CRAY
RESEARCH, INC.

CRAY-1®
COMPUTER SYSTEMS

CRAY-OS VERSION 1
REFERENCE MANUAL

SR-0011

CRAY
RESEARCH, INC.

CRAY-1®
COMPUTER SYSTEMS

CRAY-OS VERSION 1
REFERENCE MANUAL

SR-0011

RECORD OF REVISION CRAY RESEARCH, INC. PUBLICATION NUMBER SR-0011

Each time this manual is revised and reprinted, all changes issued against the previous version in the form of change packets are incorporated into the new version and the new version is assigned an alphabetic level. Between reprints, changes may be issued against the current version in the form of change packets. Each change packet is assigned a numeric designator, starting with 01 for the first change packet of each revision level.

Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower righthand corner. Changes to part of a page are noted by a change bar along the margin of the page. A change bar in the margin opposite the page number indicates that the entire page is new; a dot in the same place indicates that information has been moved from one page to another, but has not otherwise changed.

Requests for copies of Cray Research, Inc. publications and comments about these publications should be directed to:

CRAY RESEARCH, INC.,
1440 Northland Drive,
Mendota Heights, Minnesota 55120

Revision	Description
	June 1976 - First printing
A	September 1976 - General technical changes; changes to JOB, MODE, RFL, and DMP statements; names of DS and RETURN changed to ASSIGN and RELEASE. STAGEI deleted, STAGEO replaced by DISPOSE. RECALL macro added and expansions provided for all logical I/O macros. RELEASE, DUMPDS, and LOADPDS renamed to DELETE, PDSDUMP, and PDSLOAD. Detailed description of BUILD added (formerly LIB). EDIT renamed to UPDATE.
B	February 1977 - Addition of Overlay Loader; deletion of Loader Tables (information now documented in CRI publication SR-0012); deletion of UPDATE (information now documented in CRI publication SR-0013); changes to reflect current implementation.
C	July 1977 - Addition of BKSPF, GETPOS, and POSITION logical I/O macros and $BKSPF, $GPOS, and $SPOS routines. Addition of random I/O. Changes to dataset structure, JOB, ASSIGN, MODE, and DUMP statements; BUILD; logical I/O and system action macro expansions. General technical changes to reflect current implementation.
C-01	January 1978 - Correction to DISPOSE and LDR control statement documentation, addition of description of $WWDS write routine, miscellaneous changes to bring documentation into agreement with January 1978 released version of the operating system.
D	February 1978 - Reprint with revision. This printing is exactly the same as revision C with the C-01 change packet added.
D-01	April 1978 - Change packet includes the addition of the ADJUST control statement; MODE and SWITCH macros; and PDD, ACCESS, SAVE DELETE, and ADJUST permanent dataset macros. Miscellaneous changes to bring documentation into agreement with released system, version 1.01.

Revision	Description
E	July 1978 - Represents a complete rewrite of this manual. Changes are not marked by change bars. New features for version 1.02 of the operating system that are documented in this revision include: addition of the MODIFY control statement and the DSP, SYSID, and DISPOSE macros; the addition of parameters to some control statements, the implementation of BUILD. The POSITION macro has been renamed SETPOS. Other changes to bring documentation into agreement with released version 1.02 of the operating system.
E-01	October 1978 - Change packet includes the implementation of ACQUIRE and COMPARE control statements; changes to the AUDIT and LDR control statements; changes to the MODE control statement and macro; the addition of control statement continuation, GETPARAM, and the GETMODE macro; and other minor changes to bring documentation into agreement with the released version 1.03 of the operating system.
F	December 1978 - Revision F is the same as revision E with change packet E-01 added. No additional changes have been made.
F-01	January 1979 - Change packet includes implementation of some features of BUILD; the addition of the BUFIN, BUFINP, BUFOUT, BUFOUTP, BUFEOF, and BUFEOD macros and other minor changes to bring documentation into agreement with the released version 1.04 of the operating system.
F-02	April 1979 - Change packet includes the implementation of the DEBUG, RERUN, and NORERUN control statements, the RERUN, NORERUN, and BUFCHECK macros; changes to DUMP, DSDUMP, AUDIT, and ASSIGN control statements; implementation of job rerun and memory resident datasets. Other minor changes were made to bring documentation into agreement with the released version 1.05 of the operating system.
G	July 1979 - Reprint with revision. This printing obsoletes all previous versions. Changes are marked with change bars. The changes bring this documentation into agreement with the released version 1.06 of the operating system.
G-01	December 1979 - Change packet includes the implementation of the WAIT and NOWAIT options on the DISPOSE control statement; the addition of a new DUMP format and CFT Linkage Macros; and other minor changes to bring documentation into agreement with the released version 1.07 of the operating system.

Revision	Description
H	January 1980 - Revision H is the same as revision G with change packet G-01 added. No additional changes have been made.
I	April 1980 - Revision I is a complete reprint of this manual. All changes are marked by change bars. New features for version 1.08 of the operating system that are documented in this revision include: the addition of the CALL and RETURN control statements, job classes, the NA parameter on permanent dataset management control statements, the NRLS parameter on the DISPOSE control statement and PDD macro, and the CW parameter on the COMPARE control statement. Changes to the LDR control statement include the addition of the LLD, NA, USA, and I parameters and the new selective load directives. New documentation has been added for unblocked I/O, including descriptions of the READU and WRITEU macros. Other new macros include ENDRPV, DUMPJOB and the debugging aids SNAP, DUMP, INPUT, OUTPUT, FREAD, FWRITE, UFREAD, UFWRITE, SAVEREGS, and LOADREGS. Documentation on CRAY-1 interactive capabilities and changes to reflect the CRAY-1 S Series have also been added. Other changes were made to bring documentation into agreement with released Version 1.08 of the operating system. With this revision, the publication number has been changed from 2240011 to SR-0011.
I-01	October 1980 - Change.packet includes the implementation of the IOAREA, SETRPV, ROLL, and INSFUN macros and the IOAREA control statement; the addition of execute-only datasets including adding the EXO parameter to the SAVE and MODIFY control statements and the PDD macro; the lengthening of the TEXT parameter field; the addition of the DEB parameter to the LDR control statement; and a change to the formats of the UFREAD and UFWRITE macros. The DEBUG option allowing conditional execution of the SNAP, DUMP, INPUT, and OUTPUT macros has been implemented. Other minor changes were made to bring documentation into agreement with the released version 1.09 of the operating system.

Revision Description

I-02 July 1981 - This change packet includes changes to Job Control Language syntax; the addition of JCL block control statements for procedure definition (PROC, ENDPROC, &DATA, and prototype statement), conditional processing (IF, ELSE, ELSEIF, and ENDIF), and iterative processing (LOOP, EXITLOOP, and ENDLOOP); the addition of ROLLJOB, SET, LIBRARY, ECHO, PRINT, FLODUMP, and SYSREF control statements; the addition of CSECHO macro; the addition of CNS parameter to CALL statement, REPLACE parameter to BUILD statement, ARGSIZE parameter to ENTER macro, KEEP parameter to EXIT macro, USE parameter to ARGADD macro; the addition of the two JCL tables JBI and JST. Other minor changes were made to bring the documentation into agreement with the released version of 1.10 of the operating system.

J February 1982 - Reprint. This reprint incorporates revision I with change packets I-01 and I-02. No other changes have been made.

J-01 June 1982 - This change packet includes the following additions: magnetic tape characteristics, temporary and local dataset clarification, mass storage permanent datasets, magnetic tape permanent datasets, tape I/O formats, interchange format, transparent format, new accounting information, **gn=nr* parameter, several CHARGES parameters, the OPTION control statement, procedure definition, HOLD parameter, new information to the ACCESS control statement, new tape dataset parameters, tape dataset conversion parameters, SUBMIT job control statement, PDSDUMP and PDSLOAD sample listings, SID parameter on the LDR control statement, new loader errors, relocatable overlays, CONTPRV macro, SUBMIT macro, unrecovered data error information, POSITION macro, new PDD macro parameters, the LDT macro, and new glossary terms. The information formerly in Appendix C is now in the COS EXEC/STP/CSP Internal Reference Manual, publication SM-0040. Other miscellaneous technical and editorial changes were made to bring the documentation into agreement with version 1.11 of the operating system.

K July 1982 - Reprint. This reprint incorporates revision J with change packet J-01. No other changes have been made.

PREFACE

This manual describes the external features of the CRAY-1 Operating System (COS). The manual consists of three parts:

PART 1 SYSTEM DESCRIPTION

This part describes the system components, storage of information on the CRAY-1, and job processing. An introduction to job control language is also included.

PART 2 JOB CONTROL LANGUAGE

In this part, the format of each COS control statement is given, along with an explanation of the function of each.

PART 3 MACRO INSTRUCTIONS

In part 3, CAL language macro instructions are described and in some cases examples are provided.

Other CRI publications that may be of interest to the reader are:

- CRAY-1 Hardware Reference Manual, publication 2240004
- CRAY-1 S Series Hardware Reference Manual, publication HR-0808
- CRAY-1 FORTRAN (CFT) Reference Manual, publication SR-0009

CONTENTS

PART 2 JCL CONTROL STATEMENTS

INTRODUCTION AND JOB DEFINITION (continued)

BUILD (continued)

PART 3 MACRO INSTRUCTIONS

2. SYSTEM ACTION REQUEST MACROS (continued)

APPENDIX SECTION

APPENDIX (continued)

PART 1

SYSTEM DESCRIPTION

CONTENTS
PART 1 SYSTEM DESCRIPTION

FIGURES

TABLES

INTRODUCTION 1

The CRAY-1 Operating System (COS) is a multiprogramming operating system for CRAY-1 Computer Systems. The *operating system* provides for efficient use of system resources by monitoring and controlling the flow of work presented to the system in the form of jobs. The operating system optimizes resource usage and resolves conflicts when more than one job is in need of resources.

COS is a collection of programs residing in CRAY-1 CPU memory or on system mass storage following *startup* of the system. (Startup is the process of bringing the CRAY-1 and the operating system to an operational state.)

Jobs are presented to the CRAY-1 by one or more computers referred to as *front-end computers* (also referred to as *stations* in Cray manuals). A front-end computer may be any of a variety of computer systems. Since a front-end computer system operates asynchronously under control of its own operating system, software execution on the front-end computer system is beyond the scope of this publication.

COS includes linkages providing for the initiation and control of interactive jobs and data transfers between the CRAY-1 and front-end terminals. These features are available only where supported by the front-end system.

The FORTRAN compiler (CFT), library routines, the CAL assembler, and the UPDATE source maintenance program are described in separate publications.

HARDWARE REQUIREMENTS

The CRAY-1 Operating System executes on the basic configuration of the CRAY-1 Computer System. A CRAY-1 models A, B, S/500 or S/1000 consists of a Central Processing Unit (CPU), a minicomputer-based Maintenance Control Unit (MCU), and a mass storage subsystem.

A CRAY-1 Model S/1200 through S/4400 consists of a CPU, an I/O Subsystem with a mass storage subsystem and an optional IBM-compatible tape subsystem.

COS operates with any of four central memory size options: one-half million, one million, two million, and four million words.

The mass storage for Models S/500 through S/1000 is a mass storage subsystem consisting of two or more disk storage units. The mass storage for Models S/1200 through S/4400 is conventionally composed of disk storage units on the I/O Subsystem but can optionally include a mass storage subsystem.

The I/O Subsystem consists of from two to four I/O processors and one-half, one, four, or eight million words of shared Buffer Memory. The optional tape subsystem is composed of at least one block multiplexer channel, one tape controller, and two tape units. The tape units supported are IBM-compatible 9-track, 200 ips, 1600/6250 bpi devices.

Figure 1-1 illustrates a basic system configuration. For more information about CRAY-1 hardware characteristics, refer to the CRAY-1 Hardware Reference Manual, Models A and B, publication HR-0004 and to the CRAY-1 S Series Hardware Reference Manual, publication HR-0808.

SYSTEM INITIALIZATION

COS is loaded into memory (*deadstarted*) and activated through a system startup procedure performed at the MCU or I/O Subsystem. At startup, permanent datasets are re-established on mass storage. (Permanent datasets survive deadstart; the user can always assume that they are present. See part 1, section 2 of this manual for more information on datasets.)

CENTRAL MEMORY ASSIGNMENT AND CHARACTERISTICS

Memory is shared by COS, jobs running on the CRAY-1, dataset I/O buffers, and system tables associated with those jobs. COS allocates resources to each job as needed as these resources become available. As a job progresses, information is transferred between memory and mass storage. These transfers can be initiated by either the job or by COS.

Figure 1-2 illustrates the assignment of memory to COS and to jobs.

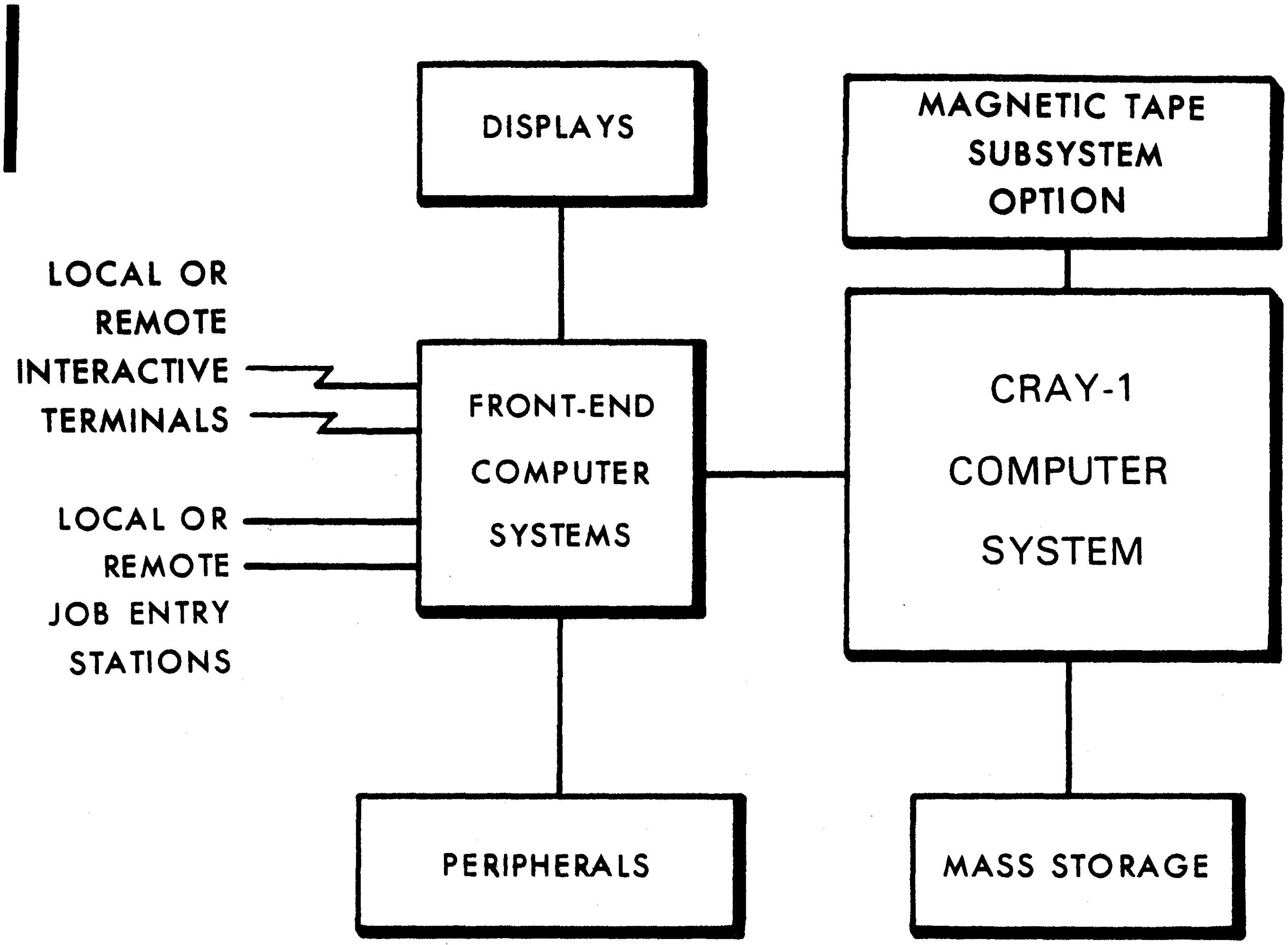

DISPLAYS
MAGNETIC TAPE SUBSYSTEM OPTION
LOCAL OR REMOTE INTERACTIVE TERMINALS
FRONT-END COMPUTER SYSTEMS
CRAY-1 COMPUTER SYSTEM
LOCAL OR REMOTE JOB ENTRY STATIONS
PERIPHERALS
MASS STORAGE

Figure 1-1. CRAY-1 system configuration

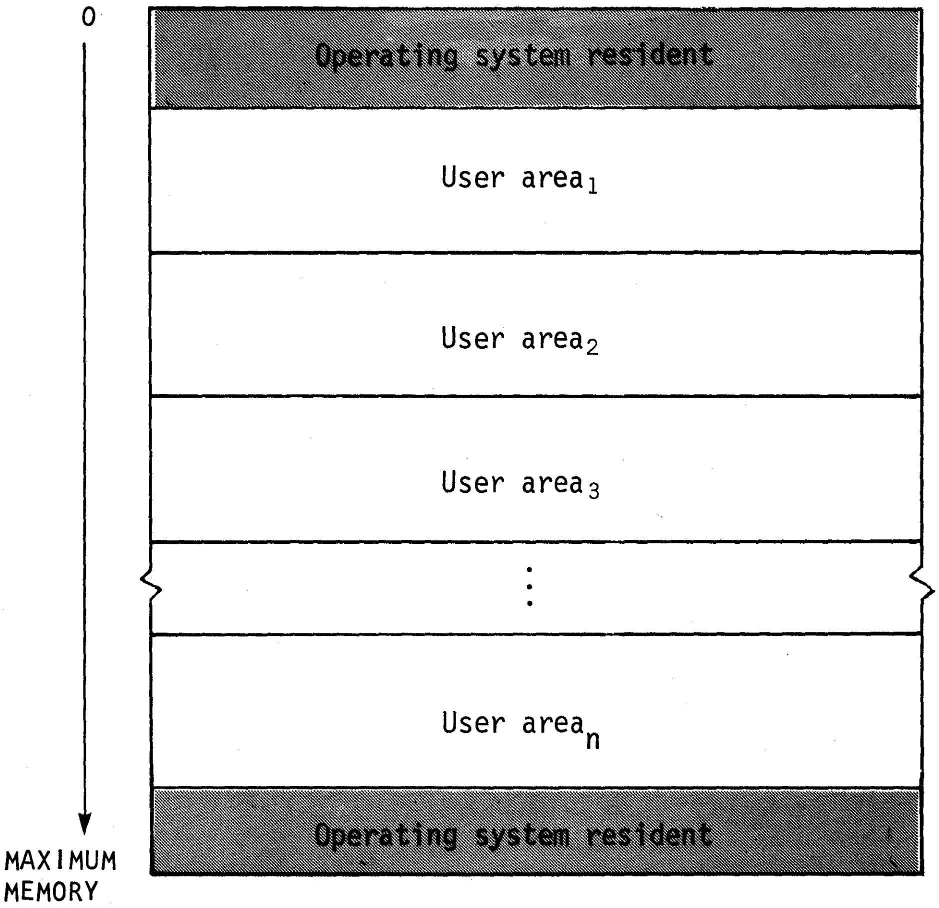


Figure 1-2. Memory assignment

MEMORY-RESIDENT COS

COS occupies two areas of memory. The memory resident portion of the operating system occupying lower memory consists of exchange packages, the System Executive (EXEC), the System Task Processor (STP), and the Control Statement Processor (CSP). The memory resident portion of the operating system occupying extreme upper memory contains station I/O buffers and space for the system log and dataset buffer.

USER AREA OF MEMORY

COS assigns every job a *user area* in memory. The user area consists of a Job Table Area (JTA) and a user field.

Job Table Area - JTA

For each job, the operating system maintains an area in memory that contains the parameters and information required for monitoring and managing the job. This area is called the Job Table Area (JTA). Each active job has a separate Job Table Area adjacent to the job's user field. The Job Table Area is not accessible to the user, although it may be dumped for analysis (see part 2, section 8).

User field

The *user field* for a job is a block of memory immediately following the job's JTA. The user field is always a multiple of 512 words. The beginning or *Base Address* (BA) and the end or *Limit Address* (LA) are set by the operating system. The Limit Address is specified by a parameter on one of the job control statements (see part 2) or by default. A user can request changes in field size during the course of a job.

Compilers, assemblers, system utility programs, and user programs are loaded from mass storage into the user field and are executed in response to control statements in the job deck. Each load and execution of a program may be referred to as a *job step*.

A detailed description of the contents of the user field is given in Appendix A. Briefly, however, the first 200_8 words of the user field are reserved for an operating system/job communication area known as the Job Communication Block (JCB). Programs are loaded starting at BA+200_8 and reside in the lower portion of the user field. The upper portion of the user field contains tables and dataset I/O buffers. The user field limit is equal to LA-1.

Memory addresses for instructions and operands are relative to BA. The CRAY-1 hardware adds the contents of BA to the address specified by a memory reference instruction to form an absolute address. A user cannot reference memory outside of the user field as defined by the BA and LA register contents; LA-1 is the user limit. (Refer to the CRAY-1 Hardware Reference Manual or to the CRAY-1 S Series Hardware Reference Manual for more information.)

MASS STORAGE CHARACTERISTICS

Mass storage for the CRAY-1/A and CRAY-1/B consists of one to thirty-two DD-19 or DD-29 Disk Storage Units (DSUs). Mass storage for CRAY-1 Models S/500 or S/1000 consists of two to thirty-two DD-29 DSUs. Mass storage for CRAY-1 Models S/1200 through S/4400 consists of two to forty-eight DD-29 DSUs, depending on the number of I/O Processors in the I/O Subsystem. These devices are physically non-removable.

All information maintained on mass storage by the CRAY-1 Operating System is organized into quantities of information known as *datasets*. In general, the user need not be concerned with the physical transfer of data between the disks and memory nor with the exact location and physical form in which datasets are maintained on mass storage. COS translates the user's logical requests for data input and output into disk controller functions automatically. For the orientation of the user the physical characteristics of disk storage units are summarized in table 1-1.

Table 1-1. Physical characteristics of disk storage units

Feature	DD-19	DD-29
Word capacity per drive	3.723×10^7	7.483×10^7
Word capacity per cylinder	92,160	92,160
Bit capacity per drive	2.424×10^9	4.789×10^9
Tracks per surface or cylinders per drive	411	823
Sectors per track	18	18
Bits per sector	32,768	32,768
Number of head groups	10	10
Latency (revolution time)	16.7 ms	16.7 ms
Access time	15 - 80 ms	15 - 80 ms
Data transfer rate (average bits per second)	35.4×10^6	35.4×10^6
Longest continuous transfer per request	92,160 words (1 cylinder)	92,160 words (1 cylinder)
Total bits that can be streamed to a unit (disk cylinder capacity)	5.9×10^6	5.9×10^6

Each disk storage unit contains a device label, datasets, and unused space to be allocated to datasets. The *device label* notes usable (unflawed) space on the disk unit and designates one of the devices as the Master Device. The *Master Device* is the *disk storage unit DSU* containing a table known as the *Dataset Catalog* (*DSC*), which contains information for maintaining permanent datasets.

To the user, mass storage *permanent datasets* are those datasets that may be assumed always present and available on mass storage. This permanence is achieved through techniques permitting the datasets noted in the DSC to be recovered or re-established in the event of system failures. Portions of COS, such as the loader, utility programs, the compiler, the assembler, and library maintenance and generation routines, reside in permanent datasets accessible by user jobs at any time.

Datasets containing job input decks and output from jobs already terminated also reside on mass storage, and because they are listed in the Dataset Catalog are regarded as permanent. This designation is somewhat misleading since their permanence is by definition rather than by tenure in the system. That is, the input dataset is permanent from the time it is staged from the front-end system to the CRAY-1 until the job terminates. Output datasets being disposed to a front end are permanent from job termination until the disposition is completed. The permanence of these system-defined datasets allows them to be recovered along with other permanent datasets after a system failure.

Any user job can create a mass storage permanent dataset that can be subsequently accessed, modified, or deleted by any other job producing the correct permission control words when attempting to associate it with a job. These permission control words are defined at the time the dataset is designated as permanent (that is, *saved*).

A permanent dataset ceases to exist when a user with the correct permission control word deletes it. This deletion notifies COS that the space occupied by the dataset is no longer permanent. However, the space is still reserved by the dataset until it is released by the user (see part 2 sections 3 and 5, respectively, for information on the RELEASE and DISPOSE control statements.)

In addition to the various permanent datasets, mass storage is used for temporary datasets. A *temporary dataset* is created by the job using it and remains temporary unless it is designated as permanent or disposed to a front end by the job. A temporary dataset neither saved as permanent nor disposed of is termed a *scratch dataset* and ceases to exist when the job terminates.

COS allocates space to datasets by sectors as space is needed. Storage assigned to a single dataset can be noncontiguous and can even be on multiple disk units. Default and maximum sizes for datasets are defined by system parameters. The user has limited control over the allocation of storage to a dataset through the ASSIGN control statement.

MAGNETIC TAPE CHARACTERISTICS

An I/O Subsystem can include an Auxiliary I/O Processor (XIOP) with the capability of addressing up to 16 block multiplexer channels of tape units. Each block multiplexer channel can be attached to IBM-compatible control units and tape units in a variety of configurations. The block multiplexer channels communicate with the control units and tape units to allow reading and writing data that can also be read and written on IBM-compatible CPUs.

Table 1-2. Physical characteristics of 200 ips, 9-track tape devices

Density (bits/inch)	Transfer rate (kilo bytes/sec)	Data/2400 ft. reel (mega bytes)	% of reel containing data	Block size (bytes)[§]
6250	1170	168	94	32768
1600	300	43	94	16384

[§] The block sizes in this table are used by the COS tape system for transparent-format tape datasets.

DATASETS 2

All information maintained by the CRAY-1 Operating System is organized into quantities of information known as *datasets*. Each dataset is identified by a symbolic name called a *dataset name* (*dn*). A dataset can be local to a job or permanent and available to the system and other jobs.

DATASET TYPES

Datasets are of two types: temporary and permanent.

TEMPORARY DATASETS

A *temporary dataset* is available only to the job that created it. Temporary datasets can be created in two ways: either explicitly by use of the ASSIGN control statement, or implicitly upon first reference to a dataset by name or unit number on an I/O request (CFT) or an OPEN macro call (CAL) (see part 3, section 2).

A temporary mass storage dataset is empty until written on. Rewind or backspace of the dataset is necessary before it can be read. A temporary dataset can be made permanent by use of the SAVE control statement. If the dataset is not made permanent, it will be released at job termination and its mass storage made available to the system.

