCH — If an incorrect label is detected when opening an input file, IOCS will stop for operator action.

SEARCH — If an incorrect label is detected, IOCS enters a multireel searching procedure for the file with the desired label.

File Density Options:

[, HIGH / LOW / 200 / 556 / 800]

This option specifies the density at which the file is to be read or written on tape. The options are as follows:

HIGH — The tape density switch is assumed to be set so that the execution of an SDH will result in the use of the correct density.

LOW — The tape density switch is assumed to be set so that the execution of an SDL will result in the use of the correct density.

200 — The file recording density is 200 cpi.

556 — The file recording density is 556 cpi.

800 — The file recording density is 800 cpi.

If a system unit is assigned to this file, the system-set density supersedes the density set by these options.

Mode Options:

[, BCD / BIN / MXBCD / MXBIN]

BCD — The file is in BCD mode.

BIN — The file is in binary mode.

MXBCD — The file is in mixed mode, and the first record is BCD.

MXBIN — The file is in mixed mode, and the first record is binary.

The MXBCD and MXBIN options may not be specified for a file unless one of the look-ahead words, described in Appendix F, is attached to the end of each physical record of the file. Since these words cannot be properly placed by programs written in the FORTRAN IV or COBOL languages, the MXBCD and MXBIN options can be used with compiled programs only when output is handled by subroutines written in the MAP language.

Label Density Options:

[, SLABEL / HILABEL / LOLABEL / FLABEL]

SLABEL — All header label operations are performed in the installation standard label density specification.

HILABEL — All header label operations are performed in high density.

LOLABEL — All header label operations are performed in low density.

FLABEL — All header label operations are performed in the same density as that of the file.

If neither HILABEL, LOLABEL, nor FLABEL is specified, the standard label density specification, defined by the assembly parameter SLABEL, is used to process labels. As distributed, the standard specification is high density. An installation can change the standard specification by changing the assembly parameter SLABEL, which is in the Loader.

Trailer label operations are always performed in the same density as that of the file.

NOTE: Only the use of a LABEL pseudo-operation or SLABEL card denotes a labeled file, whether one of the label density options is specified or not.

Block Sequence Options:

[, NOSEQ / SEQUENCE (or SEQ)]

NOSEQ — Block sequence word is neither checked if reading, nor formed and written if writing.

SEQUENCE (or SEQ) — Block sequence word is checked if reading, or formed and written if writing.

Check Sum Options:

[, NOCKSUM / CKSUM]

NOCKSUM — The check sum is neither checked if reading, nor formed and written if writing.

CKSUM — The check sum is checked if reading, or formed and written if writing.

A check sum option may be specified only if the block sequence option has been specified.

Checkpoint Options:

[, NOCKPTS / CKPTS]

NOCKPTS — No checkpoints are initiated by this file.

CKPTS — Checkpoints are initiated by this file.

Checkpoint Location Option:

[, AFTERLABEL]

Checkpoints are written following the label on this file when a reel switch occurs. If CKPTS is specified and this field is omitted, the checkpoints are written on the checkpoint file when a reel switch occurs.

File Close Options:

[, SCRATCH / PRINT / PUNCH / HOLD]

SCRATCH — The file is rewound upon termination of the application.

PRINT — The file is to be printed. It is rewound and unloaded upon termination of the application. PRINT will appear in the removal message that is printed on-line at the end of execution.

PUNCH — The file is to be punched. It is rewound and unloaded upon termination of the application. PUNCH will appear in the removal message that is printed on-line at the end of execution.

HOLD — The file is to be saved. It is rewound and unloaded upon termination of the application. HOLD will appear in the removal message that is printed on-line at the end of execution.

If the unit assigned is the System Input Unit, the System Output Unit, or the System Peripheral Punch, there is no rewind, nor is a message printed.

Starting Cylinder Number Option:

[,CYLINDER=xxx (or CYL=xxx)]

xxx is the number (000-249 for disk, 000-009 for drum)

of the starting cylinder of this file. The equal sign is required. The programmer must specify the starting cylinder number when disk or drum storage is specified for the file. This does not apply if a system unit function is specified for a file that is on disk or drum storage.

Cylinder Count Option:

[,CYLCOUNT=xxx(or CYLCT=xxx)]

xxx is the number (000-250 for disk, 000-010 for drum) of consecutive cylinders to be used by this file. The equal sign is required. The programmer must specify the cylinder count when disk or drum storage is specified for the file. This does not apply if a system unit function is specified for a file that is on disk or drum storage.

Disk and Drum Write-Checking Option:

[, WRITECK]

Write checking is performed after each disk or drum write sequence for this file.

Hypertape Reel-Switching Options:

These options may be used in conjunction with the Hypertape option (HYPER) where reel switching is likely to occur. For reel switching to occur, the programmer specifies whether or not the Hypertape is to be rewound and/or file-protected. If any of these options are used, but HYPER is not specified, a warning message is issued.

HRFP	Designates Hypertape rewind and file protection.
HRNFP	Designates Hypertape rewind, but no file protection.
HNRFP	Designates that the Hypertape is not to be rewound, but is to be file protected.
HNRNFP	Designates that the Hypertape is not to be rewound or file protected.

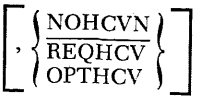
The five subfields that follow are used to provide information for cross-checking by the IBJOB Loader. These five subfields, which are not placed in the $FILE card, are as follows:

Hypertape Option:

[,HYPER]

HYPER must be specified if a program requires Hypertape for a particular file. If reel switching may occur, the Hypertape reel-switching options may be used in conjunction with HYPER. However, use of the Hypertape reel-switching options without specifying HYPER results in a warning message.

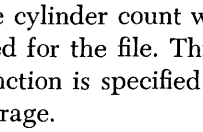
Conversion Options:



NOHCVN	An alphameric-to-BCD conversion routine is not necessary. The file may not be assigned to card equipment.
REQHCV	An alphameric-to-BCD conversion routine is required. The file must be assigned to card equipment.
OPTHCV	An alphameric-to-BCD conversion routine is optional. The file may be assigned to any input/output device.

Regardless of these options, the programmer must provide any required conversion routines.

Block-Size Check Options:



MULTI=xxxx	The block size is a multiple of xxxx.
MIN=xxxx	The minimum block size is xxxx.

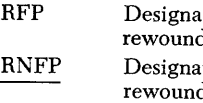
Only one of the above options may appear. The quantity specified is used by the Loader to check the block size indicated by the BLOCK option. If neither option appears, the block size is assumed to be exactly that specified by the BLOCK option. If the BLOCK option is also omitted, the assembly program assumes a block size of 14 for BCD or MXBCD files and 256 for BIN or MXBIN files.

Nonstandard Label Routine Option:

[,NSLBL=symbol]

The symbol is the external name of a nonstandard label routine. If the nonstandard label routine is part of the program segment being assembled, this routine must be made a control section with the symbol used as its external name. If the nonstandard label routine is not part of this program segment, the symbol must be a virtual symbol.

Pool-Attachment Option:



POOL	This file is to be attached to a pool.
NOPOOL	A file control block is to be generated, but this file is not to be attached to a pool.

If CHECKPOINT (or CKPT) is specified for the file-usage option, NOPOOL is automatically assumed.

If the NOPOOL option is specified for a reserve or internal file, it is assumed that the IOCS initialization sequences of .DEFIN and .ATTAC will be executed in the object program for this file.

$FILE Card

The specifications in this card override the file description that appears in the source program or relocatable object program to be loaded.

The format of the card is as follows:

1	16
$FILE	'filename', [options,...]

The format of the $FILE card differs from the FILE pseudo-operation in that card columns 1 through 5 contain $FILE and the file name (alphameric name of 18 or fewer characters) begins in column 16 and must be enclosed in quotation marks (4-8 punch).

The options in the variable field are the same as those available in the FILE pseudo-operation except for the five exceptions cited previously. The file name must be the first entry in the variable field. The other options may be entered in the variable field in any order. The options are separated from the file name and from each other by commas.

Formats of LABEL Pseudo-Operation and $LABEL Card

LABEL Pseudo-Operation

The LABEL pseudo-operation is used in conjunction with its associated FILE pseudo-operation when the file is labeled. The LABEL pseudo-operation provides label information for the file. At assembly time, the LABEL pseudo-operation causes the Macro Assembly Program to generate a corresponding $LABEL card, which later becomes input to the IBJOB Loader. Whereas the FILE pseudo-operation describes the file characteristics and supplies information for the File Dictionary, the LABEL pseudo-operation gives only labeling information to IOCS and supplies no information for the dictionary. The fields of the LABEL pseudo-operation are as follows:

1. The name field (positions 1-6) contains the internal name of file (same as in FILE pseudo-operation), or blanks.

2. The operation field (positions 8-12) must contain the five letters LABEL.

3. The varible field (positions 16-72) contains options that must be entered in the following order (left to right): external name, file serial number (or HA2), reel sequence number, retention date or days, and file identification name. The variable field must contain the equivalent of five subfields. The external name can be omitted by using a comma in position 16. The omission of the other subfields, except the last one, should be indicated by using adjacent commas (,,). The variable field of the LABEL pseudo-operation must not extend beyond position 72 (i.e., no ETC pseudo-operations can be used continue the LABEL variable field).

The options in the variable field are as follows:

External Name of File (Same as in FILE Pseudo-Operation): An alphameric literal of up to 18 BCD characters excluding blank, comma, slash, quotation mark, and asterisk. If this field is omitted (the variable field begins with a comma), the symbol in the name field is inserted as the external name. If the name field is also blank, 000000 is inserted as the external name. The contents of this subfield become the 'file name' in the $LABEL card.

File Serial Number: An alphameric field of five or fewer characters. Standard input labels will be checked against this serial number if it is present. Standard output labels for this file will contain this serial number only if the reel sequence option specifies a reel number greater than 1. Output serial numbers are normally taken from the label already present on the tape on which the first reel of the file is written.

If a disk or drum label is desired, this field must contain two BCD characters that specify the home address-2 (HA2).

Reel Sequence Number: A numeric field of four or fewer digits.

This option specifies the reel sequence number of the first reel of a magnetic tape file. When the field is omitted, the sequence number is assumed to be 1 for an output file or 0 for an input file and this field is not checked. The reel sequence number is adjusted at object time to reflect reel switching, and it is checked in standard output labels.

If a disk or drum label is desired, this field must be omitted by using two adjacent commas.

Retention Days or Date:

Days — This is a numeric field of four or fewer characters. It specifies the number of days a tape is to be retained from the date it is written. An attempt to write a labeled file on this tape before the end of the retention period results in an error message. If the field is omitted, a value of zero is assumed.

Date — A date can be entered in this field. The format is Y/D where Y is a one-digit or two-digit number indicating the year, and D is a number of three or fewer digits indicating the day of the year. The slash is used to separate the two entries. This entry is not checked and can be used merely to provide additional information in the label.

File Identification Name: This subfield must follow the last comma in the variable field. This name is checked with the name in the input label and a mismatch results in an IOCS warning message. If this field is omitted for an input file, the file identification name is not checked. For an output file, this file identification name is placed in the output label. The file identification name may contain embedded blanks but not commas.

The Macro Assembly Program checks the variable field of the LABEL pseudo-operation. If there are more than five subfields, the variable field is truncated and only the first five subfields are used. In this case, a warning message is printed. If there are fewer than five subfields, an appropriate number of commas is supplied so that the $LABEL card will have the subfields, and a warning message is printed.

Each subfield is then checked for length. Subfields that are longer than the maximum previously specified are truncated to the maximum number of characters allowed for each, and a format error message is printed. Numeric subfields are also checked for validity and the presence of any nonnumeric characters in one of these subfields will cause the printing of a format error message.

The assembler inserts quotation marks around the external file name and transfers the five subfields to the $LABEL card. (The deck name for the $LABEL card is taken from the $IBMAP card.)

$LABEL Card

The $LABEL card is generated automatically by the Macro Assembly Program from the LABEL pseudo-operation in the source program. The subfields in the variable field of the card (columns 16-72) correspond exactly to the subfields in the LABEL pseudo-operation, except that card columns 1-6 contain the term $LABEL and the file name beginning in column 16 is in quotation marks. The 'file name' may be either the internal or external name of the file.

The format of the $LABEL card is as follows:

```
1                    16
$LABEL               'filename', [ { serial } ] , [reel],
                                 [ { HA2    } ]
                                 [ { days } ] , [fileidenname]
                                 [ { date } ]
```

Formats of $POOL and $GROUP Cards

$POOL Card

This card designates which files are to share common buffer areas and can be used to assign files with different block sizes to the same buffer pool. The card may be extended by the use of $ETC cards. It should be noted that the deck name is optional and may be omitted. If present, it does not qualify any file names.

The format of the $POOL card is as follows:

```
1                    16
$POOL                BLOCK=xxxx, BUFCT=xxx,
                     or BLK
                                    'filename1',...
```

The options in the variable field are as follows:

Block Size Option:

```
16
BLOCK or BLK=xxxx
```

xxxx is a number (0000-9999) that specifies the block size for this pool. The field may be omitted. If the field is omitted, the pool block size will be the same as the largest block size of a file in the pool.