LOCAL DATASETS

A dataset where a job has access is a *local dataset*. A local dataset can be temporary or permanent. Permanent datasets are made local with the ACCESS control statement or the ACCESS library subroutine.

Tape datasets can be made local to a job with the ACCESS control statement or the ACCESS library subroutine (described in the Library Reference Manual, CRI publication SR-0014). The device resource must also be specified on the JOB control statement.

MASS STORAGE PERMANENT DATASETS

A *permanent dataset* is available to the system and to other jobs and is maintained across system startups. Permanent datasets are of two types: those created by SAVE requests made by the user or front-end system (user permanent datasets), and input, output, or COS internal datasets (system permanent datasets).

User permanent datasets are maintained for as long as the user or installation desires. They are protected from unauthorized access by use of permission control words.

When a user permanent dataset is accessed via an ACCESS control statement (see part 2, section 4), it is treated as a local dataset by the job requesting access. However, it still exists as a permanent dataset on the system and may be used by other jobs unless unique access to that dataset was granted.

System permanent datasets relate to particular jobs or reflect the current operational state of COS. A job's *input dataset* is made permanent when the job is received by the CRAY-1 and is deleted when the job terminates. *Output datasets* local to the job can be disposed while the job is running or can be made permanent when the job terminates and then deleted from the CRAY-1 after being sent to the front-end system for processing. An example of a *COS internal dataset* is the system log.

MAGNETIC TAPE DATASETS

A magnetic tape dataset is available to any job declaring tape resource requirements on the JOB statement and specifying the appropriate information on its ACCESS request.

A magnetic tape dataset can be nonlabeled (NL), ANSI-labeled (AL), or IBM standard labeled (SL), and can be recorded or read at either 1600 or 6250 bits per inch (bpi). To gain access to an existing tape dataset for reading and/or rewriting, a volume identifier list, the correct file identifier (permanent dataset name), and the desired device type must be specified. The volume identifier list can consist of 1 to 255 volume identifiers. If the PDN is omitted from the ACCESS request, the local dataset name is used as the file identifier.

To gain access to a tape dataset for creating, the file identifier, desired device type, and the NEW parameter option must be specified. If no file identifier is present the local dataset name is used. If the volume identifier list is missing from the access request, it is called a *non-specific volume scratch*. A *specific volume scratch* occurs when the volume identifier list is present at the time of the access request. New tape datasets must be written to before a read is allowed.

Other options describing the tape dataset are available from the access request. Refer to the ACCESS control statement (part 2, section 4) for more details. Using other parameter options allows more efficient tape dataset descriptions.

COS automatically switches volumes during dataset processing and returns to the first volume of a multivolume dataset in response to a REWIND command. If a permanent write error occurs when trying to write a tape block for the user, COS automatically attempts to close the current volume and continues to the next volume.

The COS tape system uses Buffer Memory as a tape block buffering area so having a COS memory circular buffer as large as or larger than a tape block is unnecessary. This technique can result in significant memory savings whenever large tape blocks are being processed and in increased transfer rates whenever smaller blocks are being processed. The only real advantage in having a large COS buffer is a reduction in the packet traffic (overhead) in the tape subsystem. The smallest circular buffer for tape datasets is 512 words (inefficient) while a buffer size greater than 8192 words (16 sectors) results in little performance improvement.

EXECUTE-ONLY DATASETS

An *execute-only dataset* is a user permanent dataset for which all unauthorized forms of examination and modification are prohibited. An execute-only dataset is loaded by the Control Statement Processor (CSP) for execution. It differs in usage from other user permanent datasets in several ways:

- The accessor of the dataset cannot open the dataset for reading or writing.
- While an execute-only dataset is loaded in memory, no DUMPJOB requests are honored.
- The dataset cannot be staged via a DISPOSE request.
- The dataset must be loaded by a dataset name call rather than by the LDR control statement.
- The dataset cannot be dumped via PDSDUMP for archiving purposes.

Because execute-only is a dataset state rather than a permission mode, it is advisable to set at minimum a maintenance permission control word to disallow modification or deletion of the secure dataset.

A tape dataset cannot be made an execute-only dataset.

MEMORY-RESIDENT DATASETS

Some datasets may be specified by the user as memory resident datasets. A *memory resident dataset* is wholly contained within one buffer (see BS parameter on the ASSIGN control statement) and remains in memory at all times. Such a dataset ordinarily occupies no mass storage space. A memory resident dataset is normally a local dataset; however, a permanent dataset can sometimes be declared memory resident.

A dataset can be declared memory resident to reduce the number of I/O requests and disk blocks transferred. This is particularly useful for intermediate datasets not intended to be saved or disposed to another mainframe. In this case, all I/O performed on the dataset takes place in the dataset buffers in memory and the contents of the buffers are not ordinarily written to mass storage. Such a dataset cannot be made permanent, nor may it be disposed to another mainframe.

Normally, a memory resident dataset is empty until written on. If an existing dataset is declared memory resident, it is loaded when the first read occurs. A user attempting to write to a memory resident dataset must have write permission. However, as long as the buffer does not appear full, no actual write to mass storage ever occurs. Therefore, changes made to an existing dataset declared memory resident are not reflected on the mass storage copy of the dataset.

A memory resident dataset must be defined through an ASSIGN control statement containing the MR parameter or through an F$DNT call to the system. If the F$DNT call is used, the Dataset Definition List (DDL) supplied should specify DDMR=1. (See the description of the ASSIGN control statement or refer to the system calls in Appendix C for more information about formats.) In addition, the buffer size parameter should specify a buffer large enough to contain the entire dataset plus one block.

If at any time the system I/O routines are called to write to the dataset and the buffer appears to be full, the dataset ceases to be treated as memory resident, the buffer is flushed to mass storage, and all memory resident indicators for the dataset are cleared.

A magnetic tape dataset cannot be declared memory resident.

INTERACTIVE DATASETS

A dataset can be specified as interactive by a logged-on user provided that this feature is supported by the front end. Batch users cannot create interactive datasets. An interactive dataset differs from a local dataset in that a disk image of the dataset is not maintained. Instead, records are transmitted to and from a terminal attached to a front-end station. Record positioning (for example, REWIND or BACKSPACE) is not possible.

Interactive datasets can be created in two ways: by interactive users through the use of the ASSIGN control statement or through an F$DNT system call.

DATASET NAMING CONVENTIONS

The user assigns a symbolic name to each user dataset. This name, the *local dataset name*, is one through seven characters, the first of which can be A-Z, $, @, or %; remaining characters can also be numeric. However, a permanent dataset name does not have this restriction; all characters in a permanent dataset name may be alphanumeric. Certain language processors may place further restrictions on dataset names.

All datasets defined by the operating system are assigned names of the form $*dn*. Since datasets whose names begin with a $ may receive special handling by the system, the user should refrain from using this format when naming datasets.

DATASET FORMATS

Three dataset formats are supported for CRAY-1: blocked, interactive, and unblocked.

BLOCKED FORMAT

Blocked format is required for external types of datasets, such as user input and output datasets. The blocked format adds control words to the data to allow for processing of variable-length records and to allow for delimiting of levels of data within a dataset. Figure 2-1 illustrates the data hierarchy within a dataset. A blocked dataset can be composed of one or more files, which are, in turn, composed of one or more records.

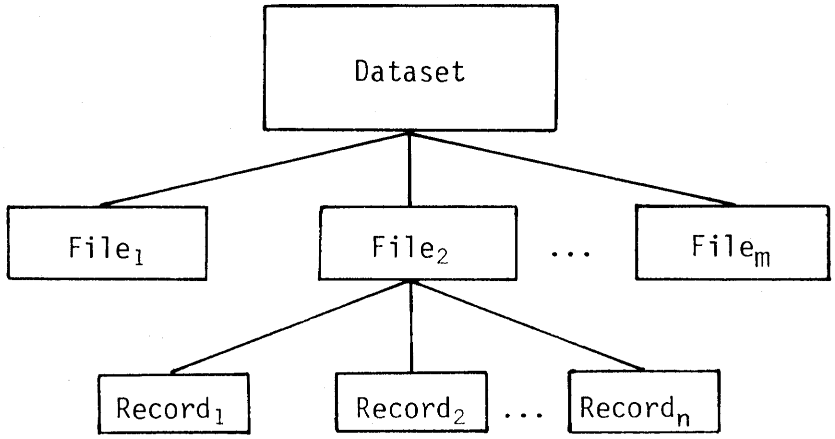


Figure 2-1. Data hierarchy within a dataset

The data in a blocked dataset can be either coded or binary. Each block consists of 512 words. There are two types of control words in a blocked dataset: block and record.

Block control word

The block control word (BCW) is the first word of every 512-word block. The format of a block control word is depicted in figure 2-2.

```
0       8       16       24       32       40       48       56     63
M|//////////////| |//////////////////////////|       BN        |  FWI
                 ↑
                BDF
```

Figure 2-2. Format of a block control word

Field	Bits	Description
M	0-3	Mode indicator (for block control word, M=0)
BDF	11	Bad data flag; indicates the following data, up to the next control word, is bad. This flag is set by the I/O Subsystem for magnetic tape datasets in interchange format.
BN	31-54	Block number. Designates the number of the current data block. The first block in a dataset is block 0.

Field	Bits	Description
FWI	55-63	Forward index. Designates the number of words (starting with 0) to the next record control word or block control word.

Record control word

A record control word (RCW) occurs at the end of each record, file, or dataset. The format of a record control word is illustrated in figure 2-3.

```
                 TRAN ──── BDF
0           8  /  / 16        24         32         40         48        56    63
┌───────────────────────────────────────────────────────────────────────────────┐
│ M|  UBC  |↓|↓|//////|             PFI          |            PFI      |  FWI   │
└───────────────────────────────────────────────────────────────────────────────┘
```

Figure 2-3. Format of a record control word

Field	Bits	Description
M	0-3	Mode indicator: 10_8 End-of-record 16_8 End-of-file 17_8 End-of-data Disregarding block control words occurring at 512-word intervals in a dataset, RCWs have the following logical relationship in a dataset. An end-of-record RCW immediately follows the data for the record it terminates. If the record is null, that is, if it contains no data, an end-of-record RCW can immediately follow an end-of-record or end-of-file RCW or can be the first word of the dataset. An end-of-file RCW immediately follows the end-of-record RCW for the final record in a file. If the file is null, that is, if it contains no records, the end-of-file RCW can immediately follow an end-of-file RCW or can be the first word of the dataset. An end-of-data RCW immediately follows the end-of-file RCW for the final file in the dataset. If the dataset is null, the end-of-data RCW can be the first word on the dataset.

Field	Bits	Description
UBC	4-9	Unused bit count. For end-of-record, UBC designates the number of unused low-order bits in the last data word of the record terminated by the end-of-record. For end-of-file and end-of-data RCWs, this field is 0. The data area protected by UBC must be zero-filled.
TRAN	10	Transparent record field. Used for an interactive output dataset only. If set, substitution of line feed for end-of-record RCWs is suppressed.
BDF	11	Bad data flag; indicates the following data, up to the next control word, is bad. This flag is set by the I/O Subsystem for magnetic tape datasets in interchange format. If flag is set, indicates when data was read from the device; an irrecoverable error was encountered in following data.
PFI	20-39	Previous file index. This field contains an index modulo 2**20 ($20,000,000_8$) to the beginning of the file. The index is relative to the current block such that if the beginning of the file is in the same block as this RCW, the PFI is 0.
PRI	40-54	Previous record (RCW) index. This field contains an index modulo 2**15 ($100,000_8$) to the block where the current record starts. The index is relative to the current block such that if the first word of data in this record is in the same block as this RCW, PRI is 0.
FWI	55-63	Forward word index. This field points to the next control word (RCW or BCW) and consists of a count of the number of data words up to the control word (that is, if the next word is an RCW or BCW, FWI is 0).

The typical dataset has many end-of-record RCWs per block. An example of dataset control words is illustrated in figure 2-4.

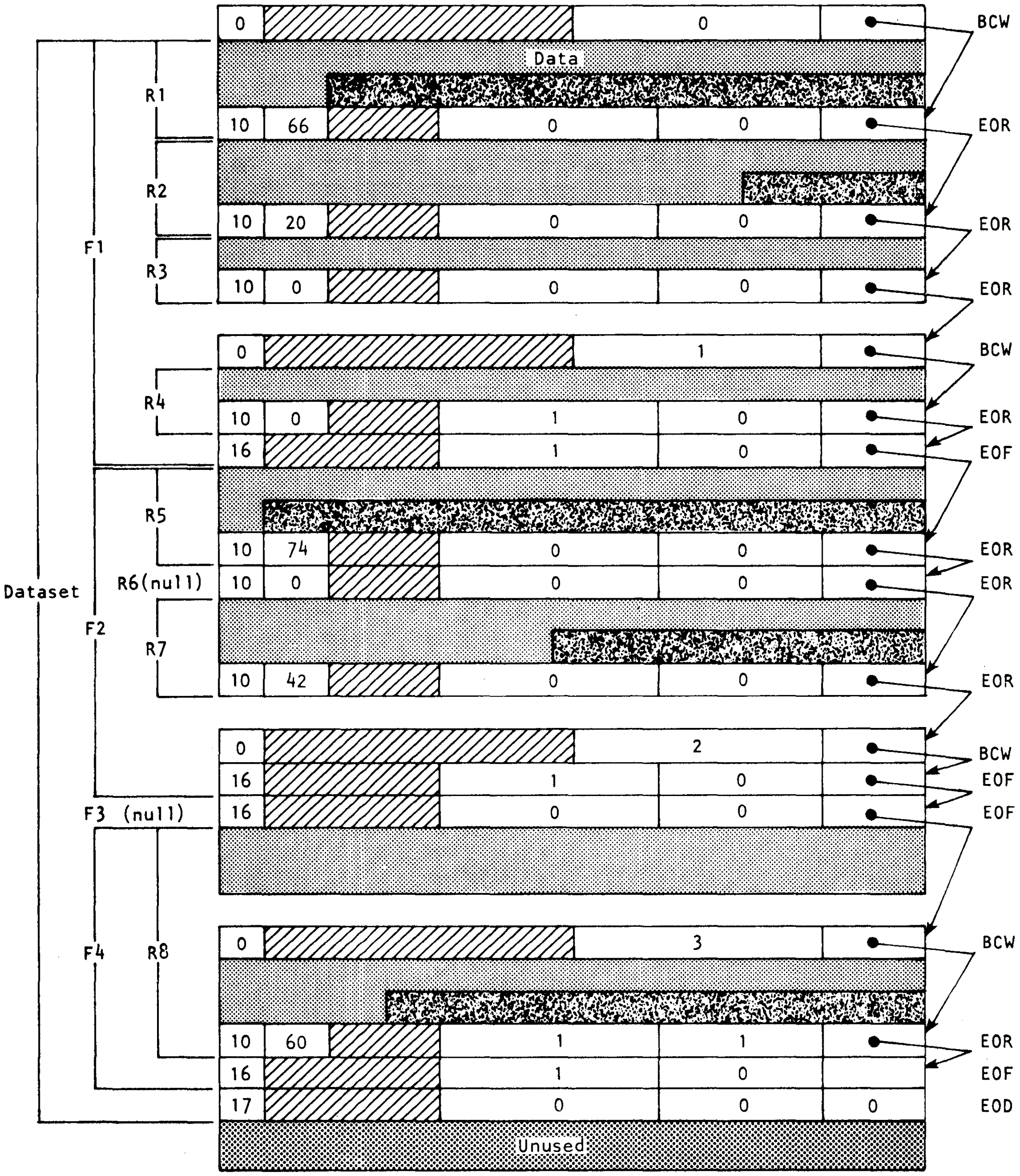


Figure 2-4. Example of dataset control words
(octal values shown)

Blank compression

Blank fields may be compressed for blocked coded files. Blank field compression is indicated by a blank field initiator code followed by a count. The default blank field initiator code is the installation parameter I@BFI which is either an ASCII code or 777_8 indicating that blank compression will not be done. Blank compression may be inhibited using an ASSIGN statement parameter or a F$DNT system call. A blank field of 3 through 96 characters is compressed to a 2-character field. The count is biased by 36_8; the actual count character is limited to $41_8 \leq$ *character count* $\leq 176_8$ (the ASCII graphics).

INTERACTIVE FORMAT

Interactive format closely resembles blocked format; however, each buffer begins with a block zero RCW. Each record transmitted to or from COS by an F$RDC or an F$WDC call must contain a single record consisting of a BCW, data, and an end-of-record RCW.

Two formats for interactive output can be assigned at creation time: character blocked and transparent. Character blocked mode is the default. In character blocked mode, an end-of-record RCW is interpreted as a line feed or a carriage return. In transparent mode, the end-of-record RCW is ignored and the user is responsible for supplying carriage control characters.

UNBLOCKED FORMAT

Dataset I/O can also be performed using unblocked datasets. Any dataset not in COS blocked format is considered unblocked. The data stream for unblocked datasets does not contain CRAY-1 Operating System RCWs or BCWs.

The system does not allocate buffers in the job's I/O buffer area for unblocked datasets; the user must specify an area for data transfer. When a read or write is performed on an unblocked dataset, the data goes directly to or from the user data area without passing through an I/O buffer. The word count of data to be transferred must be a multiple of 512.

Unblocked I/O cannot be performed on an interchange format tape dataset.

TAPE I/O FORMATS

Tape datasets are written and read on tape volumes. A *tape volume* is a reel of tape, also known as a section of the dataset (for example, in FSEC= on the ACCESS statement). Data is read or written in tape blocks. A *tape block* is a unit of data recorded on magnetic tape between two consecutive interblock gaps.

The size of tape blocks can vary up to a maximum of one million bytes.

Tape datasets can be read or written using two different formats: *interchange* or *transparent*. Tape datasets can be labeled or unlabeled.

INTERCHANGE FORMAT

Interchange format facilitates reading and writing tapes that are also to be read or written on other vendors' systems. In *interchange format* each tape block of data corresponds to a single logical record in COS blocked format.

In interchange format, tape block lengths can vary up to an installation-defined maximum of 1,048,576 bytes (131,072 64-bit words). It is recommended the maximum blocksize not exceed 100 to 200 kilobytes. Blocks exceeding this size may require special operational procedures (such as the use of special prepared tape volumes having an extended length of tape following the EOT reflective marker) and yield little increase in transfer rates or storage capacity.

When a tape dataset is read in interchange mode, physical tape blocks are represented in the user's I/O buffer with block control words (BCWs) and record control words (RCWs) added by COS. The data in each tape block is terminated by an RCW. The unused bit count field in the RCW indicates the amount of data in the last word of the tape block that is not valid data. A BCW is inserted before every 511 words of data, including the RCWs. The formats of RCWs and BCWs are described previously in this section. Figure 2-5 depicts a tape dataset in interchange mode. Tape blocks within tape label groups are not included in this format. The end of the dataset is represented by an end-of-file (EOF) RCW followed by an end-of-data (EOD) RCW.

When a tape dataset is written in interchange mode, the data must be in the I/O buffer in the user field in COS blocked format. The data in each logical record is written as a single tape block. BCWs and RCWs are not recorded on tape: block control words (BCWs) within a record are discarded; and the unused bits and terminating record control word (RCW) are also discarded. The unused bit count must be a multiple of 8. Tape datasets written in interchange mode must consist of a single file (single EOF RCW). Multiple-file tape datasets are not supported in interchange mode.

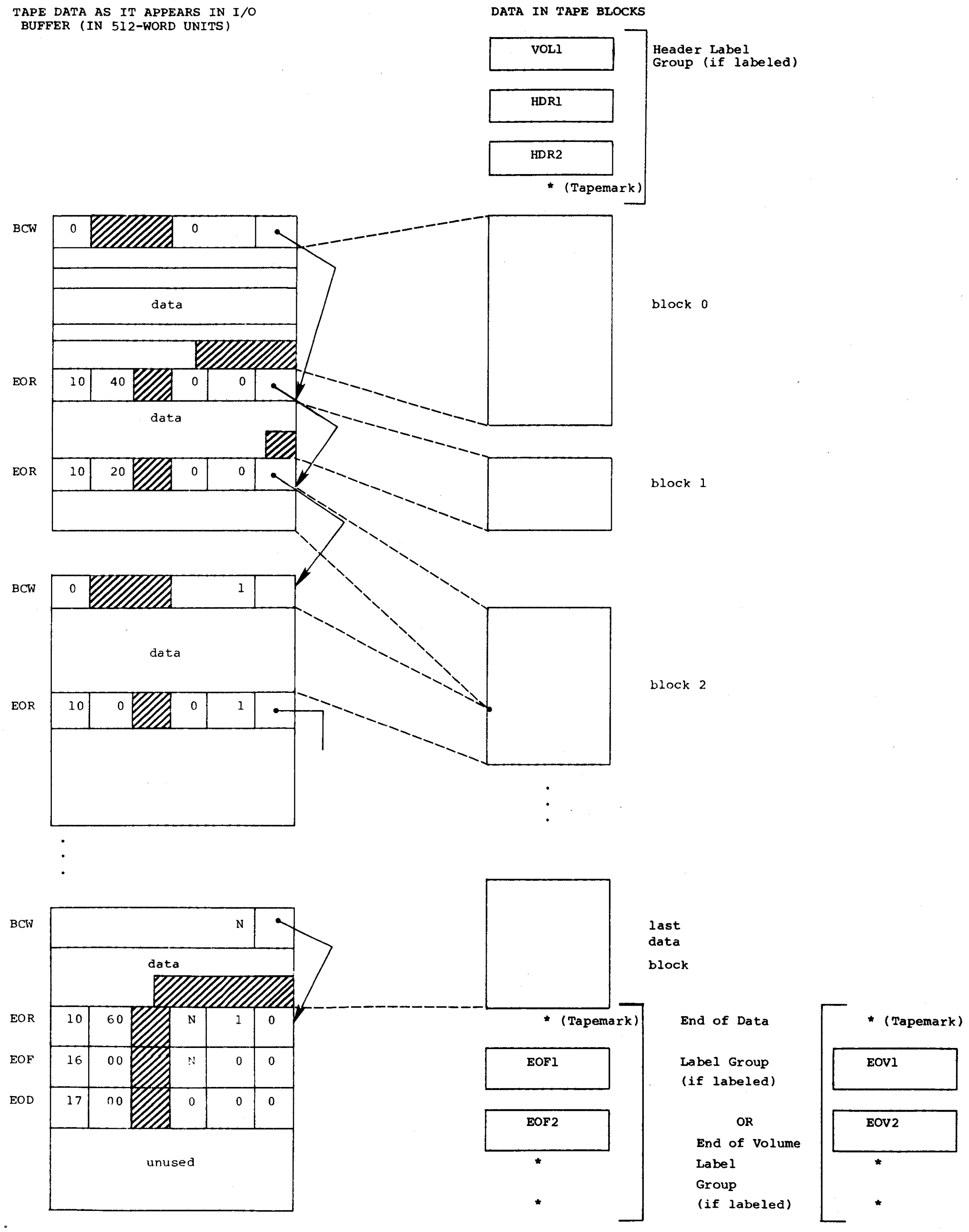


Figure 2-5. Interchange-format tape dataset (octal values shown)

TRANSPARENT FORMAT

In *transparent format* (disk image), each tape block is a fixed multiple of 4096 bytes (512 words) based on the dataset density (that is, 16384 bytes at 1600 BPI and 32768 bytes at 6250 BPI). The data in the tape block is transferred unaltered between the tape and the I/O buffer in the user field; no control words are added on reading or discarded on writing. In transparent mode, the data can be in COS blocked format or COS unblocked format.

USER LOGICAL I/O INTERFACES

When using logical I/O, the user is never directly concerned with the actual transfer of data between the devices and the system buffers. Figure 2-6 illustrates the relationship of different levels of user logical I/O interfaces and routines. Figure 2-6 summarizes the request levels and routine calls without going into detail on the movement of data between the system buffers and user program areas. For details, see Logical I/O Macros in part 3, section 3 of this publication.

The highest level of user interface is FORTRAN I/O statements; the lowest level is in the form of specially formatted requests called Exchange Processor requests.

FORTRAN statements fall into two categories: formatted/unformatted and buffered. The formatted/unformatted statements result in calls to library routines $RFI through $WUF. If the dataset is blocked, these routines call the logical record I/O routines. The logical record I/O routines perform blocking and deblocking. The logical record I/O routines communicate with COS through the Exchange Processor F$RDC and F$WDC requests.

If the dataset is unblocked, $RUA or $WUA calls the unblocked dataset routine $RLB or $WLB. These routines do no blocking or unblocking of data. The unblocked I/O routines communicate with the system through the F$RDC and F$WDC Exchange Processor calls.

Buffered I/O takes a different path from formatted/unformatted I/O. These routines interface (through an F$BIO Exchange Processor request) to routines in COS that normally perform logical I/O for system tasks. These routines, called TASK I/O or TIO, closely resemble the logical record I/O routines. TIO and the logical record I/O routines make similar requests of circular I/O routines in COS although the mechanism for making these requests is different.

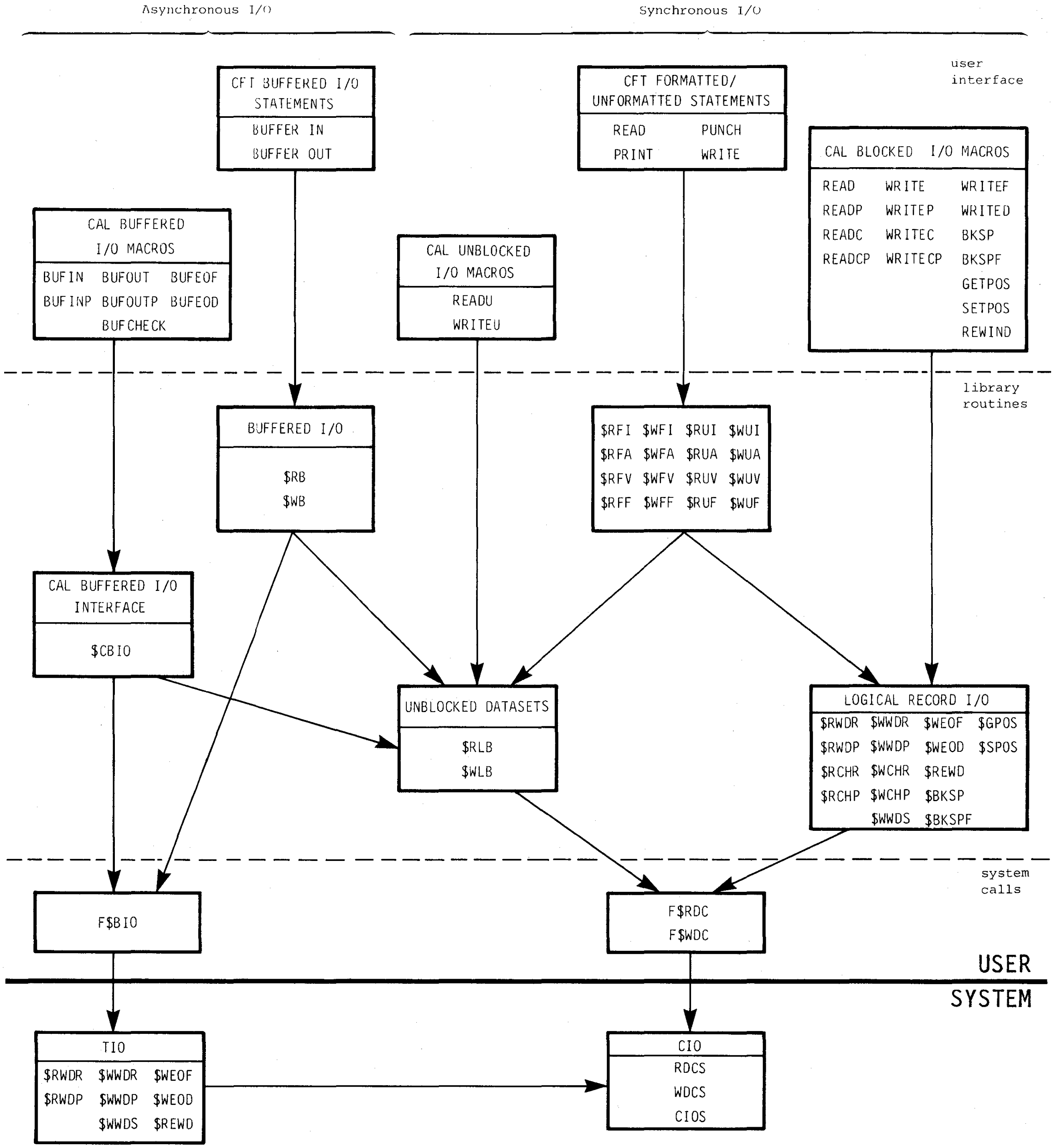


Figure 2-6. Relationship of levels of user I/O

Circular I/O routines (CIO) are the focal point for all logical I/O generated by COS. CIO communicates its needs for physical I/O to the Disk Queue Manager or Tape Queue Manager.

A FORTRAN buffered I/O request issued for an unblocked dataset results in the buffered I/O routines calling the unblocked dataset routines $RLB and $WLB, which then process these requests. These requests are processed the same as formatted/unformatted requests except that buffered I/O requests return control to the user after initiating I/O rather than waiting for completion of the I/O request. For a CAL buffered I/O request, $CBIO is called to route the request to either the blocked or unblocked I/O processing routines.

CRAY Assembly Language (CAL) I/O macros are described in part 3, section 3 of this manual. Logical Record I/O routines and FORTRAN I/O routines are described in Appendix D of this manual. Refer to the FORTRAN (CFT) Reference Manual, CRI publication SR-0009, for a description of FORTRAN statements.

DATASET DISPOSITION CODES

Each dataset is assigned a *disposition code* telling the operating system the disposition to be made of the dataset when the job is terminated or the dataset is released. The disposition code is one of the parameters of the DISPOSE and ASSIGN control statements (see part 2).

Each disposition code is a 2-character alpha code describing the destination of the dataset. The default disposition code for a dataset is SC (scratch) when a dataset is opened, unless the dataset is named $OUT. By default, COS assigns the disposition code PR (print) to $OUT when the dataset is created. No DISPOSE statement is required for $OUT; it is automatically routed back to the designated mainframe to be printed on a front-end designated printer.

DECK STRUCTURE AND JOB PROCESSING 3

A *job* is a unit of work submitted to the CRAY-1 computing system. It consists of one or more files of card images contained in a *job deck dataset*. Each job passes through several stages from job entry through job termination.

JOB DECK STRUCTURE

A job originates as a card deck (or its equivalent) at a front-end computer system. Card images in the job deck dataset are organized into one or more files. Figure 3-1 illustrates a typical job deck consisting of a control statement file, a source file, and a data file. (The physical card forms for *end-of-file* and *end-of-data* are defined by the front-end system.)

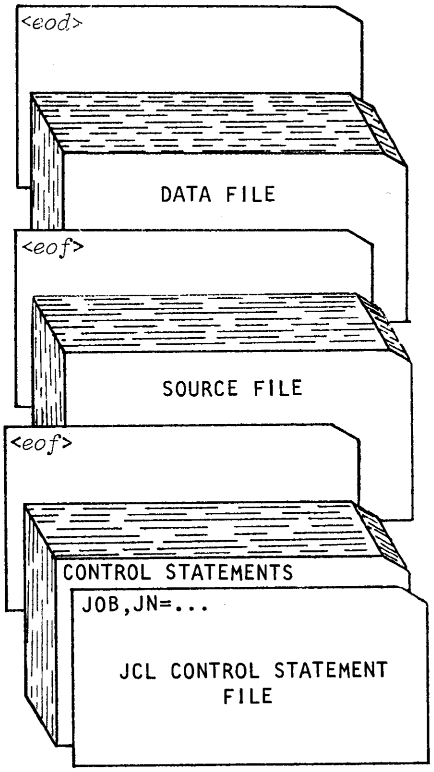


Figure 3-1. Basic job deck

The first (or only) file of the job deck must contain the job control language (JCL) control statements that specify the job processing requirements. Each job begins with a JOB statement, identifying the job to the system. If accounting is mandatory in the user's system, the ACCOUNT statement must immediately follow the JOB statement. All other control statements follow the JOB statement. Control statements may also be grouped into control statement blocks as decribed in part 2, section 2. The end of the control statement file is designated by an end-of-file (or an end-of-data if the job consists of a control statement file only).

Files following the control statement file may contain source code or data. These files are handled according to instructions given in the control statement file.

The final card in a job deck must be an *end-of-data*.

GENERAL DESCRIPTION OF JOB FLOW

A job passes through the following stages from the time it is read by the front-end computer system until it completes:

- Entry
- Initiation
- Advancement
- Termination

JOB ENTRY

A job can enter the system in the form of a job deck submitted to a front-end computer system or a local or remote job entry station. The job is transferred to CRAY-1 mass storage, where it resides until it is scheduled to begin processing. An entry is made in the system tables for the job thus making the job input dataset permanent until it is deleted at the completion of the job.

JOB INITIATION

The operating system examines the parameters on the JOB control statement to determine the resources needed. When system resources required for initiation are available, the job is initiated (scheduled to begin processing).

Initiation of a job includes preparing a Job Table Area (JTA) and user field, positioning the input dataset for the first job step, and placing the job in a waiting queue for the CPU.

When the CRAY-1 Operating System (COS) schedules the job for processing, it creates four datasets: $CS, $IN, $OUT, and $LOG.

- $CS is a copy of the job's control statement file from $IN and is used only by the system; the user cannot access $CS by name. This dataset is used to read job control statements. The disposition code for $CS is SC (scratch).

- $IN is the job input dataset. It is identified at the front-end computer system by a dataset name assigned by the user. The job itself can access the input dataset, with read only permission, by its local name, $IN, or as FORTRAN unit 5.

- $OUT is the job output dataset. The job can access this dataset by name or as FORTRAN unit 6. The disposition code for $OUT is PR (print).

- The job's logfile ($LOG) contains a history of the job. This dataset is known only to the operating system and is not accessible by the user. User messages can be added to the job's logfile with the MESSAGE system action request macro (see part 3) or the REMARK, REMARK2, or REMARKF subroutines in $FTLIB.

JOB ADVANCEMENT

Job advancement is the processing of a job according to the instructions in a control statement file. Advancement occurs as a normal advance or as an abort advance.

A normal advance causes COS to interpret the next control statement in the job's control statement file.

An abort advance occurs if the operating system detects an error or if the user requests that the job abort. An abort advance causes the operating system to search for and interpret the first control statement following the next valid EXIT control statement in the control statement file. EXIT statements that are within control statement blocks (in-line procedure, conditional, or iterative) that have not yet been invoked are ignored during the search for the next EXIT statement.

If the block currently being processed is a conditional block, only the group of control statements preceding the next conditional statement in the block is evaluated. For example, in the following sample control statement sequence, an abort advance occurs at the control statement THIS IS A JOB STEP ABORT CONDITION because it does not begin with a valid verb. Control statement interpretation resumes with the control statement: *. RESUME HERE. The EXIT statements that are included in the conditional block are ignored because they reside in blocks that are not executed.

```
.
.
.
SET,J1=0.
IF(J1.EQ.0)
 .
 .
 .
 THIS IS A JOB STEP ABORT CONDITION.
ELSEIF (J1.EQ.1)
  .
  .
  .
  EXIT.
ELSE.
  EXIT.
ENDIF.
  .
  .
  .
EXIT.
*.  RESUME HERE
  .
  .
```

JOB TERMINATION

Output from a job is placed on system mass storage. At completion of a job, the operating system appends $LOG to $OUT and makes $OUT permanent. $IN, $CS, and $LOG are released. $OUT is renamed *jobname* (from the JN parameter value of the JOB control statement) and is directed to the output queue for staging to the specified front-end computer system. When the front end has received the entire contents of $OUT, the system table entries for the dataset are deleted, and the output dataset itself is deleted from CRAY-1 mass storage.

The front-end computer processes $OUT as specified by the dataset disposition code.

If, for any reason, $OUT does not exist, $LOG is the only output returned at job termination.

JOB RERUN

Under certain circumstances, restarting of a job from its beginning may become necessary or desirable. This is referred to as rerunning a job. Conditions causing the system to attempt to rerun a job are:

- Operator command

- Uncorrectable memory error

- Uncorrectable error reading the mass storage image of a job that has been rolled out. Rolling out occurs because of system or user initiation.

- System restart

A user job may perform certain functions that normally make rerunning of a job impossible. These functions are considered nonrerunnable because they produce results that might cause the job to run differently if it were rerun. These functions include:

- Writing to a permanent dataset

- Saving, deleting, adjusting, or modifying a permanent dataset

- Acquiring a dataset from a front-end system

Ordinarily, when a job becomes nonrerunnable, it remains so. However, the user may specify in the program that the job is rerunnable. The user should do this only when changes in job results due to execution of nonrerunnable functions are acceptable. COS never makes a job rerunnable automatically.

The user may also override system monitoring of a job rerunnability, regardless of what functions the job performs. This ordinarily is done only if the job is structured to run correctly regardless of whether nonrerunnable functions are performed.

REPRIEVE PROCESSING

Normally, when a job step abort error occurs, control passes to the EXIT control statement and exit processing begins. Reprieve processing, however, allows the user to attempt recovery from many of the job step abort errors or to perform clean-up functions before continuing with the abort.

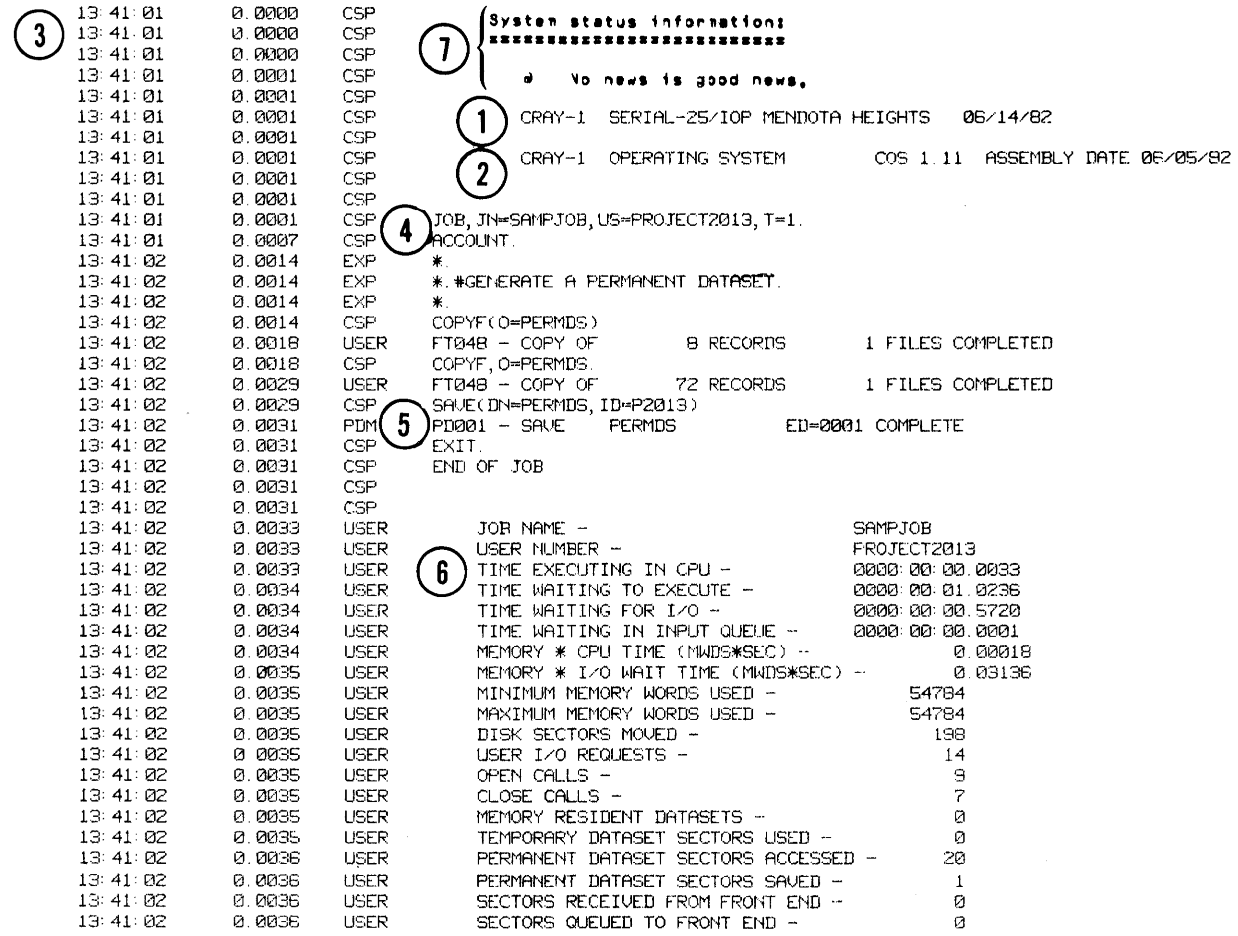
Reprieve processing may also be used during the normal termination of a job step. In this case, control transfers to the user's reprieve code instead of to the next normal job step.

Two types of error conditions are related to a job step: non-fatal and fatal. Non-fatal error conditions may be reprieved any number of times per job step by the user. Fatal error conditions can be reprieved only once for each type per job step.

When requesting reprieve processing, the user selects the error conditions to be reprieved by setting a mask in the SETRPV subroutine or macro call. If a selected error condition occurs during job processing, the user's current job step maintains control. The user's exchange package, vector mask register, error code, and error class are saved and control passes to the user's reprieve code. (Refer to the F$RPV processing description in Appendix C and to Appendix F for error codes. Also, see description of SETRPV macro for mask values.)

JOB LOGFILE AND ACCOUNTING INFORMATION

For each job run, the system produces a logfile--an abbreviated history of the progress of the job through the system. The logfile for a job appears at the end of the job output and consists of a list of comments. Each job control statement is listed sequentially, followed by any messages associated with the job step. Clock time, accumulated CPU time, and COS information are also given for each job step. A logfile usually consists of the items illustrated in figure 3-2. Item 6 illustrates the accounting information given to the user.



```
13:41:01    0.0000   CSP    System status information:
13:41:01    0.0000   CSP    **************************
13:41:01    0.0000   CSP
13:41:01    0.0001   CSP      @   No news is good news.
13:41:01    0.0001   CSP
13:41:01    0.0001   CSP      CRAY-1  SERIAL-25/IOP MENDOTA HEIGHTS   06/14/82
13:41:01    0.0001   CSP
13:41:01    0.0001   CSP      CRAY-1  OPERATING SYSTEM        COS 1.11  ASSEMBLY DATE 06/05/82
13:41:01    0.0001   CSP
13:41:01    0.0001   CSP
13:41:01    0.0001   CSP    JOB,JN=SAMPJOB,US=PROJECT2013,T=1.
13:41:01    0.0007   CSP    ACCOUNT.
13:41:02    0.0014   EXP    *.
13:41:02    0.0014   EXP    *.#GENERATE A PERMANENT DATASET.
13:41:02    0.0014   EXP    *.
13:41:02    0.0014   CSP    COPYF(O=PERMDS)
13:41:02    0.0018   USER   FT048 - COPY OF         8 RECORDS       1 FILES COMPLETED
13:41:02    0.0018   CSP    COPYF,O=PERMDS.
13:41:02    0.0029   USER   FT048 - COPY OF        72 RECORDS       1 FILES COMPLETED
13:41:02    0.0029   CSP    SAVE(DN=PERMDS,ID=P2013)
13:41:02    0.0031   PDM    PD001 - SAVE     PERMDS            ED=0001 COMPLETE
13:41:02    0.0031   CSP    EXIT.
13:41:02    0.0031   CSP    END OF JOB
13:41:02    0.0031   CSP
13:41:02    0.0031   CSP
13:41:02    0.0033   USER       JOB NAME -                             SAMPJOB
13:41:02    0.0033   USER       USER NUMBER -                          PROJECT2013
13:41:02    0.0033   USER       TIME EXECUTING IN CPU -                0000:00:00.0033
13:41:02    0.0034   USER       TIME WAITING TO EXECUTE -              0000:00:01.0236
13:41:02    0.0034   USER       TIME WAITING FOR I/O -                 0000:00:00.5720
13:41:02    0.0034   USER       TIME WAITING IN INPUT QUEUE -          0000:00:00.0001
13:41:02    0.0034   USER       MEMORY * CPU TIME (MWDS*SEC) -               0.00018
13:41:02    0.0035   USER       MEMORY * I/O WAIT TIME (MWDS*SEC) -          0.03136
13:41:02    0.0035   USER       MINIMUM MEMORY WORDS USED -              54784
13:41:02    0.0035   USER       MAXIMUM MEMORY WORDS USED -              54784
13:41:02    0.0035   USER       DISK SECTORS MOVED -                       198
13:41:02    0.0035   USER       USER I/O REQUESTS -                         14
13:41:02    0.0035   USER       OPEN CALLS -                                 9
13:41:02    0.0035   USER       CLOSE CALLS -                                7
13:41:02    0.0035   USER       MEMORY RESIDENT DATASETS -                   0
13:41:02    0.0035   USER       TEMPORARY DATASET SECTORS USED -             0
13:41:02    0.0036   USER       PERMANENT DATASET SECTORS ACCESSED -        20
13:41:02    0.0036   USER       PERMANENT DATASET SECTORS SAVED -            1
13:41:02    0.0036   USER       SECTORS RECEIVED FROM FRONT END -            0
13:41:02    0.0036   USER       SECTORS QUEUED TO FRONT END -                0
```

Figure 3-2. Example of a job logfile

(1) <u>First header line</u>: Installation-defined message, usually identifying the site and date the job was run.

(2) <u>Second header line</u>: Installation-defined message, usually identifying the operating system, its current revision level, and the date of the last revision.

(3) <u>Columns</u>: The leftmost column identifies the wallclock time for each job step and the middle column identifies the accumulated CPU time for the job. The rightmost column identifies a system module or the user as the originator of the message. All times are in decimal. Entries commonly noted include the following:

CSP	Control Statement Processor
PDM	Permanent Dataset Manager
EXP	Exchange Processor
ABORT	Abort Message
USER	Program in user field

(4) Control statements: Control statements are listed in the logfile as they are processed if requested with the ECHO statement described in part 2, section 1. When the job terminates, the last control statement processed is the last control statement printed. Control statements are not listed if JCL message class is disabled.

(5) Logfile messages: Any messages related to control statement processing are shown below the statement.

(6) Accounting information: When a job reaches completion, COS writes a summary of basic accounting data onto the logfile for the job. All times given are in hours, minutes, and seconds (to the nearest ten-thousandth of a second). The following accounting information is provided (in decimal):

- Job name and user number
- CPU time used by the job
- Time waiting to execute
- Time waiting for I/O
- Time waiting in input queue
- Memory usage based on the execution and I/O wait time in million word-seconds
- Minimum and maximum number of memory words used
- Number of 512-word disk blocks (sectors) moved
- Number of user I/O requests made by the job
- Open and close calls
- Memory-resident datasets
- Number of 512-word disk blocks (sectors) used for temporary datasets
- Number of 512-word disk blocks (sectors) accessed and saved for permanent datasets
- Number of 512-word disk blocks (sectors) received from and queued to the front end
- Number of tape devices reserved; message issued only if magnetic tape datasets have been processed.

- Number of tape volumes mounted; message issued only if magnetic tape datasets have been processed.

- Amount of tape data moved, expressed as a multiple of 512 words; message issued only if magnetic tape datasets have been processed. Each CRAY-1 disk sector consists of 512 words, and in COS blocked format each block consists of 512 words.

- Number of tape blocks moved; message issued only if magnetic tape datasets have been processed.

(7) System Bulletin: The system bulletin allows the installation to print messages in the logfile, usually about the status of the system environment. It is an installation-maintained message dataset.

JOB CONTROL LANGUAGE 4

The job control language of the CRAY-1 Operating System (COS) allows the user to present a job to the CRAY-1, define and control execution of programs within the job, and manipulate datasets associated with a job.

The job control language is composed of *control statements* with each control statement containing information for a job step. COS initially creates a *control statement dataset*, $CS, to hold job control statements. Additional control statement datasets can be created via procedure definition (part 2, section 2) or the CALL control statement (part 2, section 1).

All control statements must adhere to a set of general syntax rules.

The syntax of a control statement is:

verb	sep_1	$param_1$	sep_2	$param_2$	...	sep_n	$param_n$	*term*	*comments*

Every control statement consists of a verb and a terminator (*term*) as a minimum, except for the comment control statement (*) which does not require a terminator. Additionally, most control statements require parameters ($param_i$) and separators (sep_i) between the verb and the terminator. The maximum number of parameters (zero, one, or more) depends on the verb.

The continuation separator (the caret symbol) allows a control statement to consist of more than one line image (80 characters). The JOB, ACCOUNT, DUMPJOB, EXIT, and comment control statements cannot be continued. All other control statements may have any number of continuation card images, subject to restriction by the verb. A caret occurring within a literal string has no special significance.

A *comment* is an optional annotation to a control statement and can be a string of any ASCII graphic characters. The comment follows the line image terminator. The control statement interpreter ignores comments. All comments appear in the logfile.

Blanks are ignored unless they are embedded in a literal string. Blanks cannot precede the verb on the JOB control statement.

SYNTAX VIOLATIONS

COS notes syntax violations in the system and user logfiles. If the JOB control statement is in error, processing of the job terminates immediately. If accounting is mandatory, ACCOUNT statement errors also cause job termination. All other syntax errors cause a *job step abort* condition, which causes the system to search for an EXIT control statement. A successful search resumes control statement processing with the job step following EXIT. If no such job step exists or if an EXIT statement is not found, the job is terminated. Job step abort may also direct control to a user-specified routine (see description of Reprieve processing in part 1, section 3 of this publication).

VERBS

A *control statement verb* is the first nonblank field of a control statement specifying the action to be taken by COS during control statement processing. COS recognizes three types of control statement verbs: *system verbs*, *dataset name verbs* (*local* and *system*), and *library-defined verbs*. A control statement verb cannot be continued across a card boundary.

SYSTEM VERBS

A system verb consists of an alphabetic character which can be followed by one through six alphanumeric characters.[§] The verb requests that COS perform the indicated function. The system verbs are:

- *
- ACCESS
- ACCOUNT
- ACQUIRE
- ADJUST
- ASSIGN
- CALL
- DELETE
- DISPOSE
- ECHO
- ELSE
- ELSEIF
- ENDIF
- ENDLOOP
- ENDPROC
- EXIT
- EXITLOOP
- IF
- IOAREA
- LIBRARY
- LOOP
- MODE
- MODIFY
- NORERUN
- OPTION
- PRINT
- PROC
- RELEASE
- RERUN
- REWIND
- RETURN
- RFL
- ROLLJOB
- SAVE
- SET
- SIMABORT
- SUBMIT
- SWITCH

[§] Alphabetic characters include $, %, @, and the 26 uppercase letters A through Z. Alphanumeric characters include all the alphabetic characters and the digits 0 through 9.

LOCAL DATASET NAME VERBS

A verb that is the name of a local dataset consists of an alphabetic character followed by one through six alphanumeric characters.§ This verb requests that COS load and execute an absolute binary program from the first record of the named dataset. If the user job has a dataset with the indicated name, COS loads and executes the program from that dataset.

LIBRARY-DEFINED VERBS

A library-defined verb consists of one through eight characters. The library-defined verb is either a program§§ or procedure definition residing in a library that is a part of the current *library searchlist.* (The library searchlist is the order in which the content of the library is searched by COS. This order may be specified with the LIBRARY statement described in part 2, section 1.) A program in a library is an absolute binary program to be loaded and executed. A procedure definition is a group of control statements and/or data to be processed (see part 2, section 2).

SYSTEM DATASET NAME VERBS

COS searches for a verb that is the name of a system-defined dataset in the System Directory Table (SDR). A system-defined dataset name verb consists of an alphabetic character which can be followed by one through six alphanumeric characters.§ The System Directory Table is a list of common language processors and utilities known to the system and made available to users at startup. The name of the program (for example, CAL, CFT, or DUMP) is also the name of the dataset containing the absolute binary of the program.

VERB SEARCH ORDER

When COS encounters a verb in a control statement file, it searches for a match to that verb in the following order:

1. System verbs
2. Local dataset name verbs
3. Library-defined verbs
4. System dataset name verbs

§ Alphabetic characters include $, %, @, and the 26 uppercase letters A through Z. Alphanumeric characters include all the alphabetic characters and the digits 0 through 9.

§§ Deferred implementation

COS first searches the list of system verbs for a match. If the verb is not a system verb, COS searches for a local dataset name that might match the verb. If the verb is not the name of a local dataset, COS searches each library in the library searchlist for a match. If it does not find a library entry that matches the verb, it searches the System Directory Table (SDR) for a matching system dataset name. If a match for the verb is not found under any of these categories, COS issues a control statement error.

SEPARATORS

A *separator* is a character used as a delimiter in a control statement. It separates the verb from the first parameter, separates parameters from one another, delimits subparameters, terminates verbs and parameters, and separates a keyword from its value in parameters having keyword form.

The control statement separators allowed by COS are given in table 4-1.

PARAMETERS

A *parameter* is a control statement argument, the exact requirements of which are defined by the verb. Parameters are used in control statements to specify information to be used by the verb-defined process. Parameters that can be used with COS control statements are either *positional* or *keyword*. For certain verbs, a parameter value can be an expression. Detailed information on the use of expressions is presented later in this section. Parameters are separated by commas.