Buffer Count Option:

```
, BUFCT=xxx
```

xxx is a number (001-999) that specifies the number of buffers to be assigned to the pool. It must be equal to, or larger than, the open count of the pool. If this field is omitted, the IBJOB Loader attempts to assign at least two buffers to each file.

File Names:

```
, 'filename1',...
```

The remaining data required on the control card are the names of the files which are to be included in the pool. Each file name is an alphameric literal of up to 18 characters and is enclosed in quotation marks. Deck name qualification is meaningless, since the IBJOB Loader assigns only one file block for each unique file name in the entire application.

If the name of a file appears both on a $POOL card and in a FILE pseudo-operation with the NOPOOL option, an error message is issued and the NOPOOL option is ignored.

$GROUP Card

This card is used to allocate buffer areas and to specify how the buffers are to be shared by the various files. All files named in a $GROUP card are formed into a Reserve Group. If group specifications are not made, the IBJOB Loader attempts to assign at least two buffers to each file. It should be noted that the deck name is optional and may be omitted, if desired. If it is present, it does not qualify any file names. $ETC cards may be used.

The format of the $GROUP card is as follows:

```
1                    16
$GROUP               OPNCT=xx, BUFCT=xxx,
                                    'filename1',...
```

The options in the variable field are as follows:

Open Count Option:

```
16
OPNCT=xxx
```

xx is a number (01-99) which specifies the number of files within the group that are to be open concurrently during the execution of the program. This count determines the minimum buffer count necessary for processing the group of files. If this field is omitted, the open count is assumed to be equal to the number of files in the group.

Buffer Count Options:

```
, BUFCT=xxx
```

xxx is a number (001-999) which specifies the number of buffers to be assigned to this group. It must be equal to, or larger than, the open count of the group. If this field is omitted, the IBJOB Loader attempts to assign at least two buffers to each file.

File Names:

```
, 'filename1',...
```

The remaining items of data required on the control card are the names of the files which are to be included in the group. Each file name is an alphameric literal

of up to 18 characters and is enclosed in quotation marks. Deck name qualification is meaningless, since the IBJOB Loader assigns only one file block for each unique file name in the entire application.

If the name of a file appears both on a $GROUP card and in a FILE pseudo-operation with the NOPOOL option, an error message is issued and the NOPOOL option is ignored.

Unit Assignment Under the IBJOB Processor

The following text indicates the manner in which primary and secondary units are specified for files in Library IOCS. These specifications are entered in the unit assignment option of the FILE pseudo-operation or $FILE card.

The following notation is used in later explanations to indicate the formats in which actual symbols are entered in the unit assignment options:

X	denotes a real channel (A through H).
P	denotes a symbolic (unspecified physical) channel (S through Z).
I	denotes a symbolic (specially-flagged) channel for intersystem use (J through Q).

M	denotes the model number of an IBM 729 Magnetic Tape Unit (II, IV, V, or VI).
D	designates IBM 1301 Disk Storage.
N	designates IBM 7320 Drum Storage.
H	designates an IBM 7340 Hypertape Drive.
k	denotes a unit number (0 through 9).
a	denotes an access mechanism number (0).
m	denotes the module number (0 through 9).
s	denotes the data channel switch (also called interface) (0 or 1).

Unit Assignment Specifications

Unit specification may be made in any of the following formats:

DESIGNATION	EXPLANATION
blank	Any available unit is to be assigned to the file.
M	Any available IBM 729 Magnetic Tape Unit of this model is to be assigned to the file.
X	Any available unit on this real channel is to be assigned to the file.
P	All files in the job having this symbolic channel designation are to be assigned to the same channel.
X(k)	The *k*th available unit on the specified real channel is to be assigned to the file. The parentheses are required for IBJOB applications.
PM	Any available unit of the model indicated by *M* on the symbolic channel specified by *P* is to be assigned to the file.
P(k)M	An available unit on the symbolic channel indicated by *P*, having the model number indicated by *M*, is to be assigned to the file. In this case, *k* indicates the order of preference in assignment to the channel. If the number of available units on the channel is less than the total requested for the channel, those files with lower numbers are to be assigned to the same channel. The parentheses are required in IBJOB Processor applications.
XDam/s	The disk storage unit on channel *X*, with access mechanism number *a*, module number *m*, and data channel switch *s*, is to be assigned to the file.
XNam/s	The drum storage unit on channel *X*, with access mechanism number *a* (0), module *m* (0, 2, 4, 6, 8), and data channel switch *s* (0, 1), is to be assigned to the file.
XHk/s	The 7340 Hypertape Drive that is unit number *k* on channel *X* with data channel switch *s* is to be assigned to the file.
NONE	This entry can be used in IBJOB Processor applications to indicate that a file is not to have a unit assigned to it. The entry is made in the primary unit field. No secondary unit specification should be made. When the NONE entry is made, a file control block is generated, but no unit is assigned. The first word of the file control block, which normally contains the locations of unit control blocks, is set to zero. This may be tested by the object program.
INT	The characters INT should be entered in the primary unit field to specifiy that a file is an internal file.

System units may be assigned as follows:

IN,IN1,IN2	The file is to be read from the current System Input Unit.
OU,OU1,OU2	The file is to be written on the current System Output Unit.
PP,PP1,PP2	The current System Peripheral Punch Unit is to be used for a punch output file.
UTk	The system utility unit specified by *k* is to be used for the file. *k* can be 1, 2, 3, or 4.
CKk	The System Checkpoint Unit specified by *k* is to be used by the file. *k* can be 1 or 2. When using the distributed version of IBSYS, these functions must be attached before they can be used.

Card equipment is assigned for a file as follows:

RDX	Card reader on channel *X*.
PRX	Printer on channel *X*.
PUX	Card punch on channel *X*.

The OPTHCV or REQHCV option in the FILE pseudo-operation must be specified if a card reader is requested. The REQHCV option can only be used for a card reader.

As a special option, an asterisk (*) in the secondary unit field indicates that the secondary unit of a file is to be any unit on the same channel and of the same model as the primary unit. This unit, if available, is assigned after all other unit assignments have been made. At load time, if units of a different model are available, one of them will be assigned to the file. If no units are available on the channel, the secondary unit is the same as the primary unit.

Relative unit specification is not the same as the physical setting of a unit. The designation of A(3), for example, will result in the assignment of the third available unit on channel A, and not necessarily in the assignment of the physical unit A3. (An available unit is one presently attached to the computer and not assigned for system use.)

INTERSYSTEM UNIT ASSIGNMENT

Intersystem unit assignments allow an object program to write an intermediate output file on a unit and then reserve that unit for later use as input or output in a different phase of a job. This is done by using the symbolic channels J through Q.

An intersystem output unit may be designated by using one of the following formats:

DESIGNATION	EXPLANATION
I	Intersystem channel
IM	Intersystem channel and model
I(k)	Intersystem channel and relative unit
I(k)M	Intersystem channel, relative unit, and model

The same formats may be used to designate an intersystem input unit, with the exception that the model specification should not be made. If a relative unit is specified, the parentheses are required.

In specifying an intersystem input unit, an additional parameter may be specified. This is an R which indicates to the Loader that the reserve status for the unit is to end after the current job segment is complete. For example, if unit J(1) is to be used as input for this job and then removed from reserve status, it would be specified as J(1)R in the $FILE card or FILE pseudo-operation.

Full IOCS consists of three files on the System Library Unit. The first file contains the communication words, which will be used by the object program when it must call any of the IOCS subroutines. It also contains a loader, which is first used to load in the Preprocessor and is later used by the Preprocessor to load in the number of overlays required to reduce the LABELS package to the level of IOCS specified by the programmer.

The second file contains the Preprocessor, a postprocessor, the full LABELS package of IOCS, and the overlays which are used to change the IOCS configuration for a particular run. The postprocessor performs housekeeping operations at the end of a job.

The third file contains the routines in Random IOCS.

Full IOCS functions in the following general manner:

1. The $EXECUTE IOCS card is used to call Full IOCS. The System Monitor then initiates loading of the first IOCS file from the System Library Unit.

2. The Preprocessor reads a set of control cards from the System Input Unit and creates a group of contiguous file control blocks in storage. Unless otherwise specified, the Preprocessor prints a list of files which the program is to use. The following information is given for each file:
   a. File Number
   b. Physical input/output unit assigned to the file
   c. Starting reel sequence number
   d. If an output file, a statement on the type of tape to be mounted (labeled or unlabeled reel)

NOTE: A file that is to be mounted immediately is indicated by an asterisk (*) to the left of the printed line in the file list.

The Preprocessor also prints an error list if an error is detected while the control cards are being processed. If any errors have been found in the control cards, a single halt occurs just prior to program loading.

3. Upon recognition of the *LOAD card, the Preprocessor initiates execution of a load sequence that loads the object program. The load program must be supplied by the programmer. The unit on which the load program is located may be specified in the *LOAD card. Examples of such load programs are provided later in this section.

4. The object program is loaded and executed.

5. The programmer normally ends his program by transferring to the symbolic location IOCS. The system will then print a message indicating that the job has been completed, will set any reserve unit status for intersystem use, and will resume scanning for IOCS control cards on the System Input Unit.

Rules for Assembly

The IOCS subroutines are assembled as a group and are located in core storage immediately following the IOEX portion of the System Monitor. All storage beyond the IOCS routines is available for an object program, file control blocks, and buffer pools.

1. Source program origin — The source program should be written to be assembled at an origin above the level of IOCS used. To make the most efficient use of core storage, it is advisable to consult an IOCS listing for the last location used by each level of IOCS.

2. File control blocks — The file control blocks for the files used in the program must be reserved in contiguous locations. The blocks thus occupy a single storage area consisting of a succession of 12-word blocks. The entire area is referred to as the file block.

In making entries in the file number field (columns 13-15) of the *FILE card the file whose file control block is first in the file block is designated by 1; the file represented by the second block is designated by 2; the third is designated by 3, etc.

In no case should the load program place any part of the object program in the file block, since the file control blocks are used by the Preprocessor to store information generated from the *FILE cards.

3. Buffer pools — The space reserved for buffer pools need not be contiguous with the file block.

4. Reference to system subroutines — Within his source program, the Full IOCS programmer must define the transfer point to the IOCS subroutines that he uses. These transfer points are located relative to the origin of IOCS itself, as shown in Figure 26.

The actual location corresponding to symbolic location IOCS may vary from installation to installation. For any installation, however, this location and those defined relative to it, as shown above, can be expected to remain fixed, even when modifications are made to IOCS itself.

The SST pseudo-operation in the FORTRAN II Assembly Program (IBSFAP) can be used to define SYSORG.

In addition to the subroutine transfer points shown in Figure 26, there are other relative locations within the IOCS communication region that might be of interest to the Full IOCS programmer. These additional relative locations are provided in Appendix A.

Location	Operation	Address, Tag, Decrement/Count
1	8	16
IOCS	EQU	SYSORG
DEFINE	EQU	IOCS+4
JOIN	EQU	IOCS+6
ATTACH	EQU	IOCS+8
CLOSE	EQU	IOCS+10
OPEN	EQU	IOCS+12
READ	EQU	IOCS+14
WRITE	EQU	IOCS+16
COPY	EQU	IOCS+18
REW	EQU	IOCS+20
WEF	EQU	IOCS+22
BSR	EQU	IOCS+24
BSF	EQU	IOCS+26
CKPT	EQU	IOCS+28
STASH	EQU	IOCS+30
READR	EQU	IOCS+32
RELESE	EQU	IOCS+34
MWR	EQU	IOCS+38
RANDEF	EQU	IOCS+76
RANREQ	EQU	IOCS+77
RANCLS	EQU	IOCS+78
RANRPL	EQU	IOCS+79
RANFLG	EQU	IOCS+80 (Note)
RANCAV	EQU	IOCS+81 (Note)

Note: These words are pointers, and are indirectly addressed. See the section entitled "Random Processing Using Disk and Drum Storage Units."

Figure 26. Transfer Points to IOCS Subroutines

Program Input

Figure 27 shows a sample input deck for execution of a Full IOCS program.

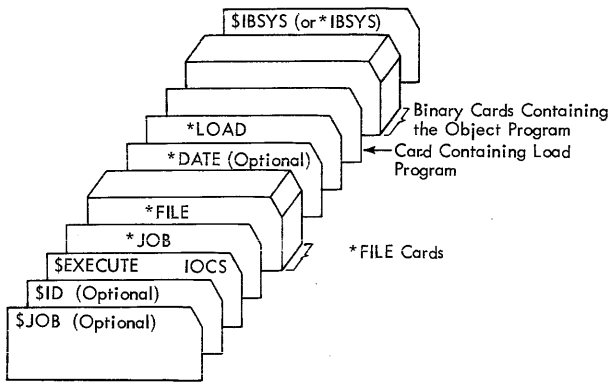


Figure 27. Input for Full IOCS

Control Cards

Eleven control cards are recognized by the Preprocessor. They are the *JOB, *FILE, *DATE, *LOAD, *RESTART, *IBSYS, $JOB, $ID, $IBSYS, $EXECUTE, and $STOP cards. The descriptions of these control cards are given in the following text.

*JOB Card

The *JOB card supplies the name of the job, defines the length of the file block and its origin in core storage, and specifies the IOCS configuration to be used by the program. If no file block origin is given, the file block is located beginning at the standard origin. The standard origin is defined as the next location following the end of the IOCS configuration specified. The origin of the file block is contained in the address portion of symbolic location LFBLK (IOCS+57) in the IOCS communication region.

The format of the *JOB card is shown in Figure 28. The contents of the card are as follows:

COLUMNS	CONTENTS
7-10	*JOB
13-30	Job Name – This is the name assigned to the job by the programmer.
33-35	The number of file control blocks (decimal) to be generated by the Preprocessor is entered in this field (right-justified).
44-48	File Block Origin (octal) – If this field is blank, the file block is located at the standard origin.
55-61	Desired IOCS configuration for this job: LABELS – This configuration provides the complete IOCS, including all sequential, labeling, and label-checking routines. BASIC – This configuration contains all sequential IOCS features, except the labeling routines. MINIMUM – This configuration deletes the use of internal files and deletes the following routines from Basic IOCS: BSF CKPT COPY JOIN REW STASH WEF EXECUTOR – This option consists of only the trap supervisor and the IOCS communication region.
63-66	Random IOCS Designation – The characters RAND in these columns specify Random IOCS for this job.

*FILE Card

Each file used in the program must be described in an *FILE card. This control card contains all of the information needed by the Preprocessor to generate a file control block. Space is provided on the card for a file number, which defines the relative location of the file control block within the file block. The order of *FILE cards in the control card set is immaterial.

The format of the *FILE card is shown in Figure 28. The contents of the card are as follows:

COLUMNS	CONTENTS
7-11	*FILE
13-15	This number indicates the relative position (decimal) of the file control block within the file block. It must not be greater than the number punched in columns 33-35 of the *JOB card.
17	Tape mounting indicator: * – This specifies that the file is to be mounted before processing is begun. blank – This specifies that it is not necessary for the file to be mounted when processing is begun.
18-21	Primary unit designation: This is the symbolic designation of the primary input/output device for the file. Entries in this field are right-justified. Unit assignment notations are ex-

COLUMNS	CONTENTS
	plained in the section "Unit Assignment in Full IOCS."
22-25	Secondary unit designation: For magnetic tape files, this is the symbolic designation of the secondary input/output device. Entries in this field are right-justified. Unit assignment designations are explained in the section "Unit Assignment in Full IOCS."
	For disk files, this field designates the number of consecutive cylinders that make up the file. The number must be right-justified in the card field and must not exceed 250.
	For drum files, this field designates the number of consecutive cylinders that make up the file. The number must be right-justified in the card field and must not exceed 10.
27	List control:
	N – This specifies that this file is not to be included in the Preprocessor file list.
	blank – The file is to be included in the file list.
28	File type:
	C – Checkpoint I – Input T – Total block output P – Partial block output
29	Reel control flag:
	M – This designates an unlabeled, multireel file.
	L (Labeled files only) – This specifies that a label search is to be made when the file is opened.
	blank – This specifies that the file is an unlabeled single-reel file, or that no label search is to be made when this labeled input file is opened.
30	For IBM 729 Magnetic Tape Units, this field specifies tape density:
	H – High density L – Low density
	For IBM 7340 Hypertape Drives, this field specifies rewind and file protection at reel-switching time:
	blank– No rewind, no file protection 1 – Rewind, no file protection 2 – No rewind, file protection 3 – Rewind, file protection
31	File mode:
	D – BCD B – Binary M – Mixed mode, first record BCD N – Mixed mode, first record binary
32	Labeling conventions:
	H – High density 729 header labels L – Low density 729 header labels S – Standard density 729 header labels 1 – Hypertape labels blank – No labels
33	Block sequence numbering flag (significant for binary files only):
	S – This specifies that IOCS should perform block sequencing on this file.
	N – No block sequencing should be performed.
34	Check sum flag (available for binary sequence numbered files only):
	C – This specifies that IOCS should perform check sum operations.
	blank – Perform no check sum operations.
35	Checkpoint conventions:
	F – This specifies that IOCS should write a checkpoint on this labeled file at each reel switch.
	C – This specifies that IOCS should write a checkpoint on the checkpoint file at each reel switch of this labeled file.
	blank – No checkpoints are to be initiated by this file.
36	Restart positioning flag:
	N – This specifies that this file should not be repositioned at restart.
	blank – This specifies that the file is to be repositioned at restart.
37	(Disk or drum storage unit) – Write check option flag:
	W – This specifies that IOCS should perform write checking on this output file.
	blank – No write checking is to be performed.
38-41	Reel sequence number field:
	For a labeled tape file – This specifies the reel sequence number of the first reel of this file.
	For a disk file – This field denotes the numeric starting cylinder number (0-249). It must be right-justified. The starting cylinder number is designated as "load point" (beginning-of-tape) for this file. IOCS computes the valid head and track limits for this file by using the secondary unit designation and the designation in this field. This number (the head and track limit) must not exceed 250 cylinders.
	For a drum file, this field denotes the numeric starting cylinder number (0-9). It must be right-justified. The sum of the entries in columns 22-25 and columns 38-41 may not exceed 10.
44-48	File serial number field:
	For a labeled tape file – This field contains the file serial number.
	For a labeled disk or drum file—Columns 44 and 45 may contain the home address identifier (HA2) for this file. This is given only if the file is labeled. This means that the System Editor has previously written this HA2 in those cylinders that are designated by the secondary unit field and the reel sequence number field.
50-53	Number of days this labeled file is to be retained (0000-9999). This field must not contain any leading blanks.
55-72	An arbitrary name assigned to the file for external recognition. This is the name of the file that will appear in file listings on the printer. It may or may not be the name of the file used in the program coding.

*DATE Card

The *DATE card provides IOCS with a creation date for any labeled output files produced by the program. If this control card is absent, the date is taken from the System Monitor.

The format of the *DATE card is as follows:

COLUMNS	CONTENTS
7-11	*DATE
13-14	Month (e.g., 06, 11)
15-16	Day (e.g., 03,21)
17-18	Year (e.g., 64,65)

*LOAD Card

This card initiates loading of the object program. It must be the last control card before IOCS is to initiate loading of the object program. This control card designates the unit on which the loading program is located. The loading program itself must be supplied by the programmer.

The format of the *LOAD card is shown in Figure 28. The contents of the card are as follows:

COLUMNS	CONTENTS
7-11	*LOAD
16-21	System unit designation: (e.g., SYSIN2). If this field is blank, System Input Unit 1 is assumed. This System Input Unit cannot be a disk or drum storage unit.

*RESTART Card

This control card initiates a restart from some previous checkpoint.

The format of the *RESTART card is as follows:

COLUMNS	CONTENTS
7-14	*RESTART

*IBSYS Card

This control card transfers control to the System Monitor.

The format of the card is as follows:

COLUMNS	CONTENTS
7-12	*IBSYS

$JOB Card

This control card defines the beginning of a job. It causes IOCS to return control to the System Monitor.

The format of the card is as follows:

COLUMNS	CONTENTS
1-4	$JOB
16-72	Name assigned to the job

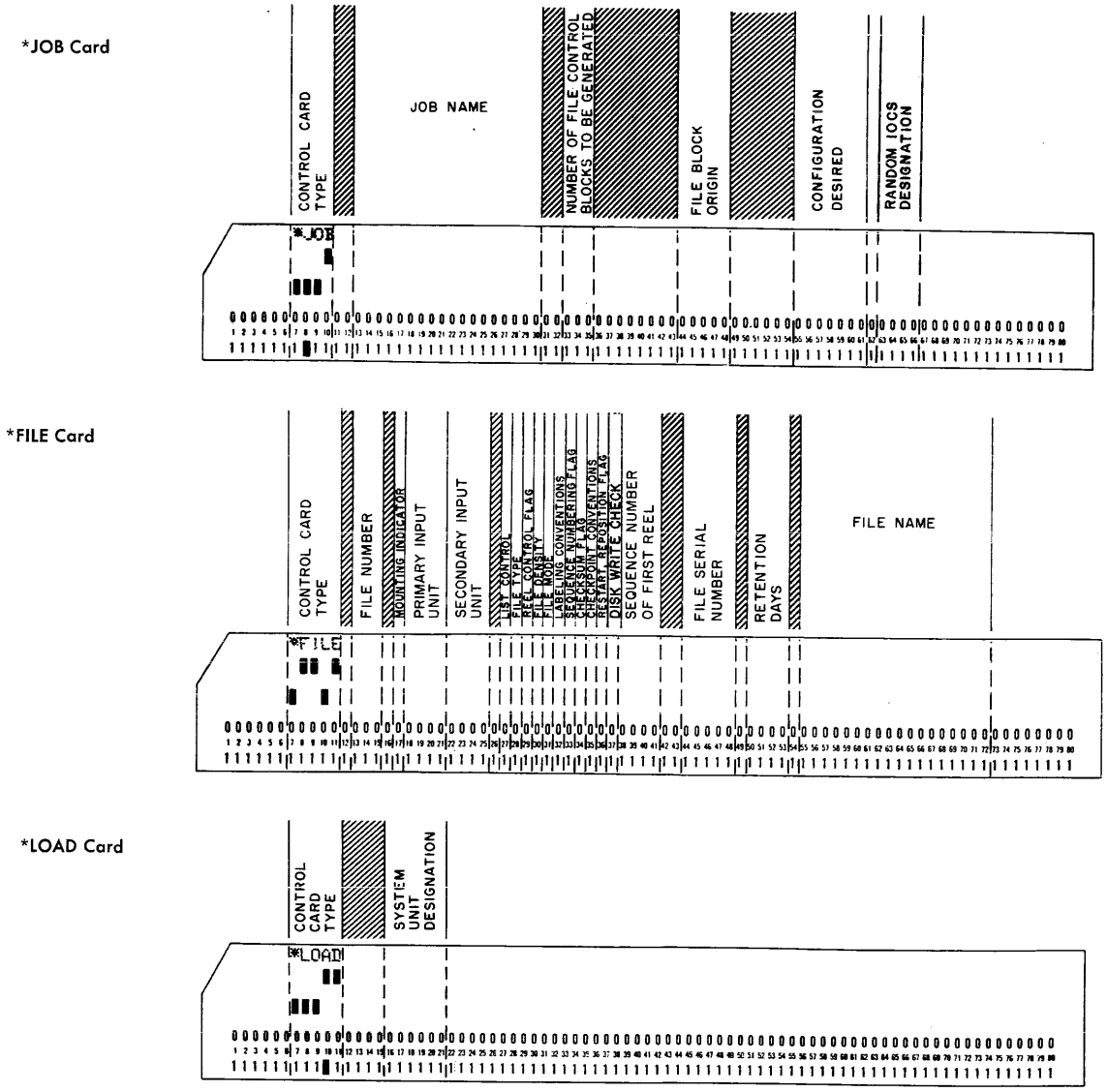


Figure 28. Formats of *JOB, *FILE, and *LOAD Control Cards

$ID Card

This control card causes transfer of control to the installation accounting routine.

The format of the card is as follows:

COLUMNS	CONTENTS
1-3	$ID
7-72	Any combination of alphameric characters or blanks

$IBSYS Card

This control card transfers control to the System Monitor.

The format of the card is as follows:

COLUMNS	CONTENTS
1-6	$IBSYS
7-72	blank

NOTE: The $IBSYS and the *IBSYS cards perform the same function: *IBSYS is a Preprocessor control card; $IBSYS is a System Monitor control card.

$EXECUTE Card

This control card causes control to be returned to the System Monitor. The System Supervisor is then brought into core storage. The System Supervisor positions the System Library Unit to the subsystem specified in the variable field of the $EXECUTE card, reads in the first record, and relinquishes control to it.

The format of the control card is as follows:

COLUMNS	CONTENTS
1-8	$EXECUTE
16-72	Name of the subsystem to which control should be relinquished.

NOTE: Upon recognition of a $EXECUTE IOCS card, IOCS retains control without returning to the System Monitor.

$STOP Card

This control card causes the System Monitor to print a message and halt. The message indicates the physical unit assignment and tape positions (record and file count) of the System Input Unit, System Output Unit, and System Peripheral Punch Unit.

The format of the card is as follows:

COLUMNS	CONTENTS
1-5	$STOP

Program Loading

Full IOCS does not contain a load program to load the object program. This permits the installation to retain complete flexibility in loading.

The Preprocessor initiates execution of a load program from the unit designated in the *LOAD control card. The loader must not destroy any portion of the System Monitor, IOCS, or the file block.

The following are sample loaders for card reader, 729 tape units, and 7340 Hypertape.