POSITIONAL PARAMETERS

A positional parameter has a precise position relative to the separators in the control statement. Even a null positional parameter must be delimited from the verb or other parameters by a separator.

The format for a positional parameter is:

value
or
$value_1{:}value_2{:}\ldots{:}value_n$

where each $value_i$ is a string of alphanumeric characters, a literal string, or a null string. All positional parameters are required to be represented by at least one *value*, although the value may be null. Rules for strings are given later in this section.

Table 4-1. Control statement separators

Function	Character	Examples
Initial separator (comma or open parenthesis)[§] - Separates the verb from the first parameter	, (	*VERB,parameter*. *VERB*(*parameter*)
Statement terminator (period if initial separator is comma; close parenthesis if initial separator is open parenthesis)[§] - Signifies end of control statement	.)	*VERB*. *VERB,parameter*. *VERB*(*parameter*)
Parameter separator (comma) - Indicates the end of one parameter and the beginning of the next	,	*VERB*(*parameter,parameter*)
Equivalence separator (equal sign) - Delimits a parameter keyword from the first parameter value for that keyword. Adjacent equivalence separators are illegal.	=	*VERB*(*keyword=value*)
Concatenation separator (colon) - Separates multiple parameter values in a keyword parameter from each other	:	*VERB,keyword=value*$_1$*:value*$_2$.
Continuation character (caret) - Indicates that the control statement consists of more than one 80-character card; may appear anywhere after the initial separator	∧	*VERB*(*...parameters...*∧ *parameters*)
Literal delimiters (apostrophes) - Identify the beginning and end of a literal string	'...' (...)	*VERB*(...'*string*'...)
Parenthesis delimiters (open and close parentheses) - Indicate a group of characters to be treated as one value	(...)	*VERB,keyword*=(*value:value*).

[§] By convention in this manual, the comma and period are used as initial and terminator separators for all control statements except for the JCL block control statements (procedure definition, iterative, and conditional) where paired parentheses are advisable.

Examples of positional parameters:

...,ABCDE,...	Parameter value is ABCDE.
...,,...	The adjacent parameter separators indicate a null positional parameter.
...,P1:P2:P3,...	The parameter consists of multiple values.
VERB() or VERB,.	Positional parameter 1 is null

KEYWORD PARAMETERS

A keyword parameter is identified by its form rather than by its position in the control statement. The keyword is a string of one to eight alphanumeric characters uniquely identifying the parameter. Parameters of this type can occur in any order but must be placed after all of the positional parameters for the control statement; or they can be omitted.

The format of a keyword parameter is:

keyword
  or
keyword=value
  or
$\textit{keyword}=\textit{value}_1:\textit{value}_2:\ldots:\textit{value}_n$

where *keyword* is an alphanumeric string that depends on the requirements of the verb, and $\textit{value}_i$ is the value associated with the keyword. A keyword parameter can occur anywhere in the control statement after all positional parameters are specified. Whether or not a keyword parameter is required depends on the verb's requirements. If the keyword is not included in the control statement, a default value can be assigned by the prototype statement.

Examples of keyword parameters:

...,DN=FILE1,...	Parameter consists of keyword and value.
...,UQ,...	Parameter consists of keyword only.
...,DN=FILE1:FILE2:FILE3,...	Parameter consists of keyword and list of values.
...,DN=,...	Null parameter value, as if omitted from the statement.
...,DN=A:::B,...	A, B, and two null parameter values are listed.

JCL PARAMETER EXPRESSIONS

The JCL block control statements described in part 2, section 2 require a parameter value known as a *JCL parameter expression*. Others, such as the prototype statement and the definition calling statement can include expressions.

An expression consists of *operands* and *operators*. Parentheses should be used to delimit expressions. See the description later in this section on the use of apostrophes and parentheses in JCL block control statements.

Operands

Expression *operands* are of four types: integer constants, literal constants, symbolic values, and subexpressions.

Integer constants - An *integer constant* is a character string of the form:

{±}*ddd*...

where *d* is a decimal digit, or

nnn...B

where *n* is an octal digit.

An integer constant has an approximate decimal range $0 \geq |I| \geq 10^{19}$. Range overflow is not detected and overflow results may be unpredictable.

Literal constants - A *literal constant* is a string of one to eight characters of the form:

'*ccc*...'L
'*ccc*...'R
'*ccc*...'H

where *c* is a character code with an ordinal number in the range 040_8 through 176_8. The value of a character constant corresponds to the ASCII character codes positioned within a 64-bit word. Alignment is indicated by the following suffixes:

L	Left-adjusted, zero-filled
R	Right-adjusted, zero-filled
H	Left-adjusted, space-filled

If no suffix is supplied, H is assumed.

Symbolic variable - A *symbolic variable* is a string of one to eight alphanumeric characters, beginning with an alpha character, of the form:

ccc...

A symbolic variable always has an associated value that is either constant or varies. COS defines a set of symbols when the job is initiated. Symbols are mnemonics for values maintained by COS and/or the user. The user may manipulate the group of symbols listed in table 4-2 through COS control statements or through system requests.

Certain symbols allow communication between COS and the job being processed. Used in the JCL block control statements defined in part 2, section 2, they provide the user with powerful tools for analyzing the progress of a job. For example, a job can request the reason for an abort situation and proceed, based on the reply from COS, through the use of conditional control statements. Symbols that are preserved over subprocedure calls are called *local* to a procedure; they are saved when a subprocedure is called. Those that are not preserved are *global* over all procedures and can be altered by any procedure. *Constants* are symbols that are never altered.

Information on predefined symbols is summarized in table 4-2.

Subexpressions - A subexpression is an expression that is evaluated so that its result becomes an operand.

Operators

Expression *operators* are of three types: arithmetic, relational, and logical. These operators are used in the FORTRAN sense. The expression operators are detailed in table 4-3.

Arithmetic Operators - All *arithmetic operations* are performed on 64-bit integer quantities. Care must be used with arithmetic operators because:

- Multiplication/division underflow or overflow of the result is not detected,
- Division by zero produces a zero result.
- Intermediate and final results are truncated. For example, 2*(13/2) yields 12 whereas (2*13)/2 yields 13.

Relational Operators - *Relational operations* return a -1 value for a TRUE result and a 0 value for a FALSE result. A value produced by arithmetic or logical operation is considered TRUE if it is a negative value.

Table 4.2 Symbolic variable table

Symbol	Set by	Range	Description	Local/Global
J0-J7	U	Any 64-bit value	Job pseudo-registers; represent user-alterable data local to a procedure. Each procedure level can be considered to have its own set of J registers.	LOCAL
G0-G7	U	Any 64-bit value	Global job pseudo-registers; represent user-alterable data global over all procedure levels. Data can be passed into or returned from procedures with the G registers.	GLOBAL
JSR	U	Any 64-bit value	Job status register; previous job step completion code (normally 0)	GLOBAL
FL	S	$0\text{-}77777777_8$	Current job field length; can be set with RFL statement.	GLOBAL
FLM	I	$0\text{-}77777777_8$	Maximum job field length	GLOBAL
SYSID	I	Literal value	COS system level of the form 'COS X.XX'	GLOBAL
SSW*n*	S	$(1 \geq n \geq 6)$	Job pseudo sense switch settings; can be set with the SWITCH statement	GLOBAL
ABTCODE	S	System error codes (See Appendix F) 0-*nnn*	COS job abort code; abort code corresponding to the last job step abort. The abort code corresponds to the abort message number (the *nnn* in AB*nnn*) issued by COS.	GLOBAL
TRUE	I	-1	True value	GLOBAL
FALSE	I	0	False value	GLOBAL
TIME	S	Literal value	Time of day in the form: *hh:mm:ss*	GLOBAL
DATE	S	Literal value	Date in the form: *mm/dd/yy*	GLOBAL
TIMELEFT	S	64-bit integer	Job time remaining in milliseconds as an integer value	GLOBAL
PDMFC	S	64-bit value	Most recent user-issued Permanent Dataset Manager request. See Appendix C.	GLOBAL
PDMST	S	64-bit value	Status of most recent Permanent Dataset Manager request. See Appendix F.	GLOBAL

U Alterable by user
S Set by COS
I System constant

Table 4-3. Expression operator table

Type	Function	Symbol	Results
A	Addition	+	64-bit sum of operands
A	Unary plus	+	Following integer operand is positive.
A	Subtraction	-	64-bit difference of operands
A	Unary minus	-	Following integer operand is negative.
A	Multiplication	*	64-bit product of operands
A	Division	/	64-bit quotient of operands
R	Equal	.EQ.	True/false
R	Not equal	.NE.	True/false
R	Less than	.LT.	True/false
R	Greater than	.GT.	True/false
R	Less than or equal	.LE.	True/false
R	Greater than or equal	.GE.	True/false
L	Inclusive OR	.OR.	A 1 bit in either operand sets corresponding bit in the result.
L	Intersection	.AND.	A 1 bit in both operands sets corresponding bit in the result.
L	Exclusive OR	.XOR.	A 1 bit is set in the result if either (but not both) corresponding bit in the operands is 1.
L	Unary complement	.NOT.	A 1 bit (or 0) is set in the result if the corresponding operand bit is 0 (or 1).

A Arithmetic
R Relational
L Logical

Logical Operators - *Logical operations* return a 64-bit result. Their functions are performed on a bit-by-bit basis.

Expression Evaluation

Expressions are evaluated from left to right, honoring nested parentheses. The operator hierarchy is:

1. Multiplication and division
2. Addition, subtraction, and negation
3. Relational operation
4. Complement (.NOT.)
5. Intersection (.AND.)
6. Inclusive OR (.OR.)
7. Exclusive OR (.XOR.)

Parentheses can be used to change the order of evaluation. For example, 2+3*4 is evaluated as 14 whereas (2+3)*4 is evaluated as 20.

> CAUTION
>
> Because COS does not check for type, the results of expression evaluation may not be as expected. For example, although both J1=1 and J2=2 are TRUE, (J1 .AND. J2) is FALSE.

PARAMETER INTERPRETATION

The cracking and interpretation of control statement parameters is performed by $CCS and GETPARAM. These processes are described in the Library Reference Manual, CRI publication SR-0014.

STRINGS

A *string* is a group of characters, delimited with either apostrophes or open and close parentheses, which is to be taken literally as a parameter value.

Strings are normally delimited with apostrophes, in which case they are referred to as *literal strings*. Strings delimited with parentheses are called *parenthetic strings*. Parentheses are advised to delimit strings in JCL block statements. See the description later in this section concerning the use of apostrophes and parentheses in JCL block control statements.

Characters in a string can be any ASCII graphic characters (codes 040 through 176). Characters otherwise recognized as separator characters are not evaluated as such when part of a string.

'SEPARATORS IN STRING,.=()' — The literal string contains separator characters which are not interpreted as such.

(ABC=DEF) — The parenthetic string contains an equal sign which is not interpreted as a separator.

STRING CONSTRUCTION RULES

Apostrophes are never treated as part of a literal string during evaluation except when doubled (see below). The outermost parentheses of a parenthetic string are not treated as part of the string during parameter evaluation if preceded by the initial, parameter, equivalence, or concatenation separators.

KEYWORD=(ABC.DEF) — ABC.DEF is the value assigned to KEYWORD.

'ABC.DEF' — ABC.DEF is the string value.

To continue literal strings across card images, place an apostrophe followed by a continuation character at the end of the line, and place the remainder of the string on the next card image preceded by an apostrophe. To continue parenthetic strings, place a continuation character at the end of the line and the remainder of the string on the next card image. A string can be any length, depending upon the control statement parameter requirements.

```
...'LITERAL STRING CONTINUED' ∧
'ACROSS CARD IMAGES'
```

This is the format for continuing literal strings across card images.

```
...(PARENTHETIC STRING CON-   ∧
TINUED ACROSS CARD IMAGES)
```

This is the format for continuing parenthetic strings across card images.

Two adjacent literal delimiters are interpreted as a null string.

''   or   () — Both are null strings.

The continuation and literal string delimiters are interpreted when included in a parenthetic string.

```
...:(STRING WITH 'EXTRA CLOSE PAREN )')...
```

STRING WITH EXTRA CLOSE PAREN) is the value of the string following the concatenation separator.

```
...=(STRING CONTINUED ACROSS ∧
CARD IMAGES)...
```

STRING CONTINUED ACROSS CARD IMAGES is the value of the string following the equivalence separator.

An apostrophe within the string is indicated by doubling it.

```
'DON''T'
```

The literal string is interpreted as DON'T.

APOSTROPHES AND PARENTHESES IN JCL BLOCK CONTROL STATEMENTS

The IF, ELSEIF, EXITLOOP, PRINT, SET, procedure definition prototype, and definition calling statements described in part 2, section 2 can include expressions. Since an expression can include a literal constant which is delimited with apostrophes, values delimited with apostrophes in these statements are always treated as literal constants. Therefore, apostrophes should only be used to delimit literal constants, and parentheses should be used to delimit strings. Apostrophes in these statements are retained as part of the value during statement cracking, parameter substrition, and parameter evaluation. Also use parentheses as the initial and terminator separators instead of the usual comma and period to allow the period to be treated as an expression operator instead of a control statement terminator.

```
IF(GO.NE.'YES'L)
```

Creates value with a literal constant: protects the expression characters during statement cracking so that periods are evaluated as expression operators instead of statement terminators and apostrophes are evaluated as part of literal constant instead of being treated as string delimiters.

```
IF,GO.NE.''YES''L.
```

ERROR. First period processed as statement terminator; expression not evaluated.

```
IF,'GO.NE.''YES''L'.
```

ERROR. GO.NE.''YES''L is the single literal constant which is created.

More specific information about how to use apostrophes and parentheses in procedure definition and calling statements is presented in part 2, section 2, Procedure Definition.

PART 2

JOB CONTROL LANGUAGE

CONTENTS
PART 2 JCL CONTROL STATEMENTS

OVERLAY LOADING (continued)

FIGURES

INTRODUCTION AND JOB DEFINITION 1

INTRODUCTION

Job control statements identify a job to the system, define operating characteristics for the job, manipulate datasets, call for the loading and execution of user programs, call programs that perform a number of utility functions for the user, and define and manipulate control statements themselves. The first file of a job dataset contains control statements that are read, interpreted, and processed.

Information on the general syntax rules and conventions for control statements is presented in part 1, section 4. This part describes COS control statements individually and gives examples in some cases. The control statements have been divided into the categories listed below. The Job Definition statements are described in this section; the other categories are defined in the remaining sections in part 2.

- Job Definition and Control - JOB, MODE, EXIT, RFL, SWITCH, *, NORERUN, RERUN, IOAREA, CALL, RETURN, ACCOUNT, CHARGES, ROLLJOB, SET, ECHO, LIBRARY, and OPTION
- Control Statement Blocks - In-line procedure definition, conditional control statement processing, and iterative control statement processing
- Dataset Definition and Control - ASSIGN and RELEASE
- Permanent Dataset Management - SAVE, ACCESS, ADJUST, MODIFY, and DELETE
- Dataset Staging Control - ACQUIRE, DISPOSE, and SUBMIT
- Dataset Utilities - COPYR, COPYF, COPYD, SKIPR, SKIPF, SKIPD, REWIND, and WRITEDS
- Permanent Dataset Utilities - PDSDUMP, PDSLOAD, and AUDIT
- Analytical Aids - DUMPJOB, DUMP, DEBUG, DSDUMP, and COMPARE, FLODUMP, PRINT, and SYSREF

The relocatable and overlay loader (LDR) and the BUILD utility are also described in part 2.

JOB DEFINITION

Several control statements allow the user to specify job processing requirements. Control statements defining a job and its operating characteristics to the operating system include the following:

- JOB defines the job to the operating system and sets characteristics such as size, time limit, and priority levels.
- MODE allows the user to set or clear the floating-point error interrupt flag.
- EXIT indicates the point in a series of control statements at which processing of control statements resumes following a job step abort from a program or indicates the end of control statement processing.
- RFL allows the user to request a new field length.
- SWITCH allows the user to turn on or turn off pseudo sense switches.
- * allows the user to annotate control statements with comments.
- RERUN and NORERUN allow the user to set job rerunnability.
- IOAREA denies or allows access to the user's I/O area.
- CALL and RETURN allow the user to manipulate control statement files.
- ACCOUNT validates the user's account number and optional password.
- CHARGES allows the user to obtain partial or total resource reporting for a job.
- ROLLJOB allows the user to protect a job by writing it to disk.
- SET allows the user to change the value of a job control language (JCL) expression.
- ECHO allows the user to control the message classes to be written to the user's logfile.
- LIBRARY allows the user to specify the library datasets to be searched for defined procedures during job processing and in which order.
- OPTION allows the user to specify user-defined options, such as the format of the job's listing.

JOB - JOB IDENTIFICATION

The JOB control statement defines the job to the operating system. It must be the first statement in a control statement file. The JOB control statement cannot be continued to subsequent cards. No leading blanks are allowed on the JOB statement. JOB is a system verb.

Format:

```
JOB,JN=jn,M=fl,T=tl,P=p,US=us,OLM=olm,CL=jcn,*gn=nr.
```

Parameters are in keyword form; the only required parameter is JN.

JN=*jn* Job name. 1-7 alphanumeric characters. This name identifies the job and its subsequent output. JN is a required parameter.

M=*fl* Memory field length. *fl* specifies an octal count of 1000_8-word (512_{10}) blocks of memory to be assigned to the job. The limit address is a function of the base address and requested field length: $(LA)=(BA)+fl*1000_8$.

If this parameter is omitted, the field length is set by the system to a value determined by an installation parameter.[§]

If M is present without a value, the field size is the maximum amount that can be assigned. The maximum amount allowed is either the total amount of memory available after the operating system is initialized or is an installation-defined maximum job field length whichever is smaller.

T=*tl* Time limit in decimal seconds after which the job is terminated by the system. If this parameter is omitted, the time limit is set to a value determined by an installation parameter. If T is present without a value, a maximum of 16,777,215 seconds (approximately 194 days) is allowed.

[§] The *fl* parameter on the JOB statement does not include the job's Job Table Area (JTA); space for the JTA is added by the system. The installation parameter, however, does include the JTA.

P=*p* Priority level at which the job enters the system. This parameter may assume the values of 0-15 decimal. If P is 0, the job will not be initiated. If omitted, a value specified by the installation is assumed.

US=*us* User number. 1-15 alphanumeric characters. The default is no user number. This parameter identifies the user submitting the job. The user number feature is provided for installation accounting; specific application is installation-defined.

OLM=*olm* Size of $OUT. *olm* specifies a decimal count of 512-word blocks. A block holds about 45 print lines. The default and maximum values for *olm* are defined by the installation.

CL=*jcn* Name of the installation-defined job class where this job is to be placed. 1 to 7 alphanumeric characters. The job is aborted if it does not fit the requirements of the indicated class or if the indicated class does not exist. The default is that the job is placed in the highest rank class in which it fits.

**gn*=*nr* Type and number of dedicated resources required by a job. Dedicated resource requirements are specified with *gn* and *nr*.

**gn* is a generic name of 1 through 7 alphanumeric characters. A *generic name* (or its installation-defined synonym) corresponds to a device type. For example, the generic name is *1600 if the job requires a tape unit capable of 1600 bpi.

nr is a positive integer; the default is 0. The job is aborted if it requests more resources than are dedicated on the JOB statement.

Generic Name	Synonym	Significance
*6250	*TAPE	Device capable of 6250 bpi or 1600 bpi
*1600[§]		Device capable of 1600 bpi

Example:

*TAPE=2 requests two 6250-bpi devices for use by the job.

[§] Deferred implementation

MODE - SET OPERATING MODE

The MODE control statement allows the user to set or clear the floating-point error interrupt flag in the mode (M) register in the exchange package for the job. This flag controls whether or not a floating-point error will cause an interrupt flag to be set in the flags (F) register. If a floating-point error condition occurs, an exit from the program occurs only if the floating-point error flag is set in the mode register.

Format:

```
MODE,M=mode.
```

Parameters:

M=mode Operating mode. May be any of the following:

DFI, 1, or 2	Disable floating-point error interrupt
EFI, 3, or 4	Enable floating-point error interrupt

EXIT - EXIT PROCESSING

An EXIT control statement indicates the point in the control statement file where processing of control statements resumes following a job step abort from a program. If no job step abort occurs, the EXIT control statement indicates the end of the control statement processing. EXIT is a system verb.

Format:

```
EXIT.
```

Parameters: none

RFL - REQUEST FIELD LENGTH

The RFL control statement allows the user to request a new field length. RFL is a system verb.

Format:

```
RFL,M=fl.
```

Parameters:

M=*fl* New field length which is the octal number of 1000_8-word (512_{10}) blocks of memory to be assigned to the job, excluding the Job Table Area. M is a required parameter. If M is present without a value, the field length is the maximum that can be assigned to the job. The maximum is either the total memory available after the operating system is initialized or is an installation-defined maximum job field length, whichever is smaller.

SWITCH - SET OR CLEAR SENSE SWITCH

The SWITCH control statement allows a user to turn on or turn off pseudo sense switches. SWITCH is a system verb.

Format:

```
SWITCH,n=x.
```

Parameters:

n Number of switch (1-6) to be set or cleared

x Switch position
ON Switch *n* is turned on; set to 1
OFF Switch *n* is turned off; set to 0

* - COMMENT STATEMENT

The comment control statement allows the user to annotate job control statements with comments. A period is not required on a comment control statement. * is a system verb.

Format:

```
*                  comment text
```

Parameters: none

NORERUN - CONTROL DETECTION OF NONRERUNNABLE FUNCTIONS

The NORERUN control statement allows the user to specify whether the operating system is to recognize functions that would make a job rerunnable. The current rerunnability of the job is not affected. NORERUN is a system verb.

Format:

$$\boxed{\text{NORERUN}\left\{,\begin{matrix}\text{ENABLE}\\\text{DISABLE}\end{matrix}\right\}.}$$

The keywords ENABLE and DISABLE are mutually exclusive. The default for the system as released is NORERUN,ENABLE; however, this is an installation option.

Selecting ENABLE instructs the system to begin monitoring functions performed by the job and to declare the job nonrerunnable if any of the nonrerunnable functions are performed.

Selecting DISABLE instructs the system to stop monitoring functions for nonrerunnable operations. If a job has already been declared to be nonrerunnable, specifying DISABLE does not make the job rerunnable again.

RERUN - UNCONDITIONALLY SET JOB RERUNNABILITY

The RERUN control statement allows the user to unconditionally declare a job to be either rerunnable or nonrerunnable. If RERUN is used to declare a job rerunnable, the subsequent execution of a nonrerunnable function may cause the system to declare the job nonrerunnable, depending on whether a NORERUN control statement or macro is also present. RERUN is a system verb.

Format:

$$\boxed{\text{RERUN}\left\{,\begin{matrix}\text{ENABLE}\\\text{DISABLE}\end{matrix}\right\}.}$$

The keywords ENABLE and DISABLE are mutually exclusive. If no parameter is specified on the control statement, installation option determines if the job is to be rerunnable or not; the default for the system as released is RERUN,ENABLE.

If ENABLE is selected, the system is instructed to consider the job to be rerunnable, regardless of what functions have been executed previously.

If DISABLE is selected, the system marks the job not rerunnable regardless of what functions have been executed previously.

The RERUN control statement in no way affects the monitoring of the user job for nonrerunnable functions.

IOAREA - CONTROL USER'S ACCESS TO I/O AREA

The IOAREA control statement locks (denies the user access to) or unlocks (gives the user access to) that portion of the user field containing the user's Dataset Parameter Area (DSP) and I/O buffers. This area follows the High Limit Memory address (HLM) of the user field if locked. IOAREA is a system verb.

Format:

$$\boxed{\text{IOAREA}\left\{,\begin{matrix}\text{LOCK}\\ \text{UNLOCK}\end{matrix}\right\}.}$$

The keywords LOCK and UNLOCK are mutually exclusive. A parameter must be specified on the control statement. When the control statement is not used, the user's I/O area is assumed to be unlocked.

If LOCK is selected, the system sets the limit address to the base of the DSPs, thereby denying direct access to the user's DSP area and I/O buffers. When the I/O area is locked, the library I/O routines make a system request to gain access to the I/O area. This introduces additional overhead in job processing but should prevent accidental destruction of the I/O area.

If UNLOCK is selected, the system sets the limit address to the value specified in JCFL, allowing access to the user's DSP area and I/O buffers.

CALL - READ CONTROL STATEMENTS FROM ALTERNATE DATASET

The CALL control statement instructs COS to begin reading control statements from the first file of the indicated dataset. CALL may appear anywhere in the control statement file. Nesting of CALL statements is allowed to seven levels. COS reads and processes the control statements from the indicated dataset until it encounters an end-of-file or a RETURN statement. Control then reverts to the previous control statement dataset; the named dataset is closed prior to the invocation of the procedure. The CALL statement can also specify values to be substituted in the procedure body. CALL is a system verb.

Format:

```
CALL,DN=dn,CNS.
```

Parameters are in keyword form.

DN=*dn* Name of dataset from which to begin reading control statements. This is a required parameter.

CNS If specified, the control statement that follows is a *procedure calling statement* containing parameters for procedure string substitution. The format of the procedure calling statement depends upon the format of the prototype statement. The prototype statement format is described in part 2, section 2. If CNS is omitted, no substitution is performed. CNS (Crack Next Statement) cannot be equated.

RETURN - RETURN CONTROL TO CALLER

The RETURN control statement returns control to the caller. The caller can be a procedure or the job's control statement file. Processing resumes with the caller's next control statement. A RETURN control statement may be embedded anywhere within the called procedure. However, it is not necessary to place a RETURN control statement at the end of the procedure because an end-of-file is interpreted as the control statement sequence of an EXIT, RETURN, and RETURN,ABORT. A RETURN encountered in the primary control statement file is ignored. RETURN is a system verb.

Format:

```
RETURN,ABORT.
```

Parameter:

ABORT After returning to the previous control statement level, ABORT causes COS to issue a job step abort. ABORT is an optional parameter.

ACCOUNT - VALIDATE USER ACCOUNT

The ACCOUNT control statement validates the user's account number and optional password. A job is processed only if the account number and password (if specified) are valid.

The ACCOUNT statement declares the user's account number to COS. It must immediately follow the JOB control statement if the installation has defined accounting as mandatory. Only one ACCOUNT statement is allowed per job. The ACCOUNT control statement cannot be continued to subsequent cards. ACCOUNT is a system verb.

NOTE

The ACCOUNT control statement parameters do not appear with the ACCOUNT control statement in the job logfile.

Format:

```
ACCOUNT,AC=ac,PW=pw.
```

Parameters are in keyword form. The only required parameter is AC; the installation defines whether a password is needed.

AC=*ac* Account number. 1-15 alphanumeric characters assigned to the user. This number identifies the user for accounting purposes, and is a required parameter. The account number is not the same as the user number on the JOB control statement, unless the site chooses to use the same characters for both numbers.

PW=*pw* Password. 1-15 alphanumeric characters. A password must be specified if the installation has made it mandatory by installation parameter.

CHARGES - JOB STEP ACCOUNTING

The CHARGES control statement allows the user to monitor a job's usage of computer resources up to a specific point in a job. Hence, CHARGES can be used for either partial or total resource reporting.

Partial reporting occurs when parameters are specified on the control statement. In this case, usage statistics for the computer resources specified on the CHARGES statement are obtained for the job steps preceding the CHARGES statement. The summary is placed in the user log and the system log.

Total reporting occurs when usage statistics are obtained for all the resources in all the available resource groups. The summary is placed in the user log and the system log.

A CHARGES statement may be placed in a job deck any number of times. If no CHARGES control statements are used in a job deck, computer resource usage statistics are gathered only upon job termination and placed in the user log.

Format:

```
CHARGES,SR=options.
```

Parameters are in keyword form.

SR=*options*

System resources used. Any one or more of the following groups of resources can be specified. Options are separated by colons. The default is a listing of the job's usage of resources in all of the following groups:

JNU Job name and user number

DS Permanent dataset space accessed, permanent dataset space saved, temporary dataset space used, 512-word disk blocks (sectors) moved, user I/O requests, memory resident datasets used, number of OPEN calls and number of CLOSE calls

WT I/O wait time, time waiting to execute and time waiting for a JXT

MM Minimum job size (words), maximum job size (words), execution-time memory integral, I/O wait-time memory integral, maximum field length used (words), minimum field length used (words), maximum JTA used (words), and minimum JTA used (words)

CPU Time executing in CPU

NBF Number of 512-word blocks (sectors) received from a front end and number of 512-word blocks (sectors) queued to a front end

TPS Number of tape devices reserved, number of tape volumes mounted, amount of tape data moved (expressed as a multiple of 512 words) and number of tape blocks moved

ROLLJOB - ROLL A USER JOB TO DISK

The ROLLJOB control statement allows the user to protect a job by writing it to disk so that it can be recovered in case a system interruption occurs. ROLLJOB is a system verb.

Format:

ROLLJOB.

Parameters: none

SET - CHANGE SYMBOL VALUE

The SET control statement changes the value of a specified valid job control language symbol. Valid symbols are those classified as alterable by the user (U) in table 4-2 in part 1. A job step abort occurs if a symbol included in a SET control statement is unknown to the system, can be set only by COS, or is a constant. SET is a system verb.

Format:

SET(*symbol=expression*)

Parameters:

symbol A valid user-alterable symbol; *symbol* is a required parameter.

expression
A valid arithmetic, logical, or literal assignment expression. It may be delimited with parentheses to simplify interpretation during control statement evaluation. *expression* is a required parameter.

Examples:

```
    SET(J1=J1+1)
```

This example increments the procedure-local register J1 by 1.

```
    SET(G1=(SYSID.AND.177777B))
```

The global register G1 is given an ASCII value which is the low-order two characters from the current system revision level (COS *X*.*XX*).

```
    SET(G3=((ABTCODE.EQ.74).AND.(G2.EQ.0)))
```

The global register G3 is assigned a value, depending upon the current values of ABTCODE and G2.

ECHO - ENABLE OR SUPPRESS LOGFILE MESSAGES

The ECHO control statement allows the user to control the message classes to be written to the user's logfile by turning the classes ON or OFF. ECHO can be used more than once during a job to toggle the printing/suppression of message classes. ECHO is a system verb.

Format:

ECHO,ON=$class_1$:$class_2$:...:$class_i$,OFF=$class_1$:$class_2$:...:$class_i$.

Parameters are in keyword form.

ON=$class_i$ Only the messages in the classes specified are written to the user's logfile. If only the keyword ON or ON=ALL is specified, all messages are written to the logfile.

JCL is the currently available message class. If the JCL message class is enabled (ON), the JCL control statements are echoed on the user's logfile; if it is disabled, the JCL control statements are not listed on the logfile.

OFF=*class*$_i$

The messages in the classes specified are not written to the user's logfile. If only the keyword OFF or OFF=ALL is specified, all messages in defined classes are suppressed. OFF=JCL suppresses echoing of JCL control statements to logfile; however, output resulting from the execution of the control statements will appear.

The keywords ON and OFF can be used in any combination: both, either, or neither. However, a particular class should not be included in both ON=*class*$_i$ and OFF=*class*$_i$, nor should both defaults (ON and OFF) be included. When the ECHO statement is not used, all messages are written to the user's logfile.

Specify each class to be written or not written instead of using the defaults (ON and OFF) because additional classes may be added.

When a job calls a procedure, the echo state of the job is the same upon return from the procedure as before, even though the procedure may use a different echo state. The following occurs when ECHO is used in conjunction with CALL and PROC: (1) The echo state of the caller (a job or another procedure) is saved so that on return to the caller the same state is in effect as before the call, and (2) When the procedure is called, a new echo state is created that affects only the procedure. If the procedure does not include an ECHO statement, the echo state of the caller is in effect. The echo state of the procedure can be changed during the procedure's execution.

LIBRARY - LIST AND/OR CHANGE LIBRARY SEARCHLIST

The LIBRARY control statement allows the user to specify the library-defined dataset names that are to be searched during the processing of control statement verbs. It also allows the user to list the current or new searchlist to the logfile for verification.

When modifying the searchlist, the current members of the searchlist may be retained in the new searchlist by including an asterisk in the LIBRARY control statement. The asterisk corresponds to all members of the current searchlist in their present order. If the asterisk is omitted, the new searchlist contains only the library dataset names identified on the LIBRARY control statement. LIBRARY is a system verb.

The default library searchlist upon job initiation consists of the single library dataset $PROC.

Format:

LIBRARY,DN=dn_1:dn_2...:dn_{64},V.

DN=dn_i Library dataset names to become members of the new library searchlist. A maximum of 64 names (separated by colons) may be specified. The order in which they appear is the order in which they are searched. An asterisk included in the list means that the current searchlist members are to be part of the new searchlist in their current order.

V List the current library searchlist on the logfile for verification. When specified along with the new searchlist, the new searchlist is listed.

OPTION - SET USER-DEFINED OPTIONS

The OPTION control statement allows the user to specify the user-defined options, such as the format of the job's listing. OPTION is a system verb.

Format:

OPTION,LPP=n,STAT=$\left\{\begin{matrix}\text{ON}\\ \text{OFF}\end{matrix}\right\}$.

Parameters:

LPP=n Number of lines per page; a decimal number from 0 through 255. If 0 is specified, the current number of lines per page is not changed. The default is an installation parameter.

STAT=$\left\{\begin{matrix}\text{ON}\\ \text{OFF}\end{matrix}\right\}$ STAT=ON causes dataset I/O statistics for each job to be printed in the user logfile whenever a dataset is released. The statistics include dataset name, device name, dataset size, number of user I/O requests, number of 512-word blocks transferred, and total time blocked for I/O for the dataset. No statistics are printed if STAT=OFF, which is the default condition.

JCL CONTROL STATEMENTS BLOCKS 2

Certain control statements are grouped in the control statement file to create a *control statement block*. The concepts and techniques for using control statement blocks assume that the reader is familiar with the control statements described in part 2, section 1 and has some experience with running simple jobs under COS control.

Control statement blocks provide the user with the following capabilities:

- *Procedure definition*. The user can request that a series of control statements and/or data be written to a library and called for processing at a later time. Parameters within this procedure can be substituted during processing.
- *Conditional control statement processing*. The user can identify control statements that are to be processed only if certain conditions are met.
- *Iterative control statement processing*. The user can identify control statements to be processed repetitively.

Parentheses are advised as initial and terminator separators in the JCL block control statements to avoid possible errors during processing resulting from the unique treatment of apostrophes and parentheses in these statements. See part 1, section 4 for a general description of the use apostrophes and parentheses in the JCL block control statements.

PROCEDURE DEFINITION

A *procedure* is a sequence of control statements and/or data that has been saved for processing at a later time. Procedures have two formats.

The *simple procedure* format consists of only the control statement body.

The *well defined procedure* format consists of a prototype definition statement, control statement body, and optional data. It provides the capability of replacing values within the procedure body with values supplied from the procedure call. These values are called *substitution parameters* and are governed by the prototype statement of the procedure.

A well defined procedure can reside in a library or non-library dataset. A simple procedure can only reside in a non-library dataset because a simple procedure has no name associated with it.

Processing (invocation) of procedures can be initiated by a procedure name call or with the CALL control statement (see part 2, section 1). A simple procedure, because it does not have a name, must be invoked with the CALL control statement without the CNS parameter. A well defined procedure can also be invoked with the CALL statement but the CNS parameter must be included. The presence of CNS on a CALL statement indicates that the procedure to be called has a prototype statement and that it should be processed as such.

Well defined procedures can be defined within the control statement stream (*in-line definition*) or as input to the BUILD utility§. When an in-line procedure definition is encountered in the JCL control statement file, it is processed and written to the system default library $PROC. See example 8 in this section for an example of how to create a user permanent procedure library.

PRODCEDURE DEFINITION FORMAT

A simple procedure format consists of only the control statement body. The format of an in-line procedure is shown in figure 2-1. The first control statement in an in-line procedure is PROC; the last is ENDPROC. A prototype statement follows PROC; it provides the name of the procedure and optionally a list of parameters that identify the substitution values within the definition body. In addition to defining the values to be substituted, the prototype statement parameters control the selection or omission of the parameters and define the default value assignments. The control statements and data to be processed are contained in the definition body. The control statements are grouped in a sequence.

If data is included in a procedure, the data is preceded by an &DATA statement and follows the control statement sequence. The &DATA statement also includes the name of the dataset to which the data is to be written after processing so that programs can use the data as source data.

§ BUILD currently does not suppport procedure entries in libraries.

A definition can be placed within a definition; such nesting can occur to any level. However, nested definitions are not defined until the outermost procedure is invoked.

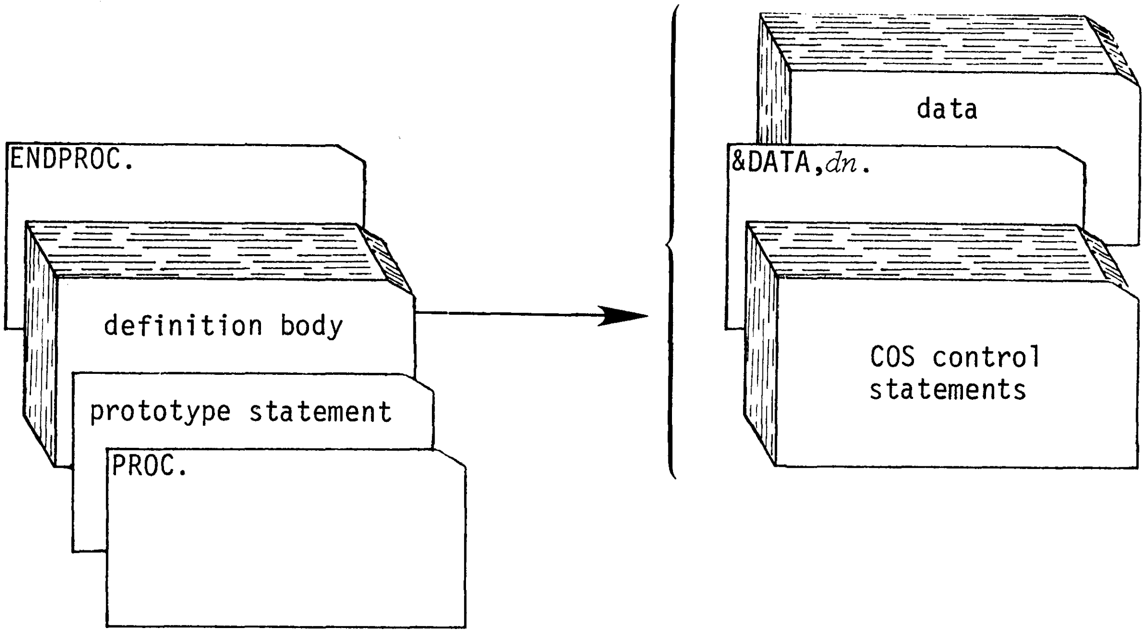


Figure 2-1. Procedure definition deck structure

PROC - Begin procedure definition

The PROC control statement defines the beginning of an in-line procedure definition block. PROC is a system verb.

Format:

```
PROC.
```

Parameters: none

Prototype statement - Introduce a procedure

The prototype control statement has two functions: (1) to specify the name of the procedure and (2) to provide the *formal parameter specifications* that define where substitution is to occur within the definition body. Value substitution is described later in this section.

Format:

$$name,p_1,p_2,p_3, \ldots ,p_n.$$

name Procedure name; 1 through 8 alphanumeric characters.

p_i Formal parameter specifications, using one of the formats listed below. A formal parameter identifies a character string within the definition body. All formal positional parameters must precede all formal keyword parameters; if not, the procedure definition is in error and the job aborts.

pos_i Positional formal parameter specification, or

$key_i=\{dvalue\}:\{kvalue\}$
Keyword formal parameter specification as follows:

keyi Formal keyword parameter

dvalue Optional default value; this value is substituted if entire keyword parameter is omitted from the calling statement.

kvalue Optional keyed default value; this value is substituted if the keyword is present but no value is specified.

Special cases:

key_i= Provides no default values and requires the caller to provide a non-null value.

key_i=: Provides no default values, but allows the user to specify key_i= or just key_i.

Procedure definition body

The procedure definition body consists of a sequence of COS control statements processed as part of the current control statement file when the procedure is called. (It can optionally include lines of text data preceded in the definition body by an &DATA control statement. See &DATA below.)

The prototype statement identifies character strings within the procedure that are to be substituted when the procedure is called. COS uses values supplied with the procedure call and default parameter values from the prototype statement to replace these strings.

An ampersand (&) must precede each parameter to be substituted (*substitution parameter*) within the definition body. If a parameter appears in the prototype but is not preceded by an ampersand in the body, substitution does not occur.

&DATA - Procedure data

Data may be included within the procedure definition body after the procedure data card.

The *dn* parameter creates a temporary dataset composed of the data identified in the procedure, including any substitutions resulting from the call. This temporary dataset allows programs such as CAL or CFT to use it as source data.

Format:

```
&DATA,dn.
```

dn Name of dataset to contain the data that follows; this is a required parameter.

The initial separator for an &DATA statement can be a blank, comma, or an open parenthesis; the statement terminator can be a blank, period, or a close parenthesis.

An &DATA specification cannot be continued to subsequent cards. All card images following an &DATA card up to the next &DATA card are written to the specified dataset after string substitution is performed. See example 7 later in this section.

ENDPROC - End procedure definition

The ENDPROC control statement indicates the end of an in-line procedure definition block. ENDPROC is a system verb.

Format:

```
ENDPROC.
```

Parameters: none

SUBSTITUTION PARAMETERS

A character string that is eligible for substitution is listed in the prototype statement as a *formal parameter specification*. This name, when preceded by an ampersand in the definition body, indicates that a value is to be substituted during procedure invocation.

COS replaces the ampersand and parameter name with its selected value. If the parameter listed in the prototype statement is not preceded by an ampersand in the body, substitution does not occur. If two ampersands precede the string, one is removed and substitution is inhibited.

Any string consisting of one through eight characters may be selected for substitution. Character strings to be substituted are delimited by any character other than numerals, alphabetics, commercial at (@), dollar sign ($), and the percent sign (%). An ASCII underline is used as a string delimiter when the next character is one of these characters. See example 3 later in this section. COS deletes the underline after evaluating the string it delimits. Thus, the underline concatenates the strings it delimits.

VALUE SUBSTITUTION

When a statement in the current control statement file calls a procedure, COS searches the definition body for the character strings preceded by ampersands. For each occurrence, it substitutes the values supplied by either the calling statement or the prototype statement.

In the prototype statement, parameters may be in positional or keyword format.

Positional parameters

Positional formal parameters allow the user to list the strings within the body that can be substituted. The calling statement lists values to be substituted for these strings in the same order in which they are listed in the prototype statement. The value supplied with the calling statement is substituted for every occurrence of the corresponding formal positional parameter within the definition body. If the caller passes too few positional parameters, null strings are substituted for the remaining formal positional parameters. If too many positional parameters are passed, the procedure call is in error and the job aborts.

Keyword parameters

Keyword formal parameters are listed in any order after all positional parameters are given on the prototype statement and the calling statement. A keyword formal parameter allows the user to specify substitution values on the prototype statement that are to be used when one is not given on the calling statement.

If the keyword formal parameter is included in the calling statement with a value, that value is substituted. If the entire keyword formal parameter is omitted from the calling statement, the *default value* on the prototype statement is substituted. If a default value is not provided on the prototype statement, the character string within the body corresponding to that formal parameter is not included in the procedure expansion.

If only the keyword portion of the keyword formal parameter (the character string itself) is included in the calling statement, without a value assigned to it, then a *keyed default value* from the prototype statement is substituted. If a keyed default value is not provided on the prototype statement, again the character string within the body corresponding to that formal parameter is not included in the procedure expansion.

A keyword parameter enclosed in apostrophes ('*KEY*'=*value*) is considered a positional parameter.

Positional and keyword parameters

When supplying both positional and keyword parameters, all positional parameters must precede all keyword parameters; COS evaluates the call's positional parameters first. The end of the caller's list of positional parameters is signaled by the appearance of a keyword parameter, statement terminator, or by specifying all positionals.

Apostrophes and parentheses

Sometimes parameter values in a procedure definition or a procedure calling statement require a special format. If a literal string (a string delimited with apostrophes) appears in either of these statements, it is processed as though it were a literal constant. That is, all apostrophes in the value remain when the value is substituted. See example 5 later in this section.

To avoid any possibility of erroneous processing, use parentheses as string delimiters in these statements. Outermost parentheses preceded by the initial, parameter, equivalence, or concatenation separators are removed during value substitution which delays processing of any separator characters in the string until the statement itself, with substituted values, is processed.

This delay is also required when specifying multiple values for the default value and/or keyed default value parameters on a procedure definition statement. See examples 1, 2, 4, and 6. Parentheses are advised in the procedure calling statement when the use of the value in the procedure statements is unknown. See examples 4, 5, and 6 later in this section.

EXAMPLES

The following examples explain the COS control statement procedure substitution process.

Example 1:

Consider a single statement procedure called LOAD which is defined as follows:

Definition

```
PROC.
LOAD,NOGO=:NX,LIBRARY=($FTLIB:$SYSLIB):MYLIB.  Prototype statement
LDR,&NOGO,LIB=&LIBRARY.                        Definition body
ENDPROC.
```

The prototype statement in this example defines two formal parameters, both of which are in keyword format. The keyword NOGO has a null value when omitted from the calling statement and a value of NX when included on the calling statement in keyword-only format. The keyword LIBRARY has the default value of $FTLIB:$SYSLIB. When LIBRARY is used in the calling statement without a value, the keyed default value, MYLIB, is substituted.

When the LOAD procedure is invoked, it expands to a single statement whose form depends on the choice of parameters:

Invocation

```
LOAD,NOGO.
LOAD.
LOAD,LIBRARY=THISLIB.
LOAD,LIBRARY,NOGO.
```

Expansion

```
LDR,NX,LIB=$FTLIB:$SYSLIB.
LDR,,LIB=$FTLIB:$SYSLIB.
LDR,,LIB=THISLIB.
LDR,NX,LIB=MYLIB.
```

Example 2:

The following in-line procedure definition creates a procedure called BLDABS.

Definition

```
PROC.
BLDABS,SOURCE,LIST,GO='NO':'YES',LIB= ∧
   :($SYSLIB:$FTLIB),MAP=FULL:PART.           Prototype statement
REWIND,DN=$BLD:&SOURCE.
CAL,I=&SOURCE,L=&LIST,ABORT.
LDR,NX,LIB=&LIB,MAP=&MAP,L=&LIST.
REWIND,DN=$ABD:&LIST.                         Definition body
SAVE,DN=$ABD,PDN=MYPROGRAM.
IF(&GO.EQ.'YES')
$ABD.
ENDIF.
ENDPROC.
```

Invocation

```
BLDABS,WORK,,GO,LIB=VLIB2.
```

Expansion

```
REWIND,DN=$BLD:WORK.
CAL,I=WORK,L=,ABORT.
LDR,NX,LIB=VLIB2,MAP=FULL,L=.
REWIND,DN=$ABD:.
SAVE,DN=$ABD,PDN=MYPROGRAM.
IF('YES'.EQ.'YES')
$ABD.
ENDIF.
```

Example 3:

This procedure exemplies the proper use of the underscore character for the definition of a formal parameter. It creates a procedure called AUDJCL.

Definition

```
PROC.
AUDJCL,DN,LEVEL,L=$OUT:AUDLST.                Prototype statement
AUDIT,PDN=&DN&LEVEL_JCL,ID=JCL,L=&L.          Definition body
ENDPROC.
```

```
Invocation                    Expansion

AUDJCL,-,05.                  AUDIT,PDN=-05JCL,ID=JCL,L=$OUT.
```

Example 4:

Parentheses are required when specifying multiple values for a single parameter value on a procedure definition prototype statement or on a calling statement. In these cases, the colon is used to separate default and Boolean values in a keyword parameter. For example:

Procedure-definition prototype statement:

```
MYPROC,POS1,KEY=(DEF1:DEF2):(B001:B002).
```

Invocation:

```
MYPROC,(POS1A:POS1B).
```

When substitution occurs during this call, POS1A:POS1B replaces all POS1 occurrences within the definition body. Both values (POS1A and POS1B) are evaluated separately during control statement evaluation. If apostrophes are on the call, 'POS1A:POS1B' is evaluated as one literal string.

Example 5:

The following procedure definition exemplifies the use of literal strings instead of parenthetical strings.

Definition

```
PROC.
PURGER,PDN,ID,ED,M.                                          Prototype
ACCESS,DN=$PURGE,PDN=&PDN,ID=&ID,ED=&ED,M=&M,UQ,NA.)
DELETE,DN=$PURGE,NA.                                 }Definition body
ENDPROC.
```

Invocation

```
PURGER,'SOURCE.MAIN',PROJECT.
```

Expansion

```
ACCESS,DN=$PURGE,PDN='SOURCE.MAIN',ID=PROJECT,ED=,M=,UQ,NA.
DELETE,DN=$PURGE,NA.
```

The apostrophes remain as part of the string in the expansion. If parentheses had been used in the invocation instead of apostrophes for the permanent dataset name, (SOURCE.MAIN), the value when the ACCESS statement is evaluated would be SOURCE.MAIN because the outermost parentheses are removed when preceded by a valid separator. This action would cause an error because the period in SOURCE.MAIN would be evaluated as a statement terminator during evaluation.

Example 6:

The following example illustrates the use of parenthetical strings instead of literal strings in a procedure definition.

Definition

```
PROC.
LGO,CALSORC,ABS,NLIB=$SCILIB:($SCILIB: ∧
  $SYSLIB:$FTLIB).                              Prototype
CAL,I=&CALSORC.                               }
LDR,NX,AB=&ABS,NOLIB=&NLIB.                   } Definition body
ENDPROC.
```

Invocation

```
LGO,,,NLIB.
```

Expansion

```
CAL,I=.
LDR,NX,AB=,NOLIB=$SCILIB:$SYSLIB:$FTLIB.
```

Parentheses were not included for the expansion of the NLIB keyed default value because parentheses are removed during processing when preceded by the concatenation delimiter (:).

If apostrophes had been used instead of parentheses for the NLIB parameter value, the colons would have been ignored as separators during expansion. Also, apostrophes are treated as part of the value when included in a procedure definition prototype statement or a calling statement. Therefore, if apostrophes had been used, the following expansion would have occurred.

```
CAL,I=.
LDR,NX,AB=,NOLIB='$SCILIB:$SYSLIB:$FTLIB'.
```

When the LDR statement is executed, the value assigned to the NOLIB parameter would be the literal string $SCILIB:$SYSLIB:$FTLIB which violates the syntax for the NOLIB parameter.

Example 7:

Consider the following procedure definition. This procedure is used to retrieve specified source decks from an UPDATE program library by the use of the &DATA option.

```
PROC.
FETCH,PLNAME,MASTERCH,DECKRNGE.            Prototype statement
ACCESS,DN=&PLNAME.
UPDATE,I=QZRRZQ2,Q,C=0,S,P=&PLNAME.
RELEASE,DN=QZRRZQ2:&PLNAME.                Definition body
&DATA QZRRZQ2
&MASTERCH_COMPILE &DECKRNGE
ENDPROC.
```

Two sample invocations and their expansions follow:

Invocation	Expansion
FETCH,COSPL,*,(ST,CT).	ACCESS,DN=COSPL. UPDATE,I=QZRRZQ2,Q,C=0,S,P=COSPL. RELEASE,DN=QZRRZQ2:COSPL. (Dataset QZRRZQ2 contains: *COMPILE ST,CT)
FETCH,FTLIBPL,*,(COS.RFD).	ACCESS,DN=FTLIBPL. UPDATE,I=QZRRZQ2,Q,C=0,S,P=FTLIBPL. RELEASE,DN=QZRRZQ2:FTLIBPL. (Dataset QZRRZQ2 contains: *COMPILE COS.RFD)

Example 8:

Ths example illustrates one mechanism for defining and maintaining user procedure libraries.

```
    ACCESS,DN=GENLIB.
    CALL,DN=GENLIB.
```

The permanent dataset GENLIB contains:

```
    ECHO,OFF.
    RELEASE,DN=$PROC.
    *.
    *.        Define procedure for ACCESS of commonly used ID.
    *.
    PROC.
    UQ,DN,ED=:1,PDN=:GENLIB,R=:READCW,W=:WRITECW,M=:MAINCW,NA=:NA.
    ACCESS,DN=&DN,ID=MYUID,PDN=&PDN,ED=&ED,R=&R,W=&W,M=&M,NA=&NA.
    RETURN.
    EXIT.
    RETURN,ABORT.
    ENDPROC.
    *.
    *.        Edit a local dataset.
    *.
    PROC.
    ED,DN,AC=:'ACCESS'.
    IF('&AC'.EQ.'ACCESS')
      UQ,&DN.
    ENDIF
    TEDI,DN=&DN.
    RETURN.
    EXIT.
    RETURN,ABORT.
    ENDPROC.
    *.
    *.        End of definitions
    *.
    UQ,PROCLIB,NA.
    SAVE,DN=$PROC,PDN=PROCLIB,ID=MYUID.
    DELETE,DN=PROCLIB,NA.
    RELEASE,DN=$PROC.
    ACCESS,DN=PROCLIB,ID=MYUID.
    LIBRARY,DN=*:PROCLIB.
    ECHO,ON.
```

CONDITIONAL CONTROL STATEMENT PROCESSING

The control statements IF, ELSE, ELSEIF, and ENDIF allow control statements to be placed in a conditional block structure. A conditional block must begin with an IF statement and conclude with an ENDIF statement. In addition to these two statements, it contains a control statement sequence that is processed only if the IF expression is true.