Sample of a Card Reader Load Program

The sample load program listed below loads an object program into core storage from the on-line card reader on channel A. The object program must be in standard row binary card format.

```
        PCC
        REF
        COUNT   20
        ABS
        SST
        ORG     SYSEND-12
        FUL
        IOCD    LOAD,,13
        TCOA    1
        TTR     LOAD
LOAD    RCDA                    READ CARD
                                READER
        RCHA    LOAD1
        LCHA    LOAD2
        CLM
        STP     LOAD3
        LXD     LOAD3,1
        TXL     LOAD3+1,1,0     TRANSFER
                                CARD
        LCHA    LOAD3
        TTR     LOAD
LOAD1   IOCT    LOAD3,1
LOAD2   IOCT    **,2,1
LOAD3   PZE     **              9ROW LEFT
        TTR*    LOAD3
        END
```

Sample of a 729 Load Program

The sample load program listed below loads an object program into core storage from 729 magnetic tape. The object program must be in standard column binary card format. In the sample program, the object program is loaded from unit B2.

```
        PCC
        REF
        COUNT   20
        ABS
        SST
        ORG     SYSEND-16
        FUL
        IORT    LOAD,,320
        TCOB    1
        TRA     LOAD
LOAD    RTBB    2               B-2(SYSPP1)
        RCHB    LOAD1
        LCHB    LOAD2
        CAL     LOAD3
        ERA     =O300500000000
        SLW     LOAD3
        ANA     =O37000000
        TZE*    LOAD3           TRANSFER
                                CARD
        TRA     LOAD
LOAD1   IOCT    LOAD3,,1
LOAD2   IOCP    LOAD3+1,,1      9ROW LEFT
LOAD3   IORT    **,,**
        PZE
        END
```

Sample of a Hypertape Load Program

The sample Load program listed below loads an object program into core storage from a 7340 Hypertape, where the object program resides in column binary card format. In the sample program, the Hypertape drive is attached to channel C.

```
            PCC
            REF
            COUNT   20
            ABS
BEGIN       BOOL    77715
            ORG     BEGIN
            FUL
            CPYD    *+3,,320
            WTR     *+2
            TRA     LOAD
LOAD1       OCT     220000200000    CTLRN
            CPYP    LOAD2+25,,2
            CPYD    LOAD2+1,,24
            WTR     LOAD2
LOAD        STCC
            TCOC    *               WAIT
            AXT     0,2
            CAL     LOAD2+25
            STA     LOAD2   STARTING
                            LOCATION
            ANA     LOAD2-1
            STD     LOAD2   COUNT OF WORDS
                            ON THIS CARD
            TZE*    LOAD2   EXIT IF THIS IS
                            END CARD
            ADD     LOAD2-2
            ARS     18
            STA     *+2
            CAL     *+3
            SLW     **  ENDING LOCATION & 1
            TRA     LOAD
            ONE     LOAD1,,LOAD2+1  TCH
                                    LOAD1
            OCT     37000000
LOAD2       XMT     **,,**
            END
```

When the Preprocessor routine of IOCS reads the *LOAD card, it causes the card containing the sample Hypertape load program to be read into storage. The Preprocessor then transfers control to the load program in the following manner:

```
RSCc    *+3     c = channel
TCOc    *
TRA     2
SMS     2*s+10  s = setting of data
                    channel switch
CTLR    *+3
CPYP    0,,3
TCH     0
HSEL    u       u = unit
HEOS
```

The Preprocessor determines the values of c, s, and u from the information in the *LOAD card.

Installation Modifications

Each installation may change the IOCS records on the system tape to reflect its machine configuration. IOCS must be reassembled if it is to be used with 1301 Disk Storage, 7320 Drum Storage, or 7340 Hypertape.

The released system tape contains the following symbolic EQU cards:

```
NOCH    EQU     0   (No 1301 or 7320 modules)
NOHYP   EQU     0   (No Hypertape channels)
```

The 0 operand for NOCH causes all 1301 and 7320 coding to be overlaid. Similarly, the 0 operand for NOHYP causes all 7340 coding to be overlaid. The number of Hypertape units or 1301 and/or 7320 modules to be used with IOCS must be specified in the operands of these two pseudo-operations.

The released system tape contains the following sense switch assignments:

```
SWTCP   EQU     2   (Checkpoint Control)
LERIG   EQU     3   (Label Control)
```

These operands may be changed to reflect nonstandard sense switch assignments.

The released IOCS tape uses a SYSORG of 3720_8 (or 2000_{10}). This origin can be changed by reassembly of IOCS.

Procedures for assembling IOCS are discussed in the publication *IBM IBSYS Operating System, System Monitor (IBSYS)*, Form C28-6248.

Unit Assignment in Full IOCS

The following text indicates the manner in which primary and secondary units are specified for files in Full IOCS. These specifications are entered in the primary and secondary unit fields of the *FILE control card. The entry in each unit assignment field must be right-justified.

The following notation is used in later explanations to indicate the formats in which actual symbols are entered into the unit assignment fields:

X	denotes a real channel (A through H).
P	denotes a symbolic (unspecified physical) channel (S through Z).
I	denotes a symbolic (specially-flagged) channel for intersystem use (J through Q).
M	denotes the model number of an IBM 729 Magnetic Tape Unit (II or IV). Only II or IV can be specified. Model V units are also signified by entering II, and Model VI units are signified by entering IV.
D	designates IBM 1301 Disk Storage.
N	designates IBM 7320 Drum Storage.
H	designates an IBM 7340 Hypertape Drive.
k	denotes a unit number (0 through 9).
a	denotes an access mechanism number (0).
m	denotes a module number (0 through 9).
s	denotes a data channel switch (also called interface).

Unit Assignment Specifications

Unit assignment specifications may be made in any of the following formats:

DESIGNATION	EXPLANATION
blank	(For secondary units only) Any available unit is assigned to the file.

DESIGNATION	EXPLANATION
M	Any available IBM 729 Magnetic Tape Unit of this model is assigned to the file.
X	Any available unit on this real channel is assigned to the file.
P	All files in a job having the same channel designation are assigned to the same channel.
Xk	The *k*th relative unit (a unit not assigned to a system unit function) of this channel is assigned if available. If not available, the first relative unit is assigned.
PM	Any available unit of the model indicated by *M* and on the symbolic channel specified by *P* is assigned to the file.
PkM	An available unit on the symbolic channel indicated by *P*, having the model number indicated by *M*, is to be assigned to the file. In this case, *k* indicates the relative unit.
XDsm	This designation is made in the primary unit field for disk assignment. The format *XDsm* represents the only assignment available, where *X* is any real channel, *D* designates disk, *s* is the data channel switch, and *m* is the module number. For data channel switch 1, s = 0 For data channel switch 2, s = 2 When a disk unit is specified in the primary unit field, the programmer must use the secondary unit field to enter a numeric value indicating the number of consecutive cylinders given to IOCS for this file. The number cannot exceed 250 and must be right-justified.
XNsm	This designation is made in the primary unit field for drum assignment. The format *XNsm* represents the only assignment available, where *X* is any real channel, *N* designates drum, *s* is the data channel switch (0, 2), and *m* is the module number (0, 2, 4, 6, 8).
XHks	The 7340 Hypertape Drive that is unit *k* on channel *X* with data channel switch *s* is to be assigned to the file. No symbolic or intersystem assignments (P or I) are allowed. A Hypertape entry in the secondary unit field is made in the same format. For data channel switch 1, s = 0 For data channel switch 2, s = 1
*	An asterisk can be entered into the secondary unit field only. It indicates that the secondary unit for the file is to be any unit on the same channel and of the same model as the primary unit. This unit, if available, is assigned after all other unit assignments have been made. If units of a different model are available, one of them is assigned. If no units are available on the channel, the secondary unit is the same as the primary unit.
INT	If the file is an internal file, the characters INT should be placed in the primary unit field of the *FILE card.

System units can be designated as follows:

DESIGNATION	EXPLANATION
IN, IN1, IN2	The file is to be read from the current System Input Unit.
OU, OU1, OU2	The file is to be written on the current System Output Unit.
PP, PP1, PP2	The current System Peripheral Punch Unit is to be used for a punch output file.
UTk	The System Utility Unit specified by *k* is to be used for the file. *k* can be 1, 2, 3, or 4.
CKk	The System Checkpoint Unit specified by *k* is to be used by the file. *k* can be 1 or 2. When using the distributed version of IBSYS, these functions must be attached before they can be used.

Card equipment is assigned for the file as follows:

RDX	Card reader on channel *X*.
PRX	Printer on channel *X*.
PUX	Card punch on channel *X*.

INTERSYSTEM UNIT ASSIGNMENT

Intersystem unit assignments allow an object program to write an intermediate output file on a unit and then reserve that unit for later use as input or output by an object program in a different phase of the job. This is done by using the symbolic channels J through Q.

These symbolic channels have two meanings. For output assignments, they are equivalent to using symbolic channels S through Z, but they signal the System Monitor to place a reserve flag and indicative data into the unit control block for the unit. For input, the designations are used to make comparisons against data in unit control blocks to find the proper reserve unit.

An intersystem output unit may be designated in one of the following formats:

DESIGNATION	EXPLANATION
I	Intersystem channel
IM	Intersystem channel and model
Ik	Intersystem channel and relative unit
IkM	Intersystem channel, relative unit, and model

The same designations may be used to designate an intersystem input unit except that the model specification should not be made. If an intersystem unit has been reserved, an additional parameter may be used when the unit is specified as input. This is an R which indicates to the System Monitor that the reserve status for the unit is to end after the current job is complete. For example, if unit J1 is to be used for input and then removed from reserve status, it would be specified as J1R in the control card.

Appendix A: Key Words in the IOCS Communication Region

The following is a list of words and information-holding areas within the communication region at the beginning of IOCS. The words are defined relative to the origin of IOCS.

(The preceding section entitled "Control and Loading Information for Full IOCS" contains a listing of relative locations of routines which are of interest to the Full IOCS user only.)

LIBRARY IOCS LOCATION SYMBOL	RELATIVE POSITION IN LIBRARY IOCS	FULL IOCS LOCATION SYMBOL	RELATIVE POSITION IN FULL IOCS	CONTENTS
	(Use .JDATE)	DATE	IOCS+39	System date. The internal mode is BCD, and the word is of the form YYbXXX, where YY=Year XXX=Day of the year (001-365)
	.LAREA−3	JOBID	IOCS+40 to IOCS+42	Job name
.LAREA	.LAREA to .LAREA+13	LAREA	IOCS+43 to IOCS+56	Label area (14 words)
.LFBLK	.LFBLK	LFBLK	IOCS+57	This word defines the origin and length of the File Block prepared by the IBJOB Loader or by the Preprocessor. It is of the form: MZE a,,b where a=File Block origin b=File Block length
	.LFBLK+1	CPFILE	IOCS+58	Address: Current checkpoint file in use
	.LFBLK+2	CPSEQ	IOCS+59	1+sequence number of last checkpoint written
	.LFBLK+3	LTSX	IOCS+60	Decrement: 2s complement of the quantity: (1+the location of the IOCS command on which the error, if any, occurred) Tag: 0 – No error 1 – Block sequencing error 2 – Check sum error 4 – End of buffer and/or parity error Address: 2s complement of the location of the TSX instruction which last entered IOCS
	.LFBLK+4	XEOB	IOCS+61	Decrement: Location of the end-of-buffer exit
	.LFBLK+5	XEOF	IOCS+62	Address: Location of the end-of-file exit
	.LFBLK+6	XERR	IOCS+63	Decrement Location of the error exit
	.LFBLK+7	LTRAD	IOCS+64	The last history word loaded into the AC: Prefix: PZE Decrement: Count of the number of words remaining in the last buffer used Address: 1+the location of the last word processed

LIBRARY IOCS LOCATION SYMBOL	RELATIVE POSITION IN LIBRARY IOCS	FULL IOCS LOCATION SYMBOL	RELATIVE POSITION IN FULL IOCS	CONTENTS
	.LFBLK+8	TRANS	IOCS+65	Address: Location of the first word processed by the last IOCS command
	.LFBLK+9	WDCT	IOCS+66	Decrement: Word count of last IOCS command executed
	.LFBLK+10	IRS	IOCS+67	Address: Contents of index register 2 at last entry to IOCS Decrement: Contents of index register 1 at last entry to IOCS
	.LFBLK+11	SENSE	IOCS+68	Contents of sense indicators at last entry to IOCS
	.LFBLK+12	FCW	IOCS+69	Address: If=0, IOCS is not in control If≠0, this is the current file being used
.AREA1	.AREA1 to .AREA1+5	LAREA1	IOCS+70 to IOCS+75	Additional label area (6 words)

Appendix B: Contents of File Control Block

Figure 29 is a detailed diagram of the contents of the 12 words in each file control block.

In the diagram, each word is divided horizontally to indicate the different contents of the bit positions when the file is on 729 magnetic tape, 7340 Hypertape, or 1301 Disk Storage and 7320 Drum Storage.

Shaded areas in the diagram indicate positions that are *not* used. However, these bit positions must not be used by the object program.

The detailed explanations of the 12 words are given below. Items marked with an asterisk are generated in the file control block by (1) the Loader when using Library IOCS, or (2) the Preprocessor when using Full IOCS. The remaining positions are initially zero.