Optional control statement sequences can be included within a conditional block using the ELSEIF and ELSE statements. If the result of an IF or ELSEIF expression evaluation is true, the control statement sequence that follows is processed and subsequent ELSE or ELSEIF conditions, even if true, are not processed. If the expression evaluates as false, the control statement sequence that follows is skipped. If all such sequences are skipped (all expression evaluations yield false), the sequence following the ELSE statement (if it exists) is processed.

The conditional block is first scanned to verify the validity of the block's syntax. If there are any syntax errors, the block is skipped without being evaluated and a job step abort error occurs. This means that any EXIT control statements within the conditional block are ignored when there is a syntax error in that conditional block. This validation occurs when the control statement file in which it is contained iş invoked.

ELSEIF and ELSE sequences are optional. Within a conditional block, only one ELSE sequence is permitted and it must be the last one in the block. There is no limit to the number of ELSEIF sequences that may be used in a conditional block.

Null blocks (for example, an ELSE statement immediately following an ELSEIF) are ignored without comment.

Conditional blocks can be constructed in the following ways:

- Conditional block
- Conditional block with ELSE
- Conditional block with ELSEIF(s)
- Conditional block with ELSE and ELSEIF(s)

CONDITIONAL BLOCK

The basic format of a conditional block, figure 2-2, begins with an IF statement and ends with an ENDIF statement. When the IF statement expression is true, the control statement sequence that follows is processed. If the expression is false, the control statement sequence is not processed.

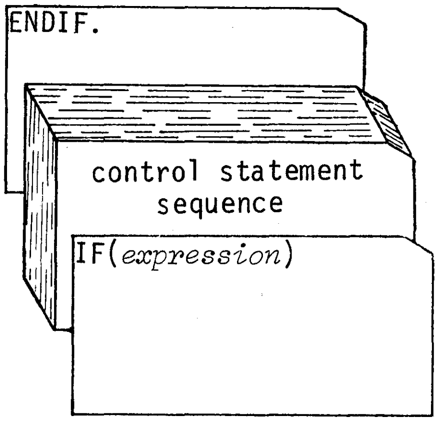


Figure 2-2. Basic conditional block structure

IF - Begin conditional block

The IF control statement defines the beginning of a conditional block. Each IF control statement must have a corresponding ENDIF control statement. IF is a system verb.

Format:

```
IF(expression)
```

Parameters:

expression
A valid JCL expression (part 1, section 4). This parameter is required.

ENDIF - End conditional block

The ENDIF control statement defines the end of a conditional block. ENDIF is a system verb.

Format:

```
ENDIF.
```

Parameters: none

Example:

Following is an example of the conditional block structure.

```
ACCESS,DN=MYPROG.
MYPROG.
EXIT.
IF(ABTCODE.NE.21)
  *.
  *.      UNEXPECTED JOB STEP ABORT ERROR
  *.
  EXIT.
ENDIF.
```

In this example, if the ACCESS request or execution of MYPROG fails, the conditional block after the EXIT control statement is processed. The conditional block determines if the job step abort occurred because a dataset was not found, in which case the processing of control statements resumes after the ENDIF control statement. If this is not the reason for the abort, the job terminates with the EXIT control statement.

CONDITIONAL BLOCK WITH ELSE

The second conditional block structure includes the ELSE control statement. The control statement sequence is processed if the expression on the IF statement is true. If the expression is not true, the sequence following the ELSE statement is processed. The block structure is illustrated in figure 2-3.

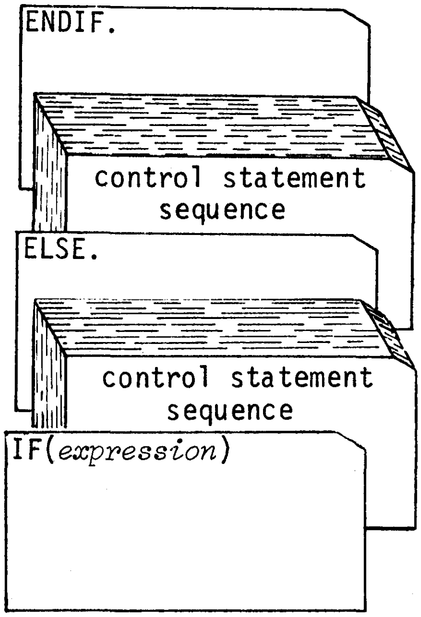


Figure 2-3. Conditional block structure including ELSE

ELSE - Define alternate condition

The ELSE control statement is used to define an alternate condition. An IF statement, as well as any ELSEIF statements (see Conditional Block with ELSE and ELSEIF), must precede the ELSE control statement. If all conditions specified by the IF and ELSEIF statements that precede the ELSE in the conditional block test as false, then the sequence of statements that follow the ELSE statement is executed. ELSE is a system verb.

Format:

```
ELSE.
```

Parameters: none

Example:

An example of a conditional block structure using the ELSE statement follows.

```
ACCESS,DN=INITJCL.
ACCESS,DN=MYPROG.
ACCESS,DN=PROG.
PREPROG.
IF(JSR.NE.0)
  CALL,DN=INITJCL.
  SWITCH,1=ON.
ELSE.
  SWITCH,1=OFF.
ENDIF.
PROG.
```

After PREPROG is executed, the conditional block determines if PREPROG has successfully executed (by its setting of JSR). The procedure INITJCL is executed and a sense switch is set if the status was bad; the sense switch is cleared if PREPROG executed properly.

CONDITIONAL BLOCK WITH ELSEIF

The third conditional block structure, shown in figure 2-4, includes one or more ELSEIF statements. Each logical expression on the IF and ELSEIF statements is tested in sequence until a true condition is found; then the corresponding control statement sequence is processed.

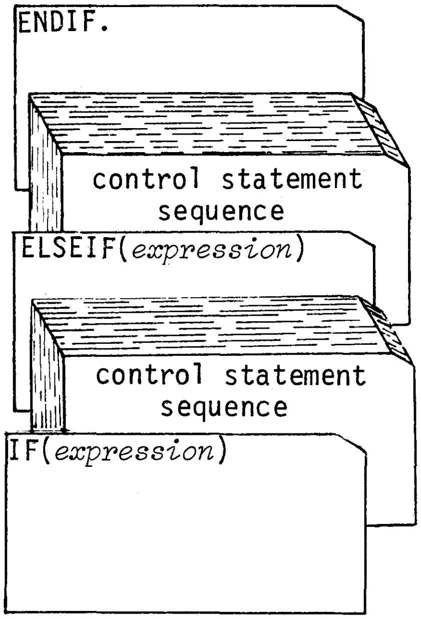


Figure 2-4. Conditional block structure including ELSEIF

ELSEIF - Define alternate condition

The ELSEIF control statement defines an alternate condition to test if the previous one tested was false. The sequence of statements following the ELSEIF statement is executed when the ELSEIF expression is true. All ELSEIF control statements must precede the optional ELSE control statement for a conditional block. An ELSEIF statement without a previously processed IF statement results in a job step abort. ELSEIF is a system verb.

Format:

```
ELSEIF(expression)
```

Parameters:

expression
Any valid JCL expression (part 1, section 4). This parameter is required.

A conditional block can contain any number of ELSEIF control statements. The block of control statements following an ELSEIF statement is processed under the following conditions:

- The expression for the IF statement is false,
- All preceding ELSEIF statement expressions are false, and
- The ELSEIF expression is true.

Example:

An example of a deck including the ELSEIF statement is:

```
IF(SYSID.EQ.'COS 1.07')
  ACCESS,DN=$FTLIB,ID=V107.
ELSEIF(SYSID.EQ.'COS 1.08')
  ACCESS,DN=$FTLIB,ID=V108.
ELSEIF(SYSID.EQ.'COS 1.09')
  ACCESS,DN=$FTLIB,ID=V109.
ENDIF.
LDR,NOLIB,LIB=$FTLIB.
```

This conditional block tries to access the correct version of the FORTRAN library, $FTLIB, for the execution of the loader that follows the conditional block.

CONDITIONAL BLOCK WITH ELSE AND ELSEIF

The conditional block structure in figure 2-5 uses ELSEIF and the ELSE statements. A block can contain any number of ELSEIF statements but can contain only one ELSE, which must be the last conditional statement before the ENDIF.

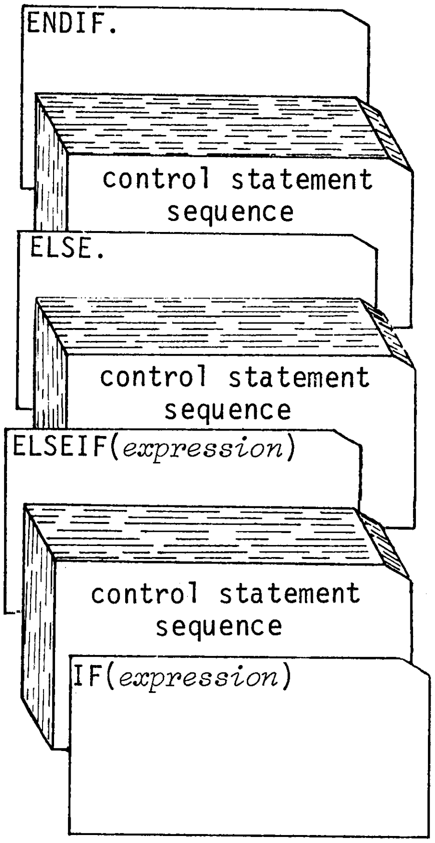

ENDIF.
control statement sequence
ELSE.
control statement sequence
ELSEIF(*expression*)
control statement sequence
IF(*expression*)

Figure 2-5. Conditional block structure including ELSEIF and ELSE

The ELSE control statement sequence in this case is processed only if:

- The expression on the IF statement is false, and
- All ELSEIF statement expressions are also false.

Example:

An example of this type of conditional block structure follows.

```
IF(TIMELEFT.GT.175)
  IF(SYSID.EQ.'COS 1.08')
    ACCESS,DN=$FTLIB,ID=V108.
  ELSEIF(SYSID.EQ.'COS 1.09')
    ACCESS,DN=$FTLIB,ID=V109.
  ELSE.
    *.
    *.  CURRENT SYSTEM LEVEL NOT RECENT ENOUGH
    *.
    EXIT.
  ENDIF.
  LDR,NOLIB,LIB=$FTLIB.
  SET,J1='YES'L.
ELSE.
  SET,J1='NOTIME'L.
ENDIF.
IF(J1.EQ.'YES'L)
  DISPOSE,DN=RESULTS,DC=ST.
ELSE.
  *.
  *.  JOB DID NOT RUN TO NORMAL COMPLETION
ENDIF.
EXIT.
```

This example is an expansion of the example for the third format and allows execution of the compiled program if there is enough time left and if the correct library is accessible. On a successful run, the dataset called RESULTS is disposed as a staged dataset.

ITERATIVE CONTROL STATEMENT PROCESSING

An iterative block, figure 2-6, contains a control statement sequence that is to be processed more than once during the processing of a job. It includes the LOOP, EXITLOOP, and ENDLOOP statements. Nesting can occur to any level. The EXITLOOP statement indicates the normal exit condition for the loop. If its expression is true, the loop is exited; if it is false, loop execution continues with the subsequent statements. Control returns to the beginning of the loop when the ENDLOOP statement is encountered.

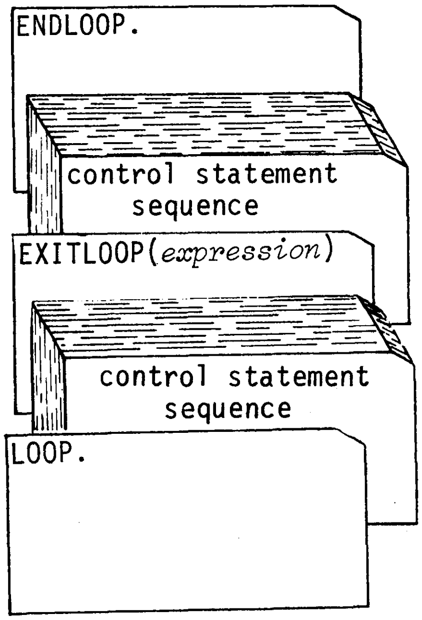


Figure 2-6. Iterative block structure

Iterative blocks are prescanned for syntax errors before actual processing begins. Any errors in the block structure cause a skipping of that block followed by a job step abort. If an iterative block is included within a conditional block, it must be totally contained within that block.

LOOP - BEGIN ITERATIVE BLOCK

The LOOP control statement is required to define the beginning of an iterative block. An ENDLOOP control statement is required in the same procedure dataset to terminate the iterative block. LOOP is a system verb.

Format:

```
LOOP.
```

Parameters: none

ENDLOOP - END ITERATIVE BLOCK

The ENDLOOP control statement terminates an iterative control statement block. If an ENDLOOP control statement occurs in a procedure dataset without a preceding LOOP statement, a job step abort occurs. Execution of the ENDLOOP statement results in control being passed to the preceding LOOP statement which begins another iteration of the loop.

Format:

```
ENDLOOP.
```

Parameters: none

EXITLOOP - END ITERATION

The EXITLOOP control statement defines the condition(s) under which the control statement block iteration is to end. If its expression is true, the loop is exited; if it is false, the control statements which follow are executed.

An EXITLOOP statement that appears outside of an iterative block causes a job step abort. When nesting iterative control statement blocks, the EXITLOOP control statement defines the exit conditions for only the most immediate iterative block. EXITLOOP is a system verb.

Format:

```
 EXITLOOP.

    or

EXITLOOP(expression)
```

Parameters:

expression
: Optional valid JCL expression (part 1, section 4). If omitted, an unconditional exit from the iterative block occurs.

Example:

The following example merges the two datasets DSIN1 and DSIN2 for 60 records.

```
SET,J1=0.
SET,J2=60.
LOOP.
  EXITLOOP(J2.EQ.0)
  IF(J1.EQ.0)
          COPYR,I=DSIN1,O=OUTDS.
          SET,J1=1.
  ELSE.
          COPYR,I=DSIN2,O=OUTDS.
          SET,J1=0.
  ENDIF.
  SET,J2=J2-1.
ENDLOOP.
REWIND,DN=DSIN1:DSIN2:OUTDS.
```

DATASET DEFINITION AND CONTROL 3

Datasets can be defined and managed by the user via three dataset control statements: ASSIGN, ACCESS, and RELEASE.

- ASSIGN creates a dataset on mass storage and assigns dataset characteristics for tape and disk.
- ACCESS (defined in part 2, section 4) makes an existing disk or tape permanent dataset local to a job or can be used to create a dataset on magnetic tape; ASSIGN assigns tape dataset characteristics.
- RELEASE relinquishes access to the named dataset for the job.

ASSIGN - ASSIGN MASS STORAGE OR MAGNETIC TAPE DATASET CHARACTERISTICS

The ASSIGN control statement creates a mass storage dataset and assigns dataset characteristics for tape and mass storage. If an ASSIGN is used for dataset creation, it must appear prior to the first reference to the dataset; otherwise, the characteristics are defined at the first reference. If an ASSIGN is used for a tape dataset, it must follow the tape ACCESS request. ASSIGN§ is a system verb.

Format:

```
ASSIGN,DN=dn,S=size,BS=blk,DV=ldv,DT=dt,DF=df,
                                   RDM,U,MR,LM=lm,DC=dc,BFI=bfi,A=un.
```

Parameters are in keyword form. The only required parameter is DN.

DN=*dn* Local dataset name. 1-7 alphanumeric characters, the first of which is A-Z, $, %, or @; remaining characters may also be numeric. DN is a required parameter.

§ ASSIGN does not create a dataset which the CFT 1.10 OPEN statement recognizes as existing.

S=*size* Dataset size. Octal number of sectors (1000_8-word blocks) to be reserved for the dataset. If the dataset size is not given, the disk space for the dataset is dynamically allocated as needed. This parameter applies to mass storage datasets only and is ignored when used for magnetic tape datasets.

BS=*blk* Buffer size. Number of 1000_8-word blocks to be reserved for user buffer. The default number of blocks is set by an installation parameter. BS generates an error if the U parameter is specified (indicating unblocked dataset structure).

DV=*ldv* Logical device on which dataset is to begin. If a logical device name is not given, one is chosen by the system. Consult the on-site analyst for possible logical device names. This parameter applies to mass storage datasets only and is ignored when used for magnetic tape datasets.

DT=*dt* Device type. The allowable device types are CRT and MS. MS is the default. This parameter applies to mass storage datasets only and is ignored when used for magnetic tape datasets.

DF=*df* Dataset format. This parameter is used only on output; it is valid only when DT=CRT. This parameter applies to mass storage datasets only and is ignored when used for magnetic tape datasets. Two formats are supported:

CB Character blocked. End-of-record RCWs are converted to line feeds. This is the default.

TR Transparent. End-of-record RCWs are not converted to line feeds. The user is responsible for inserting line feeds.

RDM Random dataset. If the RDM parameter is present, the dataset is to be accessed randomly. If the RDM parameter is not specified, the dataset is accessed sequentially. This parameter applies to mass storage datasets only and is invalid for magnetic tape datasets.

U Unblocked dataset structure. If the U parameter is present, the dataset is not in COS-defined blocked format. If the U parameter is absent, the dataset is a COS blocked dataset. (See part 1, section 2 for information on unblocked dataset format.) This parameter is invalid for interchange format tape datasets.

MR Memory resident dataset. If this parameter is present, the system I/O routines write the buffers to the disk only if they become full. If the MR parameter is absent, the dataset is not a memory resident dataset. MR generates an error if the U parameter is specified. This parameter applies to mass storage datasets only and is invalid for magnetic tape datasets.

LM=*lm* Maximum size limit for this dataset. *lm* specifies a decimal count of 512-word blocks. The job step will be aborted if this size is exceeded. The default and maximum dataset size limits are set by an installation parameter. This parameter applies to mass storage datasets only and is ignored for magnetic tape datasets.

DC=*dc* Disposition code. Disposition to be made of the dataset at job termination. This parameter applies to mass storage datasets only and is ignored for tape datasets. The default is SC.

dc is a 2-character alpha code describing the destination of the dataset as follows:

IN The dataset is placed in the input queue of the default destination station.

ST Stage to mainframe. Dataset is made permanent at the mainframe of job origin.

SC Scratch dataset. Dataset is deleted.

PR Print dataset. Dataset is printed on printer at the mainframe of job origin.

PU Punch dataset. Dataset is punched on any card punch available at the mainframe of job origin.

PT Plot dataset. Dataset is plotted on any available plotter at the mainframe of job origin.

MT Magnetic tape. Dataset is written on magnetic tape at the mainframe of job origin.

BFI=*bfi* Blank field initiation. Octal representation of ASCII code which indicates the beginning of a sequence of blanks. BFI=OFF means that blank compression is inhibited. The default code is 33_8 (ASCII ESC code) but may be changed by an installation parameter.

A=*un* Unit name. Unit names allow the user to refer to a dataset from a FORTRAN program. Each unit name is 4 characters in the form FTxx, where xx is the unit number specified. The unit number is an integer value in the range 0-102. However, because unit numbers 100, 101, and 102 are reserved for system use, a user may designate unit numbers 0-99.

Use of this parameter associates the designated unit with the dataset specified by the DN parameter. At job initiation, unit FT05 is associated with dataset $IN and unit FT06 is associated with dataset $OUT. Unit names should not be used as dataset names.

NOTE

If a dataset is used in place of a unit name or vice versa, FORTRAN '77 auxiliary statements (that is, OPEN, CLOSE, and INQUIRE) may produce unpredictable results.

RELEASE - RELEASE DATASET

The RELEASE control statement relinquishes access to the named datasets for the job. If a dataset is not permanent and its disposition code is SC (scratch), the mass storage assigned to the dataset is released to the system. If the dataset is to be staged, the dataset is entered in the output queue for staging to the default destination station. An end-of-data is written to a permanent dataset when it is released if the dataset is blocked sequential and the previous operation was a write.

Format:

RELEASE,DN=dn_1:dn_2:...:dn_8,HOLD§.

§ Deferred implementation

Parameters:

DN=dn_i Name of dataset to be released. A maximum of eight datasets may be specified.

HOLD[§] Hold generic device; do not return it to the system pool. This parameter applies to magnetic tape datasets only and is ignored for mass storage datasets.

[§] Deferred implementation

PERMANENT DATASET MANAGEMENT 4

Permanent dataset management provides methods for creating, protecting, and accessing datasets assigned permanently to mass storage or magnetic tape. Such datasets cannot be destroyed by normal system activity, deadstarting, restarting, or engineering maintenance.

The user can manage user permanent datasets only; system permanent datasets are not directly accessible by the user. (See part 1, section 2 for a description of the types of datasets.)

The user manages user mass storage and magnetic tape permanent datasets by sending requests to the system through the control statements described below. Mass storage datasets are controlled by the Permanent Dataset Manager using the CRAY-1 resident Dataset Catalog (DSC); magnetic tape datasets are processed by the Tape Queue Manager (TQM).

- SAVE enters a dataset's identification and location in a system-maintained Dataset Catalog. Datasets recorded in the Dataset Catalog via a user SAVE request are user permanent datasets and are recoverable at deadstart. SAVE applies to mass storage datasets only; it is ignored for tape datasets.

- ACCESS causes a user permanent dataset to be assigned (made local) to a job. The usage (reading or writing, for example) of a dataset is determined by permissions granted when the dataset is accessed. ACCESS is also used to create a dataset on magnetic tape or to obtain an existing one.

- ADJUST changes the size of a user permanent dataset in the Dataset Catalog. ADJUST applies to mass storage datasets only; it is ignored for tape datasets since their size is automatically changed when the output tape dataset is closed.

- MODIFY changes established information for an existing user permanent dataset in the Dataset Catalog. MODIFY applies to mass storage datasets only; it is ignored for tape datasets.

- DELETE causes removal of a saved dataset from the Dataset Catalog.

SAVE - SAVE PERMANENT DATASET

The SAVE control statement makes a local dataset permanent. Saving a dataset consists of making an entry in the DSC. A permanent dataset is uniquely identified by permanent dataset name, user identification, and edition number. SAVE is a system verb. The SAVE statement is ignored when used for magnetic tape datasets.

SAVE has a twofold function:

- Creation of an initial edition of a permanent dataset
- Creation of an additional edition of a permanent dataset

A maintenance control word controls the creation of additional editions of an existing permanent dataset. Thus, to create a subsequent edition of an existing permanent dataset, the user must match the maintenance control word of the oldest existing edition. Read and write control words specified on the oldest existing edition of a permanent dataset apply to all subsequent editions of that dataset.

Under the appropriate conditions, SAVE forces any unwritten data to disk to ensure that all of the dataset is made permanent. Since this situation occurs when the dataset has been recently written to but not yet closed, SAVE will attempt to close the dataset. The specific conditions which the dataset must meet are described under the SAVE macro (part 3, section 4).

Format:

```
SAVE,DN=dn,PDN=pdn,ID=uid,ED=ed,RT=rt,R=rd,W=wt,

                          M=mn,UQ,NA,EXO{=ON }.
                                        {  OFF}
```

Parameters are in keyword form; the only required parameter is DN.

DN=*dn* Name of a dataset that is local to the job. This dataset may be closed before the dataset is made permanent.

PDN=*pdn* Permanent dataset name. 1-15 characters assigned by the dataset creator. This is the name that is saved by the system. Default value is *dn*.

ID=*uid* User identification. 1-8 alphanumeric characters assigned by the dataset creator. The default is no user ID.

ED=*ed* Edition number. A value from 1-4095 assigned by the dataset creator. The default value is:

- One, if a permanent dataset with the same PDN and ID does not exist, or
- The current highest edition number plus one, if a permanent dataset with the same PDN and ID does exist.

RT=*rt* Retention period. User-defined value from 0-4095 specifying the number of days a permanent dataset is to be retained by the system. The default value is an installation-defined parameter.

R=*rd* Read control word. 1-8 alphanumeric characters assigned by the dataset creator. The read control word of the oldest existing edition of a permanent dataset applies to all subsequent editions of that dataset. The default is no read control word.

W=*wt* Write control word. 1-8 alphanumeric characters assigned by the dataset creator. The write control word of the oldest existing edition of a permanent dataset applies to all subsequent editions of that dataset. To obtain write permission, the user must also have unique access (UQ) to that dataset. The default is no write control word.

M=*mn* Maintenance control word. 1-8 alphanumeric characters. The maintenance control word must be specified if a subsequent edition of the same permanent dataset is saved. The default is no maintenance control word.

UQ Unique access. If the UQ parameter is specified, only this job may access the permanent dataset at the completion of the SAVE function. Otherwise, multiuser access to the permanent dataset is granted.

NA No abort. If this parameter is omitted, an error causes the job to abort.

$\text{EXO}\left\{\begin{matrix}\text{=ON}\\ \text{OFF}\end{matrix}\right\}$ Execute-only dataset. This parameter sets or clears the execute-only status of the dataset. EXO only or EXO=ON causes the dataset to be saved as execute-only. EXO=OFF or omission of this parameter causes the dataset to be saved as a non-execute-only dataset.

NOTE

When processing for the SAVE request is complete, all forms of examination of this dataset are prohibited if EXO=ON.

ACCESS - ACCESS PERMANENT DATASET

The ACCESS control statement makes an existing permanent dataset local to a job and can be used to create a tape dataset. Following the ACCESS statement, all references to the permanent dataset must be by the local dataset name specified by the DN parameter. ACCESS assures that the user is authorized to use the permanent dataset. The ACCESS control statement must precede the ASSIGN control statement or the open call for the dataset. ACCESS is a system verb.

The user need not access a permanent dataset entered into the System Directory (SDR). A tape dataset cannot reside in the SDR. A basic set of datasets is entered into the System Directory when the operating system is installed. These datasets include the loader, the CFT compiler, the CAL assembler, UPDATE, BUILD, and system utility programs such as copies and dumps. Other datasets can be entered into the System Directory according to site requirements.

Format:

ACCESS,DN=*dn*,PDN=*pdn*,ID=*uid*,ED=*ed*,R=*rd*,W=*wt*,M=*mn*,UQ,LE,NA,

CS=*cs*,DF=*df*,DT=*dt*,FSEC=*fsec*,LB=*lb*,MBS=*mbs*,NEW,XDT=*yyddd*,

RT=*rt*,VOL=vol_1:vol_2:...vol_n,CT=*ct*,RF=*rf*,RS=*rs*.

Parameters are in keyword form; DN is the only required parameter for mass storage datasets to make an existing permanent dataset local to a job.

DN=*dn* Local dataset name by which the permanent dataset is to be known. This is a required parameter.

PDN=*pdn* Name of a permanent dataset being accessed and already existing in the system. The default value is *dn*. The name can be 1-15 characters for mass storage datasets; it can be 1-44 characters for tape datasets. For a labeled tape dataset, the right-most 17 characters of the PDN must match the file identification field of the HDR1 label.

ID=*uid* User identification. 1-8 alphanumeric characters. If *uid* was specified at SAVE time, the ID parameter must be specified on the ACCESS control statement. The default is no user ID. This parameter applies to mass storage datasets only; it is ignored for magnetic tape datasets.

ED=*ed* Edition number of permanent dataset being accessed; a value from 1-4095 was assigned by the dataset creator. If the ED parameter is not specified, the default is the highest edition number known to the system (for this permanent dataset). This parameter applies to mass storage datasets only; it is ignored for magnetic tape datasets.

R=*rd* Read control word as specified at SAVE time. 1-8 alphanumeric characters assigned by the dataset creator. The read control word of the oldest existing edition of a permanent dataset applies to all subsequent editions of that dataset. The default is no read control word. To obtain read permission, this parameter must be specified on the ACCESS control statement if a read parameter was specified when the dataset was saved. This parameter applies to mass storage datasets only; it is ignored for magnetic tape datasets.

W=*wt* Write control word as specified at SAVE time. To obtain write permission, this parameter must be specified in conjunction with a UQ parameter on the ACCESS control statement if a W parameter was specified when the dataset was saved. This parameter is required prior to an ADJUST and applies to mass storage datasets only; it is ignored for magnetic tape datasets.

M=*mn* Maintenance control word as specified at SAVE time. This parameter is specified in conjunction with a UQ parameter on an ACCESS control statement if the dataset is to be subsequently deleted. That is, maintenance permission is required to delete a dataset. This parameter applies to mass storage datasets only; it is ignored when used for magnetic tape datasets.

UQ Unique access. If the UQ parameter is specified and the appropriate write or maintenance control words are specified, then write, maintenance, and/or read permission may be granted. If UQ is not specified, then multiuser read access is granted by default (if at a minimum, the read control word is specified). UQ is required to delete a permanent dataset using the DELETE control statement. This parameter applies to mass storage datasets only; it is ignored for magnetic tape datasets.

LE[§] Lowest edition number. If the LE parameter is specified, the lowest edition number known to the system for this dataset is accessed. LE must not be equated with a value and cannot be specified in conjunction with the ED parameter. This parameter applies to mass storage datasets only; it is ignored when used for magnetic tape datasets.

NA No abort. If this parameter is omitted, an error causes the job to abort.

CS=*cs* Character set of tape dataset, for data only. This parameter applies only to tape datasets; it is ignored when used for mass storage datasets.

AS ASCII; default.
SL EBCDIC

DF=*df* Tape dataset format. This parameter applies only to tape datasets; it is ignored when used for mass storage datasets.

IC Interchange format
TR Transparent format; default.

[§] Deferred implementation

DT=*dt* Tape dataset generic device name or synonym. This parameter is required for tape datasets.

Generic Name	Synonym	Significance
*6250	*TAPE	Device capable of 6250 bpi
*1600		Device capable of 1600 bpi; also used to declare density when writing

FSEC=*fsec* File section number; a numeric field from 1 through 9999 that specifies the volume in the tape dataset. The first section (or volume) of a tape datatset is numbered 0001. The default is 1. For example, to access a tape dataset starting with the eighth section, specify FSEC=8 on the ACCESS call. This parameter applies only to tape datasets; it is ignored when used for mass storage datasets.

LB=*lb* Tape dataset label type. This parameter applies only to tape datasets; it is ignored when used for mass storage datasets.

BLP	Bypass label processing§
SL	IBM standard-labeled tapes
NL	Non labeled tapes; default
AL	ANSI standard labeled tapes

MBS=*mbs* Maximum tape block size; that is, the number of bytes in the largest tape blocks to be read or written. The maximum size allowed at the installation and the default are specified as installation parameters. This parameter applies only to tape datasets; it is ignored when used for mass storage datasets.

NEW Tape dataset is to be created; the dataset must be written starting at the beginning of information. This parameter applies only to tape datasets; it is ignored when used for mass storage datasets.

XDT=*yyddd* Expiration date. Indicates the date on which this tape dataset may be overwritten. *yy* specifies the year and is a number from 0-99. *ddd* specifies the day in the year and is a number from 001 through 366. This parameter applies only to tape datasets; it is ignored when used for mass storage datasets.

§ Deferred implementation

RT=*rt*§ Retention period. User-defined value from 1 through 4095 specifying the number of days a permanent dataset is to be retained by the system. Similar to the XDT parameter but allows the user to specify relative expiration date. The default value is an installation-defined parameter.

VOL=vol_i Volume identifier; a list of 6-character alphanumeric volume identifiers comprising the tape dataset. The maximum number of volume identifiers per dataset is an installation parameter.

The following tape dataset parameters specify that record and data format conversion are to be performed at run time on the tape dataset.

CT=*ct*§ Tape dataset conversion type. *ct* is a 3-character code describing the machine internal data representation.

IBM	IBM 370 and compatible internal data representation

This parameter is required if run-time record and data format conversion are performed; default is no conversion. This parameter applies only to tape datasets; it is ignored when used for mass storage datasets. Specifying this parameter converts data on the tape from 32-bit IBM internal representation to 64-bit internal CRAY-1 representation. Real numbers and integers are converted.

RF=*rf*§ Tape dataset record format. *rf* is a 1- to 8-character code describing the record type.

U	IBM U (undefined) format; default if CT=IBM
F	IBM F (fixed) format
FB	IBM FB (fixed block) format
V	IBM V (variable) format
VB	IBM VB (variable block) format
VBS	IBM VBS (variable blocked spanned) format

RS=*rs* Tape dataset record size. *rs* is the decimal length of the record expressed in units depending upon the conversion type; if CT=IBM, *rs* is the record size expressed as a decimal number of 8-bit byte units. If the *rf* parameter is F or FB, the RS parameter is required; if *rf* is V, VB, or VBS, the RS parameter is optional; if *rf* is U, the RS parameter is not pertinent as the U record format does not contain records.

§ Deferred implementation

ADJUST - ADJUST PERMANENT DATASET

The ADJUST control statement changes the size of a mass storage permanent dataset; that is, it redefines the size of the dataset. When a permanent dataset is overwritten, and the dataset size changes, issuing an ADJUST statement informs the system of the dataset's new size. An ADJUST of a permanent dataset may be issued if the dataset has been previously accessed within the job with write permission. ADJUST is a system verb.

Under the appropriate conditions, ADJUST forces any unwritten data to disk to ensure that all of the dataset is made permanent. Since this situation occurs when the dataset has been recently written to but not yet closed, ADJUST will attempt to close the dataset. The specific conditions that the dataset must meet are described under the ADJUST macro (see part 3).

The ADJUST statement is ignored when used with magnetic tape datasets.

Format:

```
ADJUST,DN=dn,NA.
```

Parameters:

DN=*dn* Local dataset name of a permanent dataset that has been accessed with write permission. This dataset may be closed before the ADJUST statement is processed.

NA No abort. If this parameter is omitted, an error causes the job to abort.

MODIFY - MODIFY PERMANENT DATASET

The MODIFY control statement changes permanent dataset information established by the SAVE function or a previously executed MODIFY function. A permanent dataset must be accessed with unique access (UQ) and all permissions before a MODIFY of a permanent dataset can be issued. MODIFY is a system verb.

Once a permanent dataset exists, the read, write, and maintenance control words apply to subsequent editions of that permanent dataset. Therefore, permission control words can be modified only for a permanent dataset having a single edition. MODIFY applies to mass storage datasets only; it is ignored for tape datasets.

Format:

```
MODIFY,DN=dn,PDN=pdn,ID=uid,ED=ed,RT=rt,R=rd,

                              W=wt,M=mn,NA,EXO{=ON }.
                                              { OFF}
```

Parameters are in keyword form; the only required parameter is DN.

DN=*dn* Local dataset name of a permanent dataset that has been accessed with all permissions. DN is a required parameter.

PDN=*pdn* New permanent dataset name to be applied to the existing dataset. If this parameter is omitted, the existing permanent dataset name is retained.

ID=*uid* New user identification, to be applied to the existing permanent dataset. 1-8 alphanumeric characters. If this parameter is omitted, the existing user ID is retained. If this parameter is present without a value, user identification is established as binary zeros.

ED=*ed* New edition number to be applied to the existing permanent dataset. If this parameter is omitted, the existing edition number is retained.

RT=*rt* New retention period to be applied to the existing permanent dataset. If this parameter is omitted, the current retention period is retained. If this parameter is present without a value, the retention period is set to the installation-defined value.

R=*rd* New read permission control word to be applied to the existing permanent dataset. If this parameter is omitted, the existing read permission is retained. If R is present without a value, read permission is established as binary zeros.

W=*wt* New write permission control word to be applied to the existing permanent dataset. If this parameter is omitted, the existing write permission is retained. If W is present without a value, write permission is established as binary zeros.

M=*mm* New maintenance permission control word to be applied to the existing permanent dataset. If this parameter is omitted, the existing maintenance permission is retained. If M is present without a value, maintenance permission is established as binary zeros.

NA No abort. If this parameter is omitted, an error causes the job to abort.

EXO$\left\{\begin{matrix}\text{=ON}\\\text{OFF}\end{matrix}\right\}$ Execute-only dataset. This parameter sets or clears the execute-only status of a dataset. EXO only or EXO=ON causes the dataset to be modified to execute-only. EXO=OFF causes the dataset to be modified to a non-execute-only dataset. If this parameter is omitted, the execute-only status of a dataset is unchanged.

NOTE

When processing for the MODIFY request is complete, all forms of examination of this dataset are prohibited if EXO=ON.

DELETE - DELETE PERMANENT DATASET

The DELETE control statement removes a mass storage permanent dataset from the Dataset Catalog (DSC). To issue a DELETE of a dataset, the job must have previously accessed the dataset with maintenance permission, if specified on the SAVE control statement, and unique access (UQ). The dataset remains a local dataset after DELETE until job termination. DELETE is a system verb.

Format:

```
DELETE,DN=dn,NA.
```

Parameters:

DN=*dn* Local dataset name of a permanent dataset accessed with maintenance permission and unique access

NA No abort. If this parameter is omitted, a fatal error causes the job to abort.

DATASET STAGING CONTROL 5

Two control statements support staging datasets between the CRAY-1 and a front-end system: ACQUIRE and DISPOSE. Another control statement, SUBMIT, directs datasets to the CRAY input queue.

- ACQUIRE obtains a front-end resident dataset, stages it to the CRAY-1, and makes it permanent and accessible to the job making the request. Alternatively, if the dataset is already permanent on CRAY-1 mass storage, ACQUIRE allows dataset access to the job making the request.

- DISPOSE directs a dataset to the specified queue for staging to a front-end system. DISPOSE can also be used to release a local dataset or to change dataset disposition characteristics.

- SUBMIT directs a dataset on CRAY-1 mass storage local to the submitting job to the CRAY-1 input queue.

Dataset control information such as save or access codes (required by a front-end system for management of its own files) can be sent by the CRAY-1 user to the front-end system through the use of TEXT, a special parameter of the ACQUIRE and DISPOSE statements. The contents of the character string provided with the TEXT parameter are defined by the front-end system.

ACQUIRE and DISPOSE are invalid with tape datasets because these two statements apply only to the staging of datasets between a front-end computer system and the Cray computer. No interface exists between the Station Call Processor (SCP) and the Tape Queue Manager (TQM). The tape subsystem is online to the CRAY-1 computer.

ACQUIRE - ACQUIRE PERMANENT DATASET

The ACQUIRE control statement allows the user to make a dataset permanent and accessible to the job making the request. ACQUIRE is a system verb.

When an ACQUIRE control statement is issued, COS determines if the requested dataset is front-end resident or permanently resident on CRAY-1 mass storage.

If the CRAY-1 Operating System determines that the requested dataset is already permanently resident on CRAY-1 mass storage, dataset access is granted to the job making the request.

If the requested dataset is not a CRAY-1 permanent dataset, the request for the dataset is sent to the front-end system. The front-end system stages the dataset to the CRAY-1. COS then makes the dataset permanent and grants dataset access to the job making the request. Until the dataset is made permanent, processing of the job making the request is delayed.

Format:

```
ACQUIRE,DN=dn,PDN=pdn,ID=uid,ED=ed,RT=rt,R=rd,W=wt,M=mn,

                    UQ,TEXT=text,MF=mf,TID=tid,DF=df.
```

Parameters are in keyword form; the only required parameter is DN.

DN=*dn* Local dataset name by which the permanent dataset is to be known. 1-7 alphanumeric characters, the first of which is A-Z, $, @, or %; remaining characters may also be numeric. DN is a required parameter.

PDN=*pdn* Name of COS permanent dataset to be accessed or staged from a front-end system, saved, and accessed. This is the name that is saved by the system if the dataset is staged. *pdn* is 1-15 alphanumeric characters assigned by the dataset creator. The default for *pdn* is *dn*.

ID=*uid* User identification. 1-8 alphanumeric characters assigned by the dataset creator. The default is no user ID.

ED=*ed* Edition number. A value from 1 through 4095 assigned by the dataset creator. The default value is:

- One, if a permanent dataset with the same PDN and ID does not currently exist, or

- The current highest edition number of that dataset if the permanent dataset with the specified PDN and ID does exist.

RT=*rt* Retention period. User-defined value from 0-4095 specifying the number of days that a permanent dataset is to be retained by the system. The default value is an installation-defined parameter.

R=*rd* Read control word. 1-8 alphanumeric characters assigned by the dataset creator. The default is no read control word.

W=*wt* Write control word. 1-8 alphanumeric characters assigned by the dataset creator. The default is no write control word.

M=*mn* Maintenance control word. 1-8 alphanumeric characters assigned by the dataset creator. The control word must be specified if a subsequent edition of the permanent dataset is saved. If no staging occurs, and the dataset is to be subsequently deleted, this parameter may be specified in conjunction with the UQ parameter (that is, maintenance permission is required to delete a dataset).

UQ Unique access. If specified, the job is granted unique access to the permanent dataset; otherwise, multiaccess to the permanent dataset is granted. If no staging is performed because the dataset already exists, write, maintenance, and/or read permission may be granted if the appropriate write or maintenance control words are specified.

TEXT=*text*

Text to be passed to the front-end system requesting transfer of the dataset. The format for TEXT is defined by the front-end system for managing its own datasets or files. Typically, *text* is in the form of one or more control statements for the front-end system; these statements must contain their own terminator for the front end. Any COS record control words are extracted from the text string before it is passed to the front end. *text* cannot exceed 240 characters.

MF=*mf* Mainframe identifier for the front-end computer. Two alphanumeric characters. The default is the mainframe of job origin.

TID=*tid* Terminal identifier. 1-8 alphanumeric characters identifying destination terminal. The default is terminal of job origin.

DF=*df* Dataset format. This parameter defines whether a dataset is to be presented to the CRAY-1 in COS blocked format and whether the front-end system is to perform character conversion. The default is CB.

For example, a user may wish to acquire a dataset from magnetic tape in blocked binary as it appears at the front-end system. In this case, BB is specified.

df is a 2-character alpha code defined for use on the front-end system. The default is CB. Cray Research suggests support of the following codes:

CD Character deblocked. The front-end system performs character conversion to 8-bit ASCII, if necessary.

CB Character blocked. The front-end system blocks the dataset prior to staging and performs character conversion to 8-bit ASCII, if necessary.

BD Binary deblocked. The front-end system does not perform character conversion. For ACQUIRE, BD is the same as TR.

BB Binary blocked. The front-end system blocks the dataset prior to staging but does not do character conversion.

TR Transparent. No blocking/deblocking or character conversion is performed.

DISPOSE - DISPOSE DATASET

The DISPOSE control statement directs a dataset to the CRAY-1 output queue for staging to a specified front-end computer system (mainframe). DISPOSE can also be used to alter dataset disposition characteristics or to release a dataset.

Defining the DISPOSE characteristics can be done before the actual staging via the DEFER parameter. The DEFER parameter saves all selected dispose parameters for use when the dataset is released, which is when the actual staging is initiated. DISPOSE is a system verb.

Format:

```
DISPOSE,DN=dn,SDN=sdn,DC=dc,DF=df,MF=mf,SF=sf,ID=uid,TID=tid,
```

```
ED=ed,RT=rt,R=rd,W=wt,M=mn,TEXT=text,WAIT,NOWAIT,DEFER,NRLS.
```

Parameters are in keyword form; the only required parameter is DN.

DN=*dn* Local dataset name. Name by which the dataset is known at the CRAY-1. DN is a required parameter.

SDN=*sdn* Staged dataset name. 1-15 character name by which the dataset will be known at destination mainframe. The default for *sdn* is *dn*.

DC=*dc* Disposition code. Disposition to be made of the dataset. The default is PR when the DC parameter is omitted.

dc is a 2-character alpha code describing the destination of the dataset as follows:

IN Input (job) dataset. Dataset is queued as a job on the mainframe specified with the MF parameter.

ST Stage to mainframe. Dataset is made permanent at the mainframe designated by the MF parameter.

SC Scratch dataset. Dataset is released.

PR Print dataset. Dataset is printed on a printer available at the mainframe designated by the MF parameter.

PU Punch dataset. Dataset is punched on any card punch available at the mainframe designated by the MF parameter.

PT Plot dataset. Dataset is plotted on any available plotter at the mainframe designated by the MF parameter.

MT Write dataset on magnetic tape at the mainframe designated by the MF parameter.

DF=*df* Dataset format. This parameter defines whether a dataset is sent from the CRAY-1 in COS blocked format and whether the front-end system is to perform character conversion. The default is CB.

For example, a user may wish to save a dataset on magnetic tape in blocked binary as it appears at the CRAY-1. In this case, BB is specified. A user who wants a dataset printed will specify CB if the front-end computer handles deblocking.

df is a 2-character alpha code defined for use on the front-end system. Cray Research suggests support of the following codes:

CD Character deblocked. The front-end system performs character conversion from 8-bit ASCII, if necessary.

CB Character blocked. No deblocking is performed at the CRAY-1 prior to staging. The front-end system performs character conversion from 8-bit ASCII, if necessary.

BD Binary deblocked. The front-end system does not perform character conversion.

BB Binary blocked. The front-end system does not perform character conversion. The CRAY-1 does not perform deblocking prior to staging. For DISPOSE, BB is the same as TR.

TR Transparent. No blocking/deblocking or character conversion is performed.

Other codes may be added by the local site. Undefined pairs of characters may be passed but will be treated as transparent mode by the CRAY-1.

MF=*mf* Mainframe computer identifier. Two alphanumeric characters. Identifies the front-end station where the dataset is to be staged. If omitted, the mainframe where the issuing job originated is used. If MF is given a value of a CRAY-1 ID and DC=IN, the dataset is disposed to the CRAY-1 input queue after issuing a warning message (see note).

NOTE

In future versions of COS, the SUBMIT control statement will be the only way to place datasets into the CRAY-1 job input queue. Therefore, it is advisable to use SUBMIT instead of DISPOSE to dispose datasets to the CRAY-1 input queue. If DISPOSE is used to submit a job to the CRAY-1 input queue, the following informative message is printed in the logfile: SY004 - USE SUBMIT TO PLACE JOBS IN CRAY INPUT QUEUE.

SF=*sf* Special form information to be passed to the front-end system. 1-8 alphanumeric characters. SF is defined by the needs of the front-end system.

ID=*uid* User identification. 1-8 alphanumeric characters assigned by the dataset creator. The default is no user ID.

TID=*tid* Terminal identifier. 1-8 alphanumeric characters identifying destination terminal. The default is terminal of job origin, where applicable.

ED=*ed* Edition number, meaningful only if DC=ST. A user-defined value from 1 through 4095. The default value depends on the destination mainframe.

RT=*rt* Retention period, meaningful only if DC=ST. A user-defined value from 0 through 4095 specifying the number of days a dataset is to be retained by the destination mainframe. The default value depends on the destination mainframe.

R=*rd* Read control word, meaningful only if DC=ST. 1-8 alphanumeric characters. The default is no read control word.

W=*wt* Write control word, meaningful only if DC=ST. 1-8 alphanumeric characters. The default is no write control word.

M=*mn* Maintenance control word, meaningful only if DC=ST. 1-8 alphanumeric characters. The default is no maintenance control word.

TEXT=*text*

Text to be passed to the front-end system requesting transfer of a dataset. The format for TEXT is defined by the front-end system for managing its own datasets or files. Typically, *text* is in the form of one or more control statements for the front-end system; these statements must contain their own terminator for the front end. Any COS record control words are extracted from the text string before it is passed to the front end. *text* cannot exceed 240 characters.

WAIT

Job wait. When this parameter is specified, the job does not resume processing until the disposed dataset has been staged to the front-end system. If the front-end system cancels the transfer, the waiting job is aborted. Processing then resumes after the next EXIT statement, if one is present. If WAIT is not specified, processing resumes immediately upon issue of the DISPOSE, depending upon an installation option. The WAIT parameter is useful in detecting unsuccessful transfers.

NOWAIT

Job no wait. When this parameter is specified, the job will not wait until the dataset has been staged to the front-end system but resumes processing immediately. If the front-end system cancels the transfer, no special action is taken, that is, the job is not aborted. If NOWAIT is not specified, processing resumes immediately upon issue of the DISPOSE, depending upon an installation option.

DEFER

When this parameter is specified, the disposition occurs when the dataset is released either by a RELEASE request or job termination. The dispose characteristics are saved and used when the dataset is released.

NRLS

No release. When this parameter is specified, the dataset remains local to the job after the DISPOSE request has been processed. When NRLS is specified on a DISPOSE control statement, the dataset cannot be written to, until the transfer to the specified front-end is completed. Therefore, it is advisable to use WAIT with NRLS.

SUBMIT - SUBMIT JOB DATASET

With SUBMIT, a job running on the CRAY-1 can direct another dataset (which must also be a job) to the CRAY-1 input queue. The job that is submitted executes independently of the submitting job. SUBMIT is a system verb.

Format:

```
SUBMIT,DN=dn,SID=mf,DID=mf,TID=tid,DEFER,NRLS.
```

Parameters are in keyword format; the only required parameter is DN.

DN=*dn* Local dataset name. A valid local dataset name. DN is a required parameter and must be given a value.

SID=*mf* Default source front-end system identifier. Two alphanumeric characters. If an MF parameter is not specified in an ACQUIRE control statement for the submitted job, the SID parameter defines the default source front-end system for the dataset to be acquired. If the MF parameter as well as the SID are omitted, the default source identifier of the submitting job is used.

DID=*mf* Default destination mainframe identifier. Two alphanumeric characters. If an MF parameter is not specified in a DISPOSE control statement for the submitted job, the DID parameter defines the default destination front-end system for the dataset to be acquired. If the MF parameter as well as the DID are omitted, the default destination identifier of the submitting job is used.

TID=*tid* Default terminal identifier. 1-8 alphanumeric character identifier which defines the default terminal ID for the submitted job. If omitted, then the terminal ID of the submitting job is used.

DEFER Deferred submit. Selection of this parameter causes the SUBMIT characteristics to be defined, with a release of the dataset actually initiating the submit of the dataset. If omitted, the SUBMIT occurs immediately.

NRLS No release. This parameter indicates if the dataset is to remain local to the job after SUBMIT has been processed. If omitted, the dataset is released after the SUBMIT. If selected, the dataset remains local to the job after the SUBMIT. If the dataset is not released, it is available for reading only. When NRLS is specified on a SUBMIT control statement, the dataset cannot be written to, until the transfer to the specified front-end is completed.

DATASET UTILITIES 6

Utility control statements provide the user with a convenient means of copying, positioning, or dumping datasets. The following utilities are available to the user:

- COPYR, COPYF, and COPYD allow the user to copy records, files, or datasets, respectively.

- SKIPR, SKIPF, and SKIPD allow the user to skip records, files, or datasets, respectively.

- REWIND positions a dataset at the beginning of data, that is, prior to the first block control word of the dataset.

- WRITEDS is intended for initializing a random dataset but may also initialize a sequential dataset.

All parameters are in keyword form and have default values.

COPYR - COPY RECORDS

The COPYR statement copies a specified number of records from one dataset to another starting at the current dataset position. Following the copy, the datasets are positioned after the end-of-record for the last record copied.

Format:

```
COPYR,I=idn,O=odn,NR=n.
```

Parameters are in keyword form.

I=*idn* Name of dataset to be copied. The default is $IN.

O=*odn* Name of dataset to receive the copy. The default is $OUT.

NR=*n* Decimal number of records to copy. The default is 1. If the dataset contains fewer than *n* records, the copy prematurely terminates on the next end-of-file. End-of-file or end-of-data is not written. If the keyword NR is specified without a value, the copy terminates at the next end-of-file. If the input dataset is positioned midrecord, the partial record is counted as one record.

COPYF - COPY FILES

The COPYF statement copies a specified number of files from one dataset to another starting at the current dataset position. Following the copy, the datasets are positioned after the end-of-file for the last file copied.

Format:

```
COPYF,I=idn,O=odn,NF=n.
```

Parameters are in keyword form.

I=*idn* Name of dataset to be copied. The default is $IN.

O=*odn* Name of dataset to receive the copy. The default is $OUT.

NF=*n* Decimal number of files to copy. The default is 1. If the dataset contains fewer than *n* files, the copy prematurely terminates on end-of-data. End-of-data is not written. If the keyword NF is specified without a value, the copy terminates at the end-of-data. If the input dataset is positioned midfile, the partial file counts as one file.

COPYD - COPY DATASET

The COPYD statement copies one dataset to another starting at their current positions. Following the copy, both datasets are positioned after the end-of-file of the last file copied. The end-of-data is not written to the output dataset. Both input and output datasets must be blocked.

Format:

```
COPYD,I=idn,O=odn.
```

Parameters are in keyword form.

I=*idn* Name of dataset to be copied. The default is $IN.

O=*odn* Name of dataset to receive the copy. The default is $OUT.

SKIPR - SKIP RECORDS

The SKIPR control statement directs the system to bypass a specified number of records from the current position of the named dataset.

Format:

```
SKIPR,DN=dn,NR=n.
```

Parameters are in keyword form.

DN=*dn* Name of dataset to be bypassed. The default is $IN.

NR=*n* Decimal number of records to skip. The default is 1. If the keyword NR is specified without a value, the system positions *dn* after the last end-of-record of the current file. If *n* is negative, SKIPR skips backward on *dn*.

SKIPR does not bypass an end-of-file or beginning-of-data. If an end-of-file or beginning-of-data is encountered before *n* records have been bypassed when skipping backward, the dataset is positioned after the end-of-file or beginning-of-data; when skipping forward, the dataset is positioned after the last end-of-record of the current file. This statement is available for use with online tapes except that a negative value cannot be used for NR.

SKIPF - SKIP FILES

The SKIPF control statement directs the system to bypass a specified number of files from the current position of the named dataset.

Format:

```
SKIPF,DN=dn,NF=n.
```

Parameters are in keyword form.

DN=*dn* Name of dataset to be bypassed. The default is $IN.