WORD	BITS	CONTENTS
1	S*	729, 7340: Mounting flag for secondary unit (The operator has been instructed to mount a reel on the secondary unit.) 1301/7320: Write-check flag. 0 — Ignore write-check on output file 1 — Perform write-checking
	1-2	Not used.
	3-17	729, 7340: Location of unit control block of secondary unit. If no secondary unit is used, this location is the same as the one in positions 21-35. 1301/7320: Beginning-of-file locator. For multi-file applications, this locator indicates the last "file mark" location written by IOCS by a closing or write end-of-file operation in this file. Initially the locator is the same as positions 3-17 of word 8.
	18*	729, 7340: Mounting flag primary unit (The operator has been instructed to mount a reel in the primary unit.)
	18	1301/7320: Not used.
	19-20	Not used.
	21-35*	Location of unit control block of primary unit. Note that if the file is an internal file, this word takes the form: PZE chain,,0 where chain = L(0) if no chain exists.
2	S*	Mixed mode file. 0 — Single mode 1 — Mixed mode Note that for a file that uses 7340 Hypertape, mixed mode is not permissible for reading backward.
	1-2*	Checkpoint control. 00 — No checkpoints are to be initiated by this file. 01 — Checkpoints are to be written on the checkpoint file at the beginning of every reel of this file. 10 — Checkpoints are to be written on *this* file at the beginning of every reel of the file (labeled output files only).
	3	File open. 0 — File is not open 1 — File is open
	4	Reserve file. 0 — Not a reserve file 1 — Reserve file
	5	File inhibit. 0 — File is inhibited 1 — File is not inhibited
	6*	File mode. 0 — BCD mode 1 — Binary mode
	7-8*	File type. 00 — Input file 01 — Partial block output file 10 — Total block output file 11 — Checkpoint file
	9*	Labeling. 0 — File is unlabeled. 1 — File is labeled.
	10-11*	Block sequence and check sums. 00 — No block sequence words. 10 — Block sequence word is present, no checks sums are present. 11 — Block sequence word is present, check sums are present (input) or are to be computed (output).
	12*	729, 7340: Reel control flag. 0 — Single reel file, if unlabeled; no label search, if labeled 1 — Multireel file, if unlabeled; if labeled, search for label when file is opened. 1301/7320: Not used.
	13	Buffer release. 0 — Buffer has not been released. 1 — Buffer has been released.
	14	Buffer rush. 0 — A rush has not occurred for this buffer. 1 — A rush has occurred for this buffer.
	15	End-of-tape. 1 — No. 0 — Yes.
	16	File permanently closed. 0 — No. 1 — Yes.
	17	Rush on buffers during this calling sequence. 0 — Rush has not occurred in this sequence. 1 — Rush has occurred in this sequence.
	18*	729: Density of label to be read. 0 — Low density. 1 — High density. 7340: Read backward mode. 0 — No. 1 — Yes. 1301/7320: Not used.
	19*	729: Label density to be written. 0 — Low density. 1 — High density. 7340: Rewind at reel switch.

WORD	BITS	CONTENTS
		0 – No. 1 – Yes. 1301/7320: Not used.
	20*	729: File density. 0 – Low density. 1 – High density. 7340: File protect at reel switch (input file only). 0 – No. 1 – Yes. 1301/7320: Not used.
	21-27	(Input) Number of buffers in logical use.
	28-35	(Input) Number of buffers ahead.
	21-35	(Output) Output file chain.
3	S	A regenerative internal file if S is a 1.
	1	A system unit if this position contains a 1.
	2	SYSIN1 or SYSIN2 if this position contains a 1.
	3-17	729, 1301, 7320: Location of entry point for nonstandard label image routines. 7340: Location of transfer vector for label field 27.
	18-20	Not used.
	21-35	Location of control word of buffer or buffer pool to be used with this file.
4	S*	Regular or internal file. 0 – Regular file. 1 – Internal file.
	1-2	Not used.
	3-17	Counter for block sequence checking.
	18-20	Not used.
	21-35	Location of Reserve Group or Internal Group control word, if this is a reserve or internal file.
5	S*	FILE card list control. 0 – List this FILE card. 1 – Do not list this FILE card.
	1-2	Not used.
	3-17	Location of buffer pool.
	18-20	Not used.
	21-35*	Buffer synchronization chain, or location of zero word (end of chain); or buffer request chain at end-of-tape condition.
6	S-1	Device flag. 00 – 729 10 – 1301 11 – 7320 01 – 7340
	2	Not used.
	3-17	729, 7340: Count of permanent parity errors on current input reel or count of erase areas on current output reel. 1301/7320: Number of retries for current file.
	18-20	Not used.
	21-35	729, 7340: Number of erase areas written for current output reel, or number of retries for current input reel. 1301/7320: Number of retries for current file.
7	S	1301/7320: If the S position contains a 1, IOCS will, during the open routine, put positions 3-17 of word 1 of the file control block in positions 21-35 of this word.
	S-35	729, 7340: File serial number in the form bXXXXX.
	1-5	1301/7320: Number of tracks (in binary) for this record.
	6-17	1301/7320: These bits comprise the HA2 for the Verify Track order obtained from the file serial number field of the FILE card.
	18-20	1301/7320: Not used.
	21-35	1301/7320: Binary head and track locations of current position of the file.
8	S-2	1301/7320: Not used.
	S-35	729, 7340: Reel sequence number in the form bXXXXb. If this file uses a 7340 Hypertape, the S position will contain a 1 during Loader time or Preprocessor time only.
	3-17	1301/7320: The first binary head and track number for this file. This is the logical "load point" for this disk file.
	18-20	1301/7320: Not used.
	21-35	1301/7320: The last binary head and track address for this file. This is the logical "end of tape" for this disk or drum file.
9	S-35	Retention days in the form bbXXXX.
10	S-35	First six characters of file name.
11	S-23	7340: Characters 7 through 10 of the file name.
	S-35	729, 1301, 7320: Second six characters of file name.
	24-35	7340: Characters 11 and 12 of the file name.
12	S-35	7340: Last six characters of file name if file is unlabeled. Reel serial number if the file is labeled. 729, 1301, 7320: Last six characters of file name.

Disk/Drum Flag Word

Files defined for disk or drum storage units are provided with one IOCS flag word per track, as described below:

BIT	CONTENTS
S	Record flag: 0 – Indicates record not complete on this track. 1 – Indicates all of the record is on this track.
1	End-of-file flag. 0 – Indicates no end of file on this track. 1 – Indicates an end of file.
2	Not used.
3-17	Chain address. This is the binary head and track location of the next sequential track in the "tape" file. If bit 1 is on (is a 1), bits 3-17 are the binary head and track location of the preceding "file mark" in the reel. At the time the file mark is written, bits 3-17 are set from bits 3-17 of word 1 of the file control block.
21-35	Track count. The address contains the number of actual words of the "tape record." It is a substitute for the Store Channel instruction which IOCS computes.

		S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
Word 1	729 7340	MF2			Location of Unit Control Block of Secondary Unit															MF1			Location of Unit Control Block of Primary Unit														
	1301 7320	WCF			Beginning-of-File Locator																																
Word 2	729 7340 1301 7320	Mixed Mode ?	Checkpoint Control		File Open ?	Reserve File ?	File Inhibit ?	BCD or Binary Mode ?	File Type		Label ?	Block Sequence and Check Sums		Reel Control Flag ?	Buffer Release ?	Buffer Rush ?	End of Tape ?	File Perm Closed ?	Rush This Sequence ?	Rd Label Density Backward Rd Mode	Wr Label Density Rew and File Protect at Reel Switch	File Density	Number of Buffers in Logical Use								Number of Buffers Ahead					
Word 3	729 7340 1301 7320	Regen Int File ?	A System Unit ?	SYSIN1 or SYSIN2 ?	Location of Entry Point for Nonstandard Label Image Routine Location of Transfer Vector for Label Field 27 Location of Entry Point for Nonstandard Label Image Routine																		Location of Control Word of Buffer or Buffer Pool to be Used by this File													
Word 4	729 7340 1301 7320	Reg or Int File ?			Counter for Block Sequence Checking																		Location of the Group Control Word if this is a Reserve or Internal File													
Word 5	729 7340 1301 7320	File Card List ?			Location of Buffer Pool																		Buffer Synchronization Chain or Location of Zero Cell (End of Chain); or Buffer Request Chain at End-of-Tape Condition													
Word 6	729 7340 1301 7320	Device Flag 00 - 729 01 - 7340 10 { 1301 { 7320			Count of Erase Areas on Current Output Reel, or Count of Permanent Parity Errors on Current Input Reel Number of Retries for Current Input File																		Number of Erase Areas Written for Current Output Reel, or Number of Retries for Current Input Reel Number of Retries for Current Input Reel													
Word 7	729 7340	File Serial Number in the Form bXXXXX																																			
	1301 7320		Number of Tracks (in Binary) For this Record		HA2																		Binary Head and Track Location of Current File Position													
Word 8	729 7340	Reel Sequence Number in the Form bXXXXb																																			
	1301 7320				Binary Head-Track Number of "Load Point"																		Last Binary Head and Track Address													
Word 9	729 7340 1301 7320	Retention Days in the Form bbXXXX																																			
Word 10	729 7340 1301 7320	First Six Characters of File Name																																			
Word 11	729	Second Six Characters of File Name																																			
	7340	Characters 7-10 of File Name																					Characters 11 and 12 of File Name													
	1301 7320	Second Six Characters of File Name																																			
Word 12	729	Last Six Characters of File Name																																			
	7340	Last Six Characters of File Name, or Reel Serial Number																																			
	1301 7320	Last Six Characters of File Name																																			

Figure 29. Format of the File Control Block

Execution of IOCS Read Commands

IOCS Command	IOCS Command changed to	History Record in AC	Words "Read"	Buffer Released	EOB Exit
IOCy A,,n n ≤ m n > m		PZE A+n,,m-n PZE A+n,,s-n+m	n n	No First	No No
IOCD A,,n n ≤ m n > m		PZE A+n,,m-n PZE A+n,,s-n+m	n n	Yes All	No No
IOSy A,,n n ≤ m n > m		PZE A+n,,m-n PZE A+m,,0	n m	No No	No No
IORy A,,**	IORy A,,m	PZE A+m,,0	m	Yes	No

SKIP

IOCyN **,,n n ≤ m n > m	IOCyN B,,n IOCyN C,,n	PZE B+n,,m-n PZE C+n-m,,s-n+m	n n	No First
IOCDN **,,n n ≤ m n > m	IOCDN B,,n IOCDN C,,n	PZE B+n,,m-n PZE C+n-m,,s-n+m	n n	Yes All
IOSyN **,,n n ≤ m n > m	IOSyN B,,n IOSyN B,,n	PZE B+n,,m-n PZE B+m,,0	n m	No No
IORyN **,,**	IORyN B,,m	PZE B+m,,0	m	Yes

EOB≠0

IOCS Command changed to	History Record in AC	Words "Read"	Buffer Release Action	EOB Exit
	PZE A+n,,m-n PZE A+n,,s-n+m	n n	No Conditional Hold First	No No
	PZE A+n,,m-n PZE A+n,,s-n+m	n n	Conditional Hold Conditional Hold Both	No No
	PZE A+n,,m-n PZE A+m,,0	n m	No No	No No
IORy A,,m	PZE A+m,,0	m	Conditional Hold	No

LOCATE

IOCyN B,,n IOCyN B,,n	PZE B+n,,m-n PZE B+m,,0	n m	No Hold	No Yes
IOCDN B,,n IOCDN B,,n	PZE B+n,,m-n PZE B+m,,0	n m	Hold Hold	No Yes
IOSyN B,,n IOSyN B,,n	PZE B+n,,m-n PZE B+m,,0	n m	No No	No No
IORyN B,,m	PZE B+m,,0	m	Hold	No

Execution of IOCS Write Commands

EOB = 0

IOCS Command	IOCS Command changed to	History Record in AC	Words "Written"	Buffer "Written"
IOCy A,,n n≤m n>m		PZE A+n,,m-n PZE A+n,,s-n+m	n n	No First
IOCD A,,n n≤m n>m		PZE A+n,,m-n PZE A+n,,s-n+m	n n	Yes All
IOSy A,,n n≤m n>m		PZE A+n,,m-n PZE A+n,,s-n	n n	No First
IORy A,,n n≤m n>m		PZE A+n,,m-n PZE A+m,,0	n m	Yes Yes

SKIP

IOCyN **,,n n ≤ m n > m	IOCyN B,,n IOCyNC,,n	PZE B+n,,m-n PZE C+n-m,,s-n+m	n n	No First
IOCDN **,,n n ≤ m n > m	IOCDN B,,n IOCDN C,,n	PZE B+n,,m-n PZE C+n-m,,s-n+m	n n	Yes All
IOSyN **,,n n ≤ m n > m	IOSyN B,,n IOSyN C,,n	PZE B+n,,m PZE C+n,,s	0 0	No First
IORyN **,,n n ≤ m n > m	IORyN B,,n IORyN B,,n	PZE B+n,,m-n PZE B+m,,0	n m	Yes Yes

EOB≠0

IOCS Command changed to	History Record in AC	Words "Written"	Buffer Release Action	EOB Exit
	PZE A+n,,m-n PZE A+n,,s-n+m	n n	No Conditional Hold First	No No
	PZE A+n,,m-n PZE A+n,,s-n+m	n n	Conditional Hold Conditional Hold Both	No No
	PZE A+n,,m-n PZE A+n,,s-n	n n	No Conditional Hold First	No No
	PZE A+n,,m-n PZE A+m,,0	n m	Conditional Hold Conditional Hold	No No

LOCATE

IOCyN B,,n IOCyN B,,n	PZE B+n,,m-n PZE B+m,,0	n m	No Hold	No Yes
IOCDN B,,n IOCDN B,,n	PZE B+n,,m-n PZE B+m,,0	n m	Hold Hold	No Yes
IOSyN B,,n IOSyN C,,n	PZE B+n,,m PZE C+n,,s	0 0	No Conditional Hold First	No No
IORyN B,,n IORyN B,,n	PZE B+n,,m-n PZE B+m,,0	n m	Hold Hold	No No

Figure 31. Execution of IOCS Commands

Appendix D: Actions of IOCS Routines Under Abnormal Conditions

LIBRARY IOCS ROUTINE	FULL IOCS ROUTINE	CONDITION	ACTION
.DEFIN	DEFINE	Attempt to define a buffer pool inside the system boundaries	Pool error
.JOIN	JOIN	Pool 2 in use (or already joined to a different pool)	Pool error
		Pool 1 joined to any buffer pool	Pool error
.ATTAC	ATTACH	Attempt to attach a closed file that has been rewound and unloaded	Initialization error
		Attempt to attach to pool in use	Unrealiable results
		Not enough buffers in a buffer pool	Attach error
		Attempt to open a file that is already open	NOP
.OPEN	OPEN	Attempt to open a file that was rewound and unloaded	Initialization error
.CLOSE	CLOSE	Attempt to close a file that has been rewound and unloaded	NOP
.READ	READ	File never opened	End-of-file exit
		File already closed	End-of-file exit
		File is an output file	NOP
		Block sequence error	Error exit; MQs$_{-2}$=1
		Check sum error	Error exit; MQs$_{-2}$=2
		Parity error	Error exit; MQs$_{-2}$=4
		Sequence and parity errors	Error exit; MQs$_{-2}$=5
		Check sum and parity errors	Error exit; MQs$_{-2}$=6
		Attempt to locate information in two buffers with one command	End-of-buffer exit; MQs$_{-2}$=2
		Attempt to locate in more buffers than are available	End-of-buffer exit; MQs$_{-2}$=4
		Attempt to transmit a word into the system	Illegal transmit
.WRITE	WRITE	File never opened	NOP
		File already closed	NOP
		File in an input file	NOP
		Attempt to locate information in two buffers with one command	End-of-buffer exit; MQs$_{-2}$=2
		Attempt to locate in more buffers than are available	End-of-buffer exit; MQs$_{-2}$=4
.COPY	COPY	Either file not open	NOP
		No buffer(s) connected to the input file	NOP
		Files not using same pool	Invalid file use
.REW	REW	File never opened	NOP
		File already closed	NOP
		File is labeled	NOP
.WEF	WEF	File never opened	NOP
		File already closed	NOP
		File is input file	NOP
		File is labeled	NOP
.BSF	BSF	File not opened	NOP
		File already closed	NOP
		BSF one file after end-of-file exit	NOP
.BSR	BSR	File not opened	NOP
		File already closed	NOP
		Following end-of-file exit	Beginning-of-file exit
		Tape positioned in front of the first record of a file	Beginning-of-file exit
.STASH	STASH	Either file not open	NOP
		Case 1: No buffer in FILE1 chain	NTS exit
		Case 2: FILE2 group full	NTS exit
		Case 3: No buffer in FILE1 chain	NOP
		Case 4: FILE2 group full	NTS exit

This appendix contains listings of on-line messages produced by both forms of IOCS. The messages that apply to both forms are listed first. A list of Preprocessor messages applicable only to Full IOCS is presented afterward.

In the lists, the form of the message is given first in capital letters, followed by an explanation of the message. If IOCS transfers control to a System Monitor routine after printing the message, the name of that routine is indicated at the beginning of the explanation.

Error messages produced by Random IOCS are explained at the end of the section "Random Processing Using Disk and Drum Storage Units" and are not included in this section.

Messages for Library and Full IOCS

ATTACH ERROR AT XXXXX
The System Monitor routine (STOPX is entered.
There are not enough buffers in the required pool to attach the specified files. XXXXX is the location of the TSX ATTACH, 4 instruction that caused the error.

BASIC IOCS NECESSARY XXXXX
A dump is taken.
Minimum IOCS was specified in the *JOB control card. However, a call has been made to internal files or to one of the following routines: JOIN, COPY, REW, WEF, BSF, or CKPT.

CHECKPOINT INVALID, MOUNT NEW REEL TO CONTINUE UNIT XXXXX CHECKPOINT XXXXXX CODE YYYYYY
An attempt to write a checkpoint on the specified tape has failed because of bad tape or because end of tape was encountered.

(File Identification) CYL. LIMITS XCEDED
The allotted number of cylinders specified in the secondary unit field of the control card or pseudo-operation has been exceeded. The end-of-buffer exit will be taken.

(File Identification) INPUT REEL, NO LABEL
The record read as the label of an input file was not the required length. A 1BLANK image will be created in the label area. This will result in a subsequent error message. There is no stop associated with this error.

(File Identification) NO TRAILER
A dump is taken if IOCS cannot continue. The record read as the trailer label of an input file was not the required length. The end-of-file condition is assumed. There is no stop associated with this error.

(File Identification) OPEN COUNT EXCEEDED-OPEN NOT POSSIBLE
The System Monitor routine (PAWSX is entered.
An attempt has been made to open too many files in a Reserve or Internal Group. If the Start key is depressed, the program continues without opening the file.

(File Identification) SEARCH SKIP LIST FOLLOWS IF SSWX DOWN REMOUNT
(label of file skipped)
. . . .
. . . .
. . . .

(File Identification) FOUND
A search is made for the header label on the current reel. If the desired header label is not found on this reel, reel switching will occur. If the specified sense switch is depressed, the reel currently being searched is rewound and unloaded. If another reel is mounted on that unit, the search will continue on that reel. The FOUND portion of the message is printed when the specified label is found.

ILLEGAL FILE USE
A dump is taken.
This message indicates that a file designated as a total block output file has been used for partial block output, or an attempt has been made to copy buffers when both files are not in the same pool.

ILLEGAL TRANSMIT { READ / WRITE } XXXXX
The System Monitor routine (STOPX is entered.
An attempt has been made to read a word from or to write a word into the area occupied by IOCS. XXXXX is the octal location of the IOCS command that caused the error.

IOBS NECESSARY XXXXX
A dump is taken.
EXECUTOR was specified in the *JOB card as the desired IOCS configuration. However, the object program has requested a routine in Minimum or Basic IOCS at octal location XXXXX.

I/O CHECK
The System Monitor routine (PAWSX is entered.
A machine error has probably occurred. No recovery by IOCS is possible.

LABEL IOCS NECESSARY XXXXX
A dump is taken.
A reference has been made to Label IOCS at octal location XXXXX. Labels were specified in an *FILE card but not on the *JOB control card.

NEED RANDOM
A dump is taken.
An attempt to TSX RANDEF, 4 has been made by the object program. The IOCS configuration in core storage does not contain Random IOCS.

POOL ERROR TSX AT XXXXX
The System Monitor routine (STOPX is entered.
An attempt has been made to do one of the following: (1) to define a buffer pool using the storage area occupied by IOCS; (2) to join a buffer pool to a nonexistent pool; (3) to join a pool to two different pools; or (4) to join two pools of different buffer sizes.

UNIT XXXXX
CHECKPOINT XXXXX CODE YYYYYYZZZZZZ
This message is printed immediately before a checkpoint

is written. The checkpoint identification number is XXXXX, and the restart code for this checkpoint is YYYYYYZZZZZZ. The physical tape address of the unit on which the checkpoint is taken is YYYYYY.

UNIT XXXXXX
EOT ON ERASE

The System Monitor routine (STOPX is entered.
End of tape was encountered while trying to erase tape on which a parity error occurred. No recovery by IOCS is possible.

UNIT XXXXXX (File Identification)
{ LABEL HIGH DENSITY / LABEL LOW DENSITY / DISK FILE LABEL / HYPERTAPE LABEL }
(followed by entire label image)

This message is printed to indicate the contents of the label on the specified unit.

UNIT XXXXXX (File Identification)
{ MOUNT / MOUNT BLANK / MOUNT LAB. BLANK } REEL XXXX

The System Monitor routine (PAUSE is entered if IOCS needs the new reel immediately. Otherwise, no pause occurs.

A tape should be mounted on the specified unit. If MOUNT BLANK is printed, a work tape should be mounted. If MOUNT LAB. BLANK is printed, a 1BLANK tape or a labeled tape on which the retention period has expired should be mounted. When reel switching occurs for multireel files, and only one input/output unit is assigned for the file, IOCS halts to permit the reels to be changed. If two units are assigned, IOCS does not halt. It automatically begins processing the second unit.

UNIT XXXXXX (File Identification) NOT AVAILABLE FOR INITIALIZATION

The System Monitor routine (PAWSX is entered.
An attempt has been made to attach a file which has already been attached to another pool or previously closed with a rewind and unload. If the Start key is depressed, the program continues but the file is not attached.

UNIT XXXXXX (File Identification) REEL XXXX-XXXX
RECORDS REDUNDANCY HISTORY XXXXX { RETRYS / ERASES }
XXXX PERM

This message shows the number of permanent parity errors (PERM.) when reading or writing, and also shows the number of parity errors corrected (RETRYS) when reading, or the number of erased areas (ERASES) when writing. The message is printed only if the error count (address portion of word 6 of the file control block) is nonzero.

UNIT XXXXXX (File Identification) REMOVE REEL XXXX

The specified reel has been unloaded. It may be removed from the tape drive.

UNIT XXXXXX (File Identification) LABEL ERROR
(Entire Label Image)
{ INPUT REEL INVALID / OUTPUT REEL INVALID / RETENTION NOT EXPIRED / NO LABEL, BLANK CREATED / IF SSWX DOWN IGNORED }

The System Monitor routine (PAUSE is entered.
Causes of the error:

a. INPUT REEL INVALID
Either the file serial number, the reel sequence number, or the file name is incorrect. If the label control sense switch is down when the Start key is depressed, the reel is accepted as correct. If the sense switch is up, the tape is rewound and unloaded. When a new tape is mounted, it will be checked.

b. OUTPUT REEL INVALID
The reel is not labeled (i.e., the label does not begin with 1HDR or 1BLANK). If the label control sense switch is down when the Start key is depressed, the dummy file serial number ***** is used for the label. If the sense switch is up, the reel is rewound and unloaded. When a new reel is mounted, the label will be checked.

c. RETENTION NOT EXPIRED
For the tape on the specified unit, (creation date + retention days) is greater than (current date). If the label control sense switch is down when the Start key is depressed, the reel is used for output. If the sense switch is up, the tape is rewound and unloaded. When a new reel is mounted, it will be checked.

d. NO LABEL, BLANK CREATED
The record read from an output file as the label was not the required length. If the label control sense switch is up, the tape is rewound when the Start key is depressed. When a new tape is mounted, it will be checked. If the sense switch is down, the system will proceed. A 1BLANK-type label is placed on the reel and processing continues. (Care should be used since the 1BLANK-type label may cause a subsequent error message.)

Preprocessor Messages

The following messages are applicable to Full IOCS only.

BASIC MONITOR HAS ENTERED INPUT/OUTPUT CONTROL SYSTEM

IOCS has received control from the System Monitor.

CYL LIMIT ERROR – MAX SIZE GIVEN PRESS START TO CONTINUE

The System Monitor routine (PAUSE is entered.
The *FILE card contains an error in the number of cylinders defined for the file. The upper limit is assumed to be the end of the disk or drum file.

ERROR IN RELOCATABLE CARD TYPE

The System Monitor routine (PAWSX is entered.
An error in card type has been detected in the Random IOCS record. This probably indicates a bad system tape.

END OF JOB

A dump is taken only if an error has occurred.

IOCS LOOKING FOR CONTROL CARDS

IOCS is looking for one of the following control cards: $ID, $JOB, $EXECUTE, $STOP, *IBSYS, *JOB, *RESTART.

IOCS RETURNING TO BASIC MONITOR

IOCS is returning control to the System Monitor.

JOB DISCONTINUED

This message indicates that execution of a job has been suppressed.

JOB – (Job Identification) DATE XX/XX/XX PAGE XXX

This is the format of the heading for the file list produced by the Preprocessor at the beginning of a job.

JOB-RESTART PAGE XXX

This is the format of the heading for the file list produced at restart.
The following is the format of the file list produced at either beginning of job or restart.

NO.	FILE NAME	UNIT	MOUNT TAPES
XXX	(File Identification)	XN	
		XDAM/S	TRKS XXXX-XXXX BLANK LO DEN LAB
		XNAM/S	TRKS XXXX-XXXX BLANK LO DEN LAB
		XHK/S	REEL XXXX HI DEN LAB
		PRX	UNLABELLED
		PUX	LABELLED
		RDX	
		+	
		CORE	

MOUNT INDICATED TAPES

The System Monitor routine (PAUSE is entered.

All files marked with an asterisk (*) to the left of the file number should be mounted immediately. Figure 32 is a sample of the file list produced at the beginning of a job.

REDUNDANCY ON SYSTEM TAPE

The System Monitor routine (PAWSX is entered.

A redundancy has occurred during reading of the System Tape.

IOCS may not be in core storage properly.

RESTART XXXXXX

SET SENSE SWITCHES

1	2	3	4	5	6
UP	UP	UP	UP	UP	UP
DN	DN	DN	DN	DN	DN

This message is printed during restart. XXXXXX is the checkpoint number that is being restarted. The sense switches should be set as indicated.

SET KEYS TO RESTART CODE

The System Monitor routine (PAUSE is entered.

The panel entry keys should be set to the restart code corresponding to the checkpoint record with which the restart is to begin (usually the last one written). The Start key should then be depressed.

THERE IS NO XXX UNIT

A dump is taken.

SYSXXX was requested in an *FILE card but was not attached in IBSYS.

UNIT XXXXXX

INCORRECT CHECKPOINT TAPE ADDRESS

A dump is taken.

An illegal unit address for the checkpoint tape has been specified in the panel keys during restart.

UNIT XXXXXX

NOISE RECORDS MAY CAUSE INCORRECT POSITIONING

Noise records detected while positioning unit XXXXXX may cause the restart to position the specified unit incorrectly.

UNIT XXXXXX

POSITIONING FAILED

A dump is taken.

The restart program has been unable to position unit XXXXXX properly.

XXXXXX UNITS NOT ASSIGNED. NONE AVAILABLE.

A dump is taken.

XXXXXX more input/output units were requested in *FILE cards than were attached in IBSYS.

INTER SYSTEM RESERVE INPUT NOT AVAILABLE.
IF SENSE SWITCH 4 DOWN, ANOTHER UNIT WILL BE ASSIGNED.
IF SENSE SWITCH 4 UP, JOB WILL BE SKIPPED.

The System Monitor routine (PAWSX is entered.

A reserve input file specified as J-Q in an *FILE card is not being held in reserve status as requested.

Preprocessor Control Card Error List

The following messages concern errors detected in Full IOCS control cards:

IOCS PREPROCESSOR ERROR LIST. JOB – (Job Identification) PAGE XX

This is the header line for the error messages which follow. This message is not printed unless an error, or errors, have been detected.

CARD XXX ILLEGAL OPERATION – (XXXXXX)

Columns 7-12 of the control card contain XXXXXX, which is not *JOB, *FILE, *DATE, *LOAD, *RESTART, or *IBSYS. The card is ignored.

CARD XXX FILE BLOCK ORIGIN NOT OCTAL

The origin specified in the *JOB card is not in octal form. The specified origin is ignored and the standard origin is used for the file block.

CARD XXX FILE CONTROL BLOCK COUNT ERROR

No count of the number of file control blocks to be generated by IOCS was provided in the *JOB card. An arbitrary count of 50 is used.

```
JOB-  TEST  6                DATE  03/01/64                PAGE      1

  FILE DESCRIPTIONS (MOUNT FILES MARKED WITH *)--

     NO.   FILE NAME          UNIT   MOUNT TAPES--
  *   1    FILE A             B5    REEL 0001
      2    FILE B             B6    REEL 0001          BLANK-HI DEN LAB
                              A5    REEL 0002          BLANK-HI DEN LAB
      3    FILE C             A3    REEL 0001          BLANK-LO DEN LAB
      5    FILE D             B1    REEL 0002          BLANK-LO DEN LAB
      7    FILE E             B2    REEL 0004          BLANK-HI DEN LAB
     10    FILE F           ED00/0  TRKS 8000 - 8399
     11    FILE G            DH0/1  REEL 0001          BLANK-LABELLED
                             DH1/1  REEL 0002          BLANK-LABELLED
  *  12    FILE H             A6    REEL 0002          BLANK-HI DEN LAB
     21    FILE K             PRA
     22    FILE L             B6    REEL 0001
                              A5    REEL 0002
     25    FILE M           ED01/0  TRKS 0000 - 0999 BLANK-LABELLED
  *  26    FILE N            CH0/1  REEL 0003
                             CH1/1  REEL 0004
```

Figure 32. Sample of Preprocessor File List

CARD XXX FILE BLOCK WILL NOT FIT
The count of the number of file control blocks, as entered in the *JOB card, is less than the number of *FILE cards. A dump is taken.

CARD XXX JOB CARD OUT OF PLACE
The *JOB card has been placed incorrectly in the input deck. The card is ignored.

CARD XXX FCB OVERLAPS IOCS
The file block origin has been specified at a location within IOCS. The standard origin is used for the file block.

CARD XXX FILE CONTROL BLOCK NUMBER ERROR
The file number specified in columns 13-15 of the *FILE card is either nonnumeric or greater than the count given in the *JOB card. No file control block is generated for this file.

CARD XXX UNIT 1 ILLEGAL
The primary unit field of the *FILE card does not contain a proper designation. No unit control block location is stored, and any reference to the file will cause an error halt.

CARD XXX UNIT 2 ILLEGAL
The secondary unit field of the *FILE card does not contain a proper designation. If the primary unit was correct, that designation is used for the secondary unit.

CARD XXX REEL SEQUENCE NUMBER NOT NUMERIC
The sequence number in the *FILE card is nonnumeric. The number is assumed to be 1.

CARD XXX HYPER UNIT NOT AVAILABLE
There is no unit control block for the Hypertape unit designated in the *FILE card. No unit control block location is stored in the first word of the file control block. Any reference to the file will cause an error halt.

CARD XXX NO SUCH ACCESS – MUST CORRECT
The *FILE card has asked for a nonexisting access mechanism. No unit control block location is stored in the first word of the file control block. Any reference to the file will cause an error halt.

CARD XXX DATE ERROR
The date specified in the *DATE card is not in a valid form. The date from the System Monitor is used.

CARD XXX SOME UNIT 1 IS (*)
The notation (*) has been specified in the primary unit field of the *FILE card. This is an invalid designation in a primary unit field. No unit control block location is stored in the first word of the file control block. Any reference to the file will cause an error halt.

CARD XXX CYLINDER START NON-NUMERIC
Alphabetic characters instead of numeric characters have been used in columns 38-41 of the *FILE card to specify cylinder number. The highest cylinder is assumed to be the starting cylinder.

CARD XXX CONTROL CARD EOF
The System Monitor routine (PAWSX is entered. More control cards are needed in System Input Unit 1. If none can be provided, the operator should take a dump.

CARD XXX CONTROL CARD REDUNDANCY
A redundancy has been detected while reading control cards from tape. The card is processed as read.

CONTINUE IF CONTROL CARD ERRORS CAN BE IGNORED
This message is printed at the end of the Preprocessor control card error list. The System Monitor routine (PAWSX is entered.
The operator should continue only if so instructed by the programmer. Otherwise, a dump (TRA 115) should be taken.

Appendix F: Standard Look-Ahead Words for Mixed-Mode Files

When processing a mixed-mode file, IOCS examines the last word of each physical record (look-ahead word) to determine the mode in which the next physical record is to be read or written. The octal representations of the standard look-ahead words are shown in Figure 33.

Standard Look-Ahead Word	Mode of Record Just Written	Mode of Next Record
XX XX 00 00 00 00	BCD	BCD
00 01 00 05 00 04	Binary	Binary
XX XX 11 11 07 07	BCD	Binary
00 00 00 01 00 00	Binary	BCD

Figure 33. Standard Look-Ahead Words

Actually IOCS uses only bits 21-23 of a look-ahead word in the determination of the mode. Therefore, if a mixed-mode file is not intended for the use of any system other than IOCS, look-ahead words of the form shown in Figure 34 may be used in place of the standard look-ahead words.

Look-Ahead Word	Mode of Record Just Written	Mode of Next Record
XX XX XX X0 XX XX	BCD	BCD
XX XX XX X5 XX XX	Binary	Binary
XX XX XX X1 XX XX	BCD	Binary
XX XX XX X1 XX XX	Binary	BCD

Figure 34. Look-Ahead Words For IOCS

The following glossary contains definitions of terms as used in this manual. The definitions do not necessarily apply to other systems.

Additional Label Information
: Information that can be placed in BCD positions 101 through 120 of the IBM standard 120-character label.

Alpha Record
: This term is used in the discussion of Random IOCS to describe the record that is taken from a sequential file. This record usually contains information used to update one or more beta records taken from a random access file.

.ATTAC (ATTACH)
: The IOCS routine that attaches a list of files to a buffer pool.

Available Buffer
: A buffer not presently in use, nor reserved for later use. For each buffer pool, there is a chain of available buffers.

Available Word Locator
: The first word of each buffer (the first buffer control word) which is used to keep track of the location of the next available word and the number of available words remaining in the buffer.

Available Words
: Those words in an input buffer that have not yet been "read," or those words in an output buffer that have not yet been filled with data.

Beta Record
: This term is used in the discussion of Random IOCS to describe the record secured from a random access file.

Block
: A physical record, i.e., a punched card, line of print, or the data contained between two interrecord gaps on magnetic tape, disk storage, or drum storage.

Block Count
: The count of the number of blocks (physical records) that have been processed on the current reel of a tape file, or the number of records processed on a disk or drum storage file or unit record file. The count is kept in the file control block for the file.

Blocking
: The process of combining more than one logical record into a block.

Block Sequence Word
: A word that can be appended to each block of a binary file in order to provide additional checking information. It contains the block sequence number and may also contain a folded check sum.

.BSF (BSF)
: The IOCS routine that backspaces a tape, disk, or drum over a specified number of files.

.BSR (BSR)
: The IOCS routine that backspaces over one data block of a file.

Buffer
: An intermediate storage area in which a block of input data is held until it is processed, or in which a block of output data is stored before transmission to an output unit.

Buffer Count
: In the $POOL control card, buffer count specifies the number of buffers to be assigned to a pool. In the $GROUP card, or in a Reserve or Internal Group Control Word, buffer count specifies the number of buffers to be reserved for use by the files in the group.

Buffer Control
: The control used by those IOCS commands that operate on all the words, or the remaining words, in a buffer.

Buffer Control Words
: The two words at the beginning of each buffer. Buffer Control Word 1 serves a dual purpose by chaining buffers together and keeping track of the contents of a buffer. Buffer Control Word 2 is a data channel command used to fill or empty the buffer.

Buffer Pool
: A group of equal-sized buffers that are related to the same two pool control words. The buffers in a pool are connected to permit sharing of buffers among a group of files.

Buffer Pool Control Words
: The two words at the beginning of each buffer pool that provide information about the status of buffers for use by the file control blocks.

Buffer Size
: The maximum number of data words that a buffer can hold. Buffer size determines the maximum number of words that can be read from an input block, and limits output blocks to that size.

Calling Sequence
: A sequence beginning with an instruction that transfers to an IOCS routine. The calling sequence contains one or more parameter statements that define the scope of the operation to be performed by the routine.

Checkpoint
: A reference point taken during execution of a program. Disk unit, drum unit, and tape positions, the status of machine registers and switches, and the contents of storage are recorded for the purpose of later restarting the program from exactly that point in execution.

Checkpoint File
: The file on which checkpoint records may be written. The checkpoint file should not be attached to a buffer pool, but must be opened. The file cannot be labeled.

Checkpoint Records
: Records written at periodic intervals by a program. The records contain the complete contents of core storage and all the machine registers and machine status conditions. These records may be used later to restart a program without requiring a complete rerun of the program.

CKPT (CKPT)
: The IOCS routine that prepares a checkpoint on the current checkpoint file.

Check Sum (folded)
: An 18-bit folded sum of the data words in a block. The check sum appears in positions S-17 of the block sequence word.

.CLOSE (CLOSE)
: The IOCS routine that terminates input/output activity on a file.

Command (IOCS)
Instructions used in IOCS to control "reading" and "writing" of data words from and to buffers. Their form is identical to IBM 7607 Data Channel Commands and their interpretation is similar.

Communications Region
A region at the beginning of IOCS in which transfer vectors to IOCS routines and a number of information-holding words are located.

Connected Buffer
A buffer currently in use by a file.

.COPY (COPY)
The IOCS routine that transfers data from an input file to an output file by appending a processed input buffer or buffers to the output file buffer chain.

Count Control
The control used by those IOCS commands that "read" or "write" a specified number of words.

Creation Date
The year and the day of the year on which a file was created. The creation date is contained within the label of a labeled file.

Current Buffer
The buffer in which words are already being "read" or "written," or are about to be "read" or "written." The connected buffer is the current buffer for a file.

Deblocking
The separation of a block of data into logical records or other segments.

DEFIN (DEFINE)
The IOCS routine that is used to impose the structure of a buffer pool upon a specified area in storage.

Disconnect (a buffer)
The removal of a buffer from further immediate access by the programmer. The current buffer is truncated and no further words can be "read" from or "written" into it.

Disk/Drum Limit
The ending track address on disk or drum.

Disk/Drum Record Holding Area
A buffer pool whose buffers contain the beta records taken from a disk or drum storage file.

End of Available Buffers
A condition under which IOCS is unable to provide a buffer for use by a file.

End-of-Buffer (*eob*) Switch
A programmed switch, specified in the .READ (READ) and .WRITE (WRITE) calling sequence, that controls the interpretation of nontransmitting IOCS commands. Under certain circumstances, *eob* is an address to which the program transfers when it reaches the end of a buffer.

End of File (*eof*)
The end of all records related to the same file. In a .READ (READ) calling sequence, an *eof* address is specified as the address to which the program will transfer when end of file is recognized.

End of Tape (*eot*)
The physical end of available tape on a reel. In a .WEF (WEF) calling sequence, *eot* is an address to which the program will transfer if an end-of-tape condition is recognized.

End of Reel Trailer Label
A label at the end of one reel of a multireel file.

End of File Trailer Label
The trailer label at the end of a file.

Error Exit (*err*)
An address specified in a .READ (READ) calling sequence to which the program will transfer upon recognition of an uncorrectable parity error, or a block sequence or check sum error.

External File
A file on an input/output unit, as contrasted with an internal file which is contained within core storage.

File
A collection of related information having its own logical beginning and end. IOCS routines are used to process records from or to specified files.

File Block
The area in storage that contains all of the file control blocks for a program.

File Block Origin
The location of the first word of the file block.

File Control Block
Twelve words in storage containing the control information and characteristics of a file to be processed by IOCS.

File List
A list of files included by the Full IOCS programmer in his source program.

Force-Sequential Processing
The condition in random processing in which each request for a beta record forces the processing of all such requests that have preceded it.

Full-Random Processing
The condition in random processing in which each request for a beta record is independent of other such requests.

Group (of beta records)
Mutually dependent beta records within a set.

Header Label
The label at the beginning of a file, or at the beginning of a segment of a file, that contains identifying information about the file and indicates the order in which the reels of a multireel file should be read.

History Record
A record of action present in the AC and MQ when an exit is taken from a .READ (READ) or .WRITE (WRITE) calling sequence. It provides the programmer with information about the effect of the last IOCS command executed.

IBJOB Processor
An integrated processor that consists of a group of programs. The components of the Processor can be used to translate programming languages, to assemble programs, and to load and execute compiled and assembled programs.

IBJOB Monitor
This is the supervisory portion of the IBJOB Processor. It provides communication between the System Monitor (IBSYS) and the components of the IBJOB Processor.

IBJOB Loader
This is the Loader that is a component of the IBJOB Processor. It loads program decks produced by the Macro Assembly Program and combines any required subroutines from the Subroutine Library with the program decks.

IBSYS
The symbolic name for the System Monitor.

Immediate File
A type of file in Full IOCS that is opened automatically when the file is attached to a buffer pool. When an immediate file is closed, it cannot be reopened.

Initialization
The term applied in Full IOCS to the two steps of attaching a file to a pool and opening the file.

Initialize
To set up an area in storage or to prepare key locations or status indicators prior to the actual beginning of a routine or subroutine.

Input/Output Unit
The physical device on which a recording medium is mounted. Allowable units in IOCS are magnetic tape, disk, drum, card reader, punch, or printer.

Internal File
A file that is contained entirely within a chain of buffers in core storage. Such a file has no external input/output unit associated with it.

Internal Group
A group of internal files. The group is preceded in the file list by an Internal Group Control Word (the SIX Word).

Internal Group Control Word
A word, with the prefix SIX, that precedes a group of internal files in a file list. It specifies the minimum number of buffers that must be available for the group, and the number of files in the group that will be open at the same time.

Intersystem Channel
A symbolic channel designation (J through Q) used to assign a file to a unit and then reserve that unit for later use in a different phase of a job.

Intersystem Reserve Unit
A logical input/output device that is given a reserve designation by one subsystem to indicate that the unit may not be returned to the availability chain until the reserve status is ended. It may be referenced later by another subsystem when called for, using the same reserve designation.

.JOIN (JOIN)
The IOCS routine that connects two separately defined buffer pools, thus forming a larger pool whose buffers are not necessarily contiguous.

Labels
Records at the beginning and end of a tape, disk, or drum file that contain control information for that file. Labels also identify segments of a file.

Labeled File
A tape, disk, or drum file, each segment of which begins with a header label and ends with a trailer label.

Label Identifier
The first field of a label which identifies it as a header label, or an end-of-reel or end-of-file trailer label.

Loader
See IBJOB Loader.

Locating
The use of IOCS nontransmitting commands to determine the location of data in an input buffer or to find space within an output buffer.

Logical Record
A record that is logically self-contained. The term is used in this publication to distinguish a data word from a physical record (block) which may contain several logical records.

Macro Assembly Program (MAP)
This is the assembly program within the IBJOB Processor. It assembles source programs written in the MAP language and assembles the generated MAP programs that are output from the FORTRAN IV and COBOL Compilers.

Mode
The form in which information is recorded; either binary or BCD.

Module Number
The number (0-9 for disk; 0, 2, 4, 6, or 8 for drum) used to designate a module within a disk or drum storage unit.

Multifile Reel
A reel of magnetic tape containing more than one file.

Multireel File
A magnetic tape file that is contained on more than one reel. IOCS can process labeled multireel files, and unlabeled multireel files that contain only one file mark on each reel.

Mutually Dependent Records
A group of beta records that must be delivered to the user's processing routine as a group.

Mutually Independent Records
Beta records that can be processed as each record of a set becomes available.

.MWR (MWR)
The IOCS entry point that gives control to (PROUT, the IOEX routine that prints messages on the System Printer.

Nondata Operation
An input/output operation that does not cause movement of data. These operations include: Rewind, Unload, Backspace Record, Backspace File, and Write End-of-File.

Nonstandard Label
A label that varies from the IBM standard label formats.

Nontransmitting Commands
IOCS commands that do not transmit data. Instead, they provide the location of data in an input buffer, locate space for data in an output buffer, or skip over data words in a buffer.

.OPEN (OPEN)
The IOCS routine that prepares a file for input/output activity.

Open Count
The number of files in a Reserve Group or Internal Group that will be open at the same time.

Parity Error
An error in transmission. It probably indicates that the file is being read in the wrong mode.

Partial Block Output File
A file subtype in Full IOCS that allows the "writing" of any number of machine words, regardless of buffer size.

Physical Record
A block of data; that is, a card, a line of print, or the data contained between two interrecord gaps on disk, drum, or magnetic tape.

Preparations for Processing
As used in this manual, this phrase encompasses all preparatory steps required or accomplished before a programmer can open files and begin processing data.

Preprocessor
That portion of Full IOCS that creates the file control blocks required by a program. It interprets control cards and serves as the IOCS monitor.

Proceed
The function performed at the completion of an IOCS command that allows the program to proceed to the next command in the command list.

Pseudo-Operation
A symbolic representation of information to an assembler or compiler.

Queue
A line of core storage words holding control information for beta record requests.

Random Processing
Processing using records taken from a random access storage device.

.RANCA (RANCAV)
A flag word used in Random IOCS.

.RANCL (RANCLS)
The close routine used in Random IOCS.

.RANDE (RANDEF)
The routine in Random IOCS used to be define disk/drum record holding areas and queue areas.

.RANFL (RANFLG)
A flag word in Random IOCS that indicates to the main program whether or not Random IOCS has a beta record, or records, ready for processing.

.RANRE (RANREQ)
The routine in Random IOCS used to secure desired beta records.

.RANRP (RANRPL)
The routine in Random IOCS used to replace the old beta record disk (or drum) address in a request with a new address. Its primary use is for chaining beta records.

Read
To transmit data from an input unit to an input buffer.

"Read"
To transmit data from an input buffer to working storage by using IOCS transmitting commands, or to locate or skip over data in an input buffer by using nontransmitting commands.

.READ (READ)
The IOCS routine that "reads" input data in the manner specified by IOCS commands.

Recording Medium
The medium on which a file is recorded, such as magnetic tape or disk storage.

Reel Sequence Number
A four-digit number (0000-9999) that indicates the order of this reel within a file.

Reel Serial Number
A five-character identification code assigned to a reel when it enters an installation. This number normally appears on the outer surface of a reel for visual identification.

Reserve File
A file that can be opened and closed repeatedly during a program. The file is listed within a Reserve Group in the file list.

Reserve Group
A group of reserve files. The group is preceded in the file list by a Reserve Group Control Word (the SVN Word).

Reserve Group Control Word
A word with the prefix SVN that precedes a group of reserve files in a file list. It specifies the minimum number of buffers that must be available for the group, and the number of files in the group that will be open at the same time.

Restart
To resume execution of a program from a reference point established by taking a checkpoint.

Retain (a buffer)
To keep a buffer connected for "reading" or "writing."

Retention Days
The number of days (0000-9999) after the creation date that a file is to be retained.

REW (REW)
The IOCS routine that can be used to rewind an unlabeled file.

Self-Loading Record
An input record in which the first word contains an IOCS command that can be used to read the record properly.

Set (of beta records)
All beta records requested by an alpha record in a single entry to Random IOCS.

Single-Reel File
An unlabeled file that is contained on a single reel of tape.

Skip
The process of bypassing data words in a buffer while "reading" or "writing." Skipping is accomplished by using nontransmitting IOCS commands.

Special Count Control
The control used by those IOCS commands that process by count unless terminated by an end-of-buffer condition.

Standard Origin
The next location following the end of the IOCS configuration specified.

Standard Label
A label that adheres to an IBM standard label format.

.STASH (STASH)
The IOCS routine that transfers data between an internal file and an external file or between internal files by reconnecting buffers from one file to another.

Subsystem
A system operating under the IBSYS System Monitor.

Symbolic Channel
A symbolic designation (S through Z) for a channel.

System Monitor
The over-all monitor for the IBSYS Operating System. It contains the routines and control information necessary for continuous system operation.

Terminate
To cease executing commands in a command list.

Total Block Output File
A file subtype in Full IOCS that requires that all words to be placed in a buffer be "written" using one command sequence.

Trailer Label
A record at the end of a labeled file, or at the end of each reel of a multireel labeled file.

Transmitting Commands
IOCS commands that move data between buffers and working storage.

Trap Supervisor, IOEX
That portion of the System Monitor that remains in storage at all times and receives control from each data channel trap or interrupt.

Truncate (a buffer)
To disconnect a buffer from use even though there are unused words in the buffer. When an input buffer is truncated, it returns to a chain of available buffers in the buffer pool. When an output buffer is truncated, it joins a chain of output buffers waiting to be written on an output unit.

Unbuffer
The process of adjusting the physical position of a tape to correspond to its logical position. For an output file, all buffers in use are truncated and a delay occurs until all writing ceases. For an input file, all buffers in use are returned to the pool and the tape is positioned to a point following the block corresponding to the buffer which contained the last word "read."

Unit Control Block (UCB)
A series of four-word blocks in IOEX. There is a unit control block for every input/output unit attached to the system.

Variable-Length Records
A series of records varying in length.

.WEF (WEF)
The IOCS routine that is used to write a file mark on an unlabeled output file.

WRITE
To transmit data from an output buffer to an input/output unit.

"WRITE"
To transmit data from working storage to an output buffer by using IOCS transmitting commands, or to locate empty space in an output buffer by using nontransmitting commands.

WRITE (WRITE)
The IOCS routine that "writes" output data in the manner specified by IOCS commands.

Index

Reader's Comments

IBM 7090/7094 IBSYS OPERATING SYSTEM
INPUT OUTPUT CONTROL SYSTEM
Form C28-6345-2

From

Name _____

Address _____

Your comments regarding the completeness, clarity, and accuracy of this publication will help us improve future editions. Please check the appropriate items below, add your comments, and mail.

	YES	NO
Does this publication meet the needs of you and your staff?	____	____
Is this publication clearly written?	____	____
Is the material properly arranged?	____	____

If the answer to any of these questions is "NO," be sure to elaborate.

How can we improve this publication? Please answer below.

☐ Suggested Addition (Page , Timing Chart, Drawing, Procedure, etc.)

☐ Suggested Deletion (Page)

☐ Error (Page)

COMMENTS:

No Postage Necessary if Mailed in U.S.A.

STAPLE

STAPLE

C28-6345-2

FOLD

FOLD

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P. O. BOX 390
POUGHKEEPSIE, N. Y. 12602

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS,
DEPARTMENT D91

CUT ALONG LINE

FOLD

FOLD

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601

9/64:5M-C

STAPLE