NF=*n* Decimal number of files to bypass. The default is 1. If the keyword NF is specified without a value, the system positions *dn* after the last end-of-file of the dataset. If *n* is negative, SKIPF skips backward on *dn*.

If *dn* is positioned midfile, the partial file skipped counts as one file.

SKIPF does not bypass an end-of-data or beginning-of-data. If beginning-of-data is encountered before *n* files have been bypassed when skipping backward, the dataset is positioned after the beginning-of-data; when skipping forward, the dataset is positioned before the end-of-data of the current file. This statement is available for use with online tapes except that a negative value cannot be used for NF; for interchange format tapes (DF=IC), NF can only be 1.

For example, if *dn* is positioned just after an end-of-file, the following control statement will position *dn* after the previous end-of-file. If *dn* is positioned midfile, *dn* will be positioned at the beginning of that file.

```
SKIPF,DN=dn,NF=-1.
```

SKIPD - SKIP DATASET

The SKIPD control statement directs the system to position a dataset at end-of-data, that is, after the last end-of-file of the dataset. It has the same effect as the following statement:

```
SKIPF,DN=dn,NF.
```

If the specified dataset is empty or already at end-of-data, the statement has no effect.

Format:

```
SKIPD,DN=dn.
```

The parameter is in keyword form.

DN=*dn* Name of dataset to be skipped. The default is $IN.

REWIND - REWIND DATASET

The REWIND control statement positions the named datasets at the beginning-of-data, that is, prior to the first block control word of the dataset. The $IN dataset represents an exception. After REWIND, $IN is positioned after the control statement file. If any of the named datasets is not open, REWIND opens it. REWIND is a system verb.

REWIND causes an end-of-data to be written to the dataset if the previous operation was a write or if the dataset is null. If the dataset is not memory resident, the buffers are flushed to mass storage when REWIND follows a write operation. If the dataset is memory resident, the end-of-data is still placed in the buffer, but the buffer is not flushed. For an online magnetic tape dataset, REWIND positions the tape dataset to the beginning of the first volume accessed by the user.

Format:

> REWIND,DN=dn_1:dn_2:...:dn_8.

Parameters are in keyword form.

DN=dn_i Names of datasets to be rewound. A maximum of eight datasets can be specified, separated by colons.

WRITEDS - WRITE RANDOM OR SEQUENTIAL DATASET

The WRITEDS control statement is intended for initializing a blocked dataset. It writes a dataset containing a single file consisting of a specified number of records of a specified length. This utility is especially useful for random datasets because a record written on a random dataset must end on a pre-existing record boundary. Direct-access datasets, implemented in CFT as defined by the ANSI x3.9-1978 FORTRAN standard, can be initialized (and even extended) without the help of WRITEDS.

WRITEDS can also be used to write a sequential dataset.

Format:

> WRITEDS,DN=*dn*,NR=*nr*,RL=*rl*.

Parameters are in keyword form; the only required parameters are DN and NR.

DN=*dn* Name of dataset to be written. DN is a required parameter.

NR=*nr* Decimal number of records to be written. NR is a required parameter. Set to the largest value that may be needed, since a dataset cannot be extended when it is in random (RDM) mode.

RL=*rl* Decimal record length, that is, the number of words in each record. The default is zero words, which generates a null record.

If the record length is 1 or greater, the first word of each record is the record number as a binary integer starting with 1.

PERMANENT DATASET UTILITIES 7

The following utility routines are provided for permanent datasets:

- PDSDUMP dumps all specified permanent datasets to a user-specified dataset. Input and output datasets may be included in the dump.
- PDSLOAD loads permanent datasets that have been dumped by PDSDUMP and updates or regenerates the Dataset Catalog. Input and output datasets are also loaded via PDSLOAD.
- AUDIT produces a report containing status information for each permanent dataset. AUDIT does not include input or output datasets.

PDSDUMP - DUMP PERMANENT DATASET

PDSDUMP dumps specified permanent datasets to a dataset, which may then be saved or staged to a station as desired. Conditions that cause a dataset to be omitted from dumping include:

- The dataset is execute-only,
- There are dataset allocation conflicts,
- The dataset has catastrophic errors,
- Inconsistent allocation has occurred,
- The dataset resides on a down device, or
- The dataset has an inactive entry in the system's Queued Dataset Table (QDT).

Format:

PDSDUMP,DN=*dn*,DV=*ldv*,PDS=*pds*,CW=*cw*,

ID=*uid*,US=*usn*,ED=*ed*,X,C,D,I,O,S.

All parameters are in keyword form. Optional parameters establish criteria for datasets being dumped.

DN=*dn* Name of dataset where dump is written. The default is $PDS. Multiple dumps to a dataset are possible; if the dataset specified already exists, the dump is appended to it.

DV=*ldv* Dump all datasets residing on logical device *ldv*. Currently only one *ldv* can be specified.[§]

PDS=*pds* Dump all editions of the specified permanent dataset. Editions may be limited by ED parameter.[§]

CW=*cw* Installation-defined control word regulating use of PDSDUMP. If the user number is specified on the JOB control statement, the CW parameter is not usually required; only the datasets with that user number are selected. If the CW parameter is omitted, only the datasets belonging to the user number as specified on the JOB control statement can be dumped. If the CW parameter is present and the correct control word is used, any dataset can be dumped. If an invalid control word is given, the job is aborted. When the user number is omitted from the JOB control statement, CW is a required parameter.

ID=*uid* Dump all datasets with user identification as specified.[§] If ID is specified without a value, all datasets which meet the rest of the criteria and have a null id are dumped.

US=*usn* Dump all datasets with specified user number.[§]

ED=*ed* Edition number of permanent dataset dumped; meaningful only if PDS parameter is specified.[§]

X Dump expired datasets.

C Dump selected datasets never dumped or datasets modified since the last dump of the dataset.

D Delete datasets that are dumped.

I Dump system input datasets.
O Dump system output datasets.
S Dump user permanent datasets.

} See note

[§] By default, all permanent datasets specified by the parameters are dumped.

NOTE

If none of these parameters is specified, the input, output, and user permanent datasets are all dumped. If any of these parameters is specified, only those datasets of the type specified are dumped.

Multiple calls to PDSDUMP may be made if the dump dataset is to include several permanent datasets requiring specification of different parameters.

Example:

```
    PDSDUMP,DN=DUMPA,PDS=LIB1.
    PDSDUMP,DN=DUMPA,PDS=LIB2.
```

This example results in a dataset DUMPA that contains all editions of LIB1 and all editions of LIB2.

PDSDUMP produces a listing (figure 7-1) on $OUT identifying the datasets dumped or bypassed and summarizing the dump run. The date and time in the heading line refer to the time when the dump run started. The permanent dataset name, edition number, ID, and user number are extracted from the DSC entry for each dataset selected. Each message is followed by the notation DUMPED or NOT DUMPED. The notation NOT DUMPED indicates the dataset was selected but could not be accessed for dumping. A user logfile message further explains the problem encountered.

When dumping to a tape dataset, the recording format for the tape dataset must be transparent (for example, DF=TR on ACCESS statement). If the dataset is recorded in interchange format loading, using the dump dataset leads to unsuccessful results.

```
 PDSDUMP - PERMANENT DATASET DUMP UTILITY   DUMP ON 01/07/82 AT   14:50:44
AUDPL           ED=0001 ID=QITTYQAT USR=SYSTEM          DUMPED
AUDPL           ED=0002 ID=QITTYQAT USR=SYSTEM          DUMPED
DSCED           ED=0001 ID=QITTYQAT USR=SYSTEM          DUMPED
DSCED           ED=0002 ID=QITTYQAT USR=SYSTEM          DUMPED
TXBUILD         ED=0001 ID=QITTYQAT USR=SYSTEM          DUMPED
TXBUILD         ED=0002 ID=QITTYQAT USR=SYSTEM          DUMPED
TXBUILD         ED=0003 ID=QITTYQAT USR=SYSTEM          DUMPED
LONGDATASETNAME ED=0001 ID=QITTYQAT USR=SYSTEM          DUMPED
LONGDATASETNAME ED=0002 ID=QITTYQAT USR=SYSTEM          DUMPED
LONGDATASETNAME ED=0003 ID=QITTYQAT USR=SYSTEM          DUMPED
LONGDATASETNAME ED=0004 ID=QITTYQAT USR=SYSTEM          DUMPED
DSBUILD         ED=0001 ID=QITTYQAT USR=SYSTEM          DUMPED
DSBUILD         ED=0002 ID=QITTYQAT USR=SYSTEM          DUMPED
DSBUILD         ED=0003 ID=QITTYQAT USR=SYSTEM          DUMPED
DSBUILD         ED=0004 ID=QITTYQAT USR=SYSTEM          DUMPED
AUDPL           ED=0003 ID=QITTYQAT USR=SYSTEM          DUMPED
DSCED           ED=0003 ID=QITTYQAT USR=SYSTEM          DUMPED
TXBUILD         ED=0004 ID=QITTYQAT USR=SYSTEM          DUMPED
AUDPL           ED=0004 ID=QITTYQAT USR=SYSTEM          DUMPED
DSCED           ED=0004 ID=QITTYQAT USR=SYSTEM          DUMPED
             20 DATASETS SELECTED FOR DUMPING
```

Figure 7-1. PDSDUMP listing

PDSLOAD - LOAD PERMANENT DATASET

PDSLOAD loads permanent datasets from a dataset created by PDSDUMP. If the dataset already exists, it is not reloaded.

Format:

PDSLOAD,DN=*dn*,PDS=*pds*,CW=*cw*,ID=*uid*,US=*usn*,ED=*ed*,DV=*dvn*,A,I,O,S,NA.

All parameters are in keyword form. Optional parameters establish criteria for datasets being loaded.

DN=*dn* Name of dataset from which permanent dataset is to be loaded. The default is $PDS.

PDS=*pds* Load all editions of the specified permanent dataset. Editions may be limited by the ED parameter.[§]

[§] By default, all permanent datasets that are specified by the parameters are loaded.

CW=*cw* Installation-defined control word to regulate the use of PDSLOAD. The CW parameter is usually not required. If the CW parameter is used when the user number is specified, the datasets with the user number are searched. If the CW parameter is omitted when the user number is specified, only the datasets belonging to that user number may be loaded.

When the user number is omitted from the JOB control statement, CW is a required parameter. When the CW parameter is specified on the PDSLOAD control statement, the user can load any datasets with the correct control word. If an invalid control word is given, the job is aborted.

ID=*uid* Load all datasets with user identification as specified.

US=*usn* Load all datasets with specified user number.[§]

ED=*ed* Edition number of dataset to be loaded; meaningful only if PDS parameter is specified.[§]

DV=*dvn* The name of a logical device where the output dataset is assigned before it is opened. If omitted, COS assigns a device at open time. If specified, the supplied device name is built into the DNT entry for the output dataset (the one being loaded). Note that COS can choose not to honor this assignment. This parameter is not involved in any way in the selection of a dataset for loading.

A Load only active datasets; that is, do not load expired datasets.

I Load input datasets.

O Load output datasets. See note following.

S Load saved datasets.

NA Do not abort if there is not a dataset matching the specifications to load on the $PDS dataset. This parameter applies only to this situation. It does not prevent any other abort condition from occurring or offer reprieve processing of any kind.

[§] By default, all permanent datasets that are specified by the parameters are loaded.

NOTE

If none of these parameters is specified, the input, output, and saved datasets are loaded. If any of these parameters is specified, only those datasets of the type specified are loaded.

PDSLOAD produces a listing on $OUT identifying the datasets loaded or bypassed and summarizing the load run. The date and time in the heading line refer to the time when the load run started. The permanent dataset name, edition number, ID, and user number are extracted from the PDD for each dataset selected and successfully loaded. Each message is followed by the notation LOADED or NOT LOADED. The notation NOT LOADED indicates the dataset was selected but not loaded. A user logfile message further explains the problem encountered.

```
 PDSLOAD - PERMANENT DATASET RESTORE UTILITY  LOAD ON 01/07/82 AT 17:13:47
ENTIT           ED=0001 ID=TAQI     USR=SYSTEM          LOADED
DSBUILD         ED=0001 ID=TAQI     USR=SYSTEM          LOADED
TXBUILD         ED=0001 ID=TAQI     USR=SYSTEM          LOADED
AUDPL           ED=0001 ID=TAQI     USR=SYSTEM          LOADED
DSCED           ED=0001 ID=TAQI     USR=SYSTEM          LOADED
               5 DATASETS SELECTED FOR LOADING
```

AUDIT - AUDIT PERMANENT DATASETS

The AUDIT utility provides reports on the status of each permanent dataset known to the system. If the user number for the job is SYSTEM, AUDIT reports on all permanent datasets. Otherwise, AUDIT only reports on those permanent datasets whose user number matches the user number for the job. AUDIT does not include input and output datasets.

If more than one parameter is selected, only those datasets which meet all criteria are listed. Parameter values can be selected that conflict with each other, such as PDN and PREFIX. For example, requesting that permanent dataset names that begin with ABC (PDN=ABC-) and whose prefix characters are BOT (PREFIX=BOT) would result in no permanent datasets being listed.

AUDIT supplies the following information on the listing:

Permanent dataset name
Edition number
User identifications
Dataset size in words
Retention time in decimal
Number of accesses in decimal
Total block count in decimal
Creation date/time
Last dump date/time
Last access date/time
Last modification date/time
Logical device name
Number of datasets selected

Format:

```
AUDIT,L=ldn,B=bdn,PDN=pdn,ID=uid,PREFIX=pfx,DV=dvn,
   SZ=dsz,X=mm/dd/yy:'hh:mm:ss',TCR=mm/dd/yy:'hh:mm:ss',
      TLA=mm/dd/yy:'hh:mm:ss',TLM=mm/dd/yy:'hh:mm:ss'.
```

Parameters are in keyword form.

L=*ldn* List dataset name. The default is $OUT.

B=*bdn* Specifies dataset to receive the binary output. If B is specified alone, the dataset is $BINAUD. If the B parameter is omitted, no binary output is written. For a description of the binary output format, refer to the COS Product Set Internal Reference Manual, CRI publication SM-0041.

PDN=*pdn* Name of permanent dataset or datasets to be listed. Up to 15 alphanumeric characters may be specified. A shorthand notation may be used where a dash represents any number of characters or no characters and an asterisk represents any one character.

Examples:

PDN=ABC- List all permanent dataset names beginning with ABC.

PDN=A*** List all 4-character permanent dataset names beginning with A.

PDN=-A*- List all permanent dataset names containing the letter A followed by one or more other characters.

PDN=- List all names.

PDN=***- List all names having three or more characters.

ID=*uid* List all permanent datasets with the specified user identification. The default is to list all IDs. If ID is present without an equated value, datasets having a null id are selected.

PREFIX=*pfx*

List all permanent datasets whose names begin with the specified prefix. *pfx* is 1-8 characters. The default is no prefix specified.

DV=*dvn* List all permanent datasets on the specified logical device. The default is to list permanent datasts on all devices.

SZ=*dsz* List all permanent datasets greater than or equal to the specified size. Size is specified in words. The default is to list all sizes.

X=*mm/dd/yy*:'*hh*:*mm*:*ss*'

List all permanent datasets that are expired as of the specified *mm/dd/yy*:'*hh*:*mm*:*ss*'. *mm/dd/yy* may be specified alone. The default expiration date and time are "now" if only X is specified.

TCR=*mm/dd/yy*:'*hh*:*mm*:*ss*'

List all permanent datasets that have been created since the specified *mm/dd/yy*:'*hh*:*mm*:*ss*'. The keyword cannot be specified alone; however, TCR=*mm/dd/yy* is sufficient.

TLA=*mm/dd/yy*:'*hh*:*mm*:*ss*'

List all permanent datasets that have not been accessed since the specified *mm/dd/yy*:'*hh*:*mm*:*ss*'. The keyword cannot be specified alone; however, TLA=*mm/dd/yy* is sufficient.

TLM=*mm/dd/yy*:'*hh*:*mm*:*ss*'

List all permanent datasets that have been modified since the specified *mm/dd/yy*:'*hh*:*mm*:*ss*'. The keyword cannot be specified alone; however, TLM=*mm/dd/yy* is sufficient.

ANALYTICAL AIDS 8

The following control statements provide analytical aids to the programmer:

- DUMPJOB and DUMP are generally used together to examine the contents of registers and memory as they were at a specific time during job processing. DUMPJOB captures the information so that DUMP can later format selected parts of it.
- DEBUG produces a symbolic dump.
- DSDUMP dumps all or part of a dataset to another dataset in one of two formats: blocked or unblocked.
- COMPARE compares two nearly identical datasets and lists all differences.
- FLODUMP dumps flowtrace tables when a program aborts with flowtrace active.
- PRINT writes the value of an expression to the logfile.
- SYSREF generates a global cross-reference listing for a group of CAL or APML programs.

DUMPJOB - CREATE $DUMP

The DUMPJOB control statement causes creation of the local dataset $DUMP, if not already existent. $DUMP receives an image of the memory assigned to the job (JTA and user field) when the DUMPJOB statement is encountered. If DUMPJOB is placed after a system verb (excluding the comment and EXIT statements), the dump is of the Control Statement Processor (CSP). A DUMPJOB statement is not honored if an execute-only dataset is loaded in memory; a DUMPJOB to an execute-only dataset is rejected.

If $DUMP already exists, it is overwritten each time a DUMPJOB control statement is processed. If $DUMP is permanent and the job does not have write permission, DUMPJOB aborts. If $DUMP is permanent and the job has write permission, the dataset is overwritten.

If the DUMPJOB/DUMP sequence fails because of such situations as destroyed system-managed DSPs, assign $DUMP and save it with unique access. DUMPJOB writes to $DUMP, and job termination automatically adjusts $DUMP. $DUMP can then be inspected in a separate job.

$DUMP is created as an unblocked dataset by DUMPJOB for use by DUMP. DUMPJOB is a system verb and cannot be continued to subsequent cards.

Format:

```
DUMPJOB.
```

Parameters: none

DUMP - DUMP REGISTERS AND MEMORY

DUMP reads and formats selected parts of the memory image contained in $DUMP and writes the information onto another dataset. The DUMP statement can be placed anywhere in the control statement file after $DUMP has been created by the DUMPJOB control statement.

Placing the DUMPJOB and DUMP statements after an EXIT statement is conventional and provides the advantage of giving the dump regardless of which part of the job causes an error exit. The usage of DUMP and DUMPJOB, however, is not restricted to this purpose.

DUMP can be called any number of times within a job. This might be done to dump selected portions of memory from a single $DUMP dataset or it might be done if $DUMP has been created more than once in a single job.

Format:

DUMP,I=*idn*,O=*odn*,FW=*fwa*,LW=*lwa*,JTA,NXP,V,DSP,FORMAT=*f*,CENTER.

Parameters are in keyword form.

I=*idn* Name of the dataset containing the memory image. The dataset $DUMP is created by DUMPJOB and is the default, but any dataset in the $DUMP (unblocked) format is acceptable.

O=*odn* Name of the dataset to receive the dump; default is $OUT.

FW=*fwa* Octal first word address of memory to dump. The default is 0.

LW=*lwa* Octal last word address+1 of memory to dump. The default is 200_8. Specifying the keyword LW without a value causes the limit address to be used.

JTA Job Table Area to be dumped. The default is no dump.

NXP No exchange package, B registers, or T registers dumped. The default causes exchange package, B registers, and T registers to be dumped.

V Vector registers to be dumped. The default is no dump of V registers.

DSP Logical File Tables (LFTs) and Dataset Parameter Areas (DSPs) to be dumped. The default is to not dump LFTs and DSPs.

FORMAT=*f* Format for the part of memory selected by FW and LW. The options are:

O Octal integer and ASCII character. This is the default.

D Decimal integer and ASCII character

X Hexadecimal integer and ASCII character

G Floating point or exponential (depending on the value of the number) and ASCII character

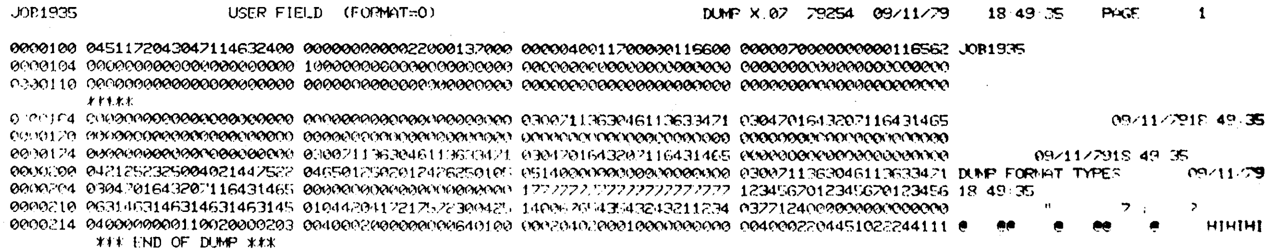
P 16-bit parcel (4-word boundaries are forced for FW and LW)

M Mixed hexadecimal and octal written in ASCII. Each 16-bit parcel is represented as five characters; the first is a hexadecimal digit representing the upper 4 bits and the next four are octal characters representing the lower 12 bits.

CENTER Dump 100_8 words on each side of the address contained in the P register of the exchange package. The format is P.

Examples:

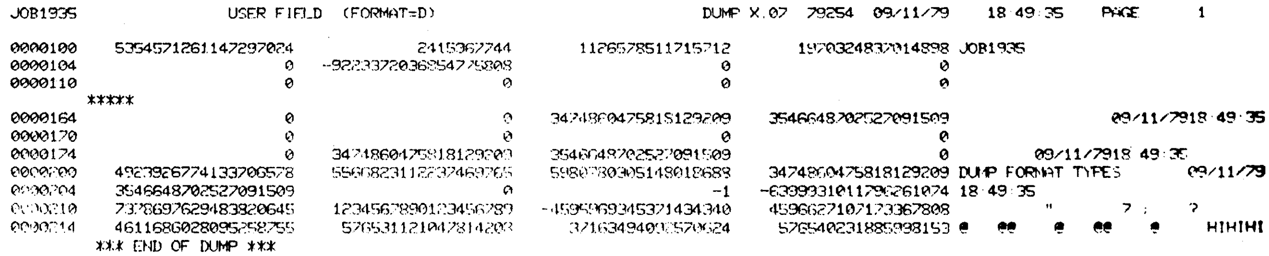
The following example is a portion of the dump obtained using format O, the default format type:



```
JOB1935                  USER FIELD  (FORMAT=O)                                   DUMP X.07  79254  09/11/79    18 49 35    PAGE      1

0000100 0451172043047114632400 0000000000022000137000 0000040011700000115600 0000070000000000116562 JOB1935
0000104 0000000000000000000000 1000000000000000000000 0000000000000000000000 0000000000000000000000
0000110 0000000000000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000
        *****
0000164 0000000000000000000000 0000000000000000000000 0300711363046113633471 0304701643207116431465                 09/11/7918 49 35
0000170 0000000000000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000
0000174 0000000000000000000000 0300711363046113633471 0304701643207116431465 0000000000000000000000         09/11/7918 49 35
0000200 0421252325004021447522 0465012502012426250105 0514000000000000000000 0300711363046113633471 DUMP FORMAT TYPES        09/11/79
0000204 0304701643207116431465 0000000000000000000000 1777777777777777777777 1234567012345670123456 18 49 35
0000210 0631463146314631463145 0104420417217522300425 1400626543543243211234 0377124000000000000000          "        ? ;     ?
0000214 0400000000110020000203 0040002000000000640100 0002040200010000000000 0040002204451022244111 @    @@    @    @@    @      HIHIHI
          *** END OF DUMP ***
```

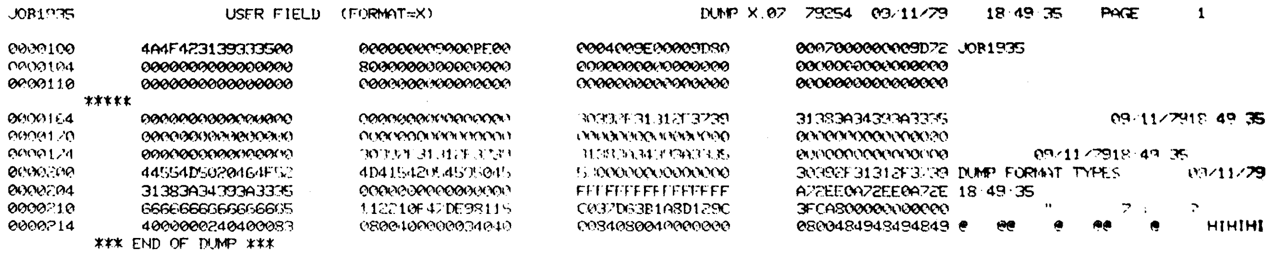
A portion of the dump in format D:



```
JOB1935                  USER FIELD  (FORMAT=D)                                   DUMP X.07  79254  09/11/79    18 49 35    PAGE      1

0000100    5354571261147297024            2415367744       1126578511715712       1970324837014898 JOB1935
0000104                      0  -9223372036854775808                      0                      0
0000110                      0                     0                      0                      0
        *****
0000164                      0                     0    3474860475818129209    3546648702527091509                 09/11/7918 49 35
0000170                      0                     0                      0                      0
0000174                      0   3474860475818129209    3546648702527091509                      0         09/11/7918 49 35
0000200    4923926774133706578   5560823112237469765    5980780305148018688    3474860475818129209 DUMP FORMAT TYPES        09/11/79
0000204    3546648702527091509                     0                     -1   -6399331011796261074 18 49 35
0000210    7378697629483820645   1234567890123456789   -4595969345371434340    4596627107173367808          "        ? ;     ?
0000214    4611686028095258755    576531121047814203      37163494095570624     576540231885998153 @    @@    @    @@    @      HIHIHI
          *** END OF DUMP ***
```

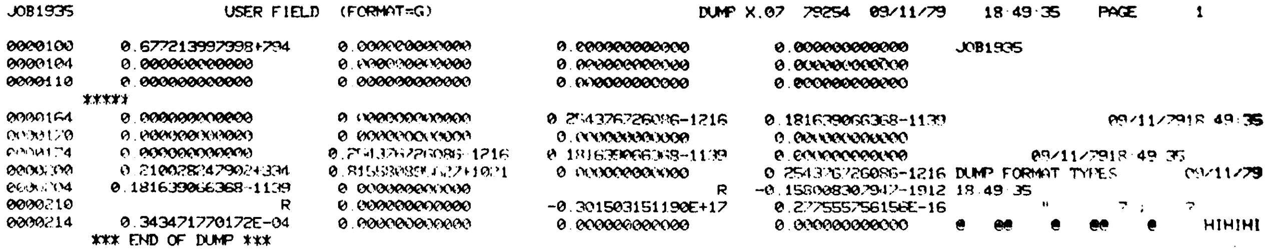
A portion of the same dump specifying format X:



```
JOB1935                  USER FIELD  (FORMAT=X)                                   DUMP X.07  79254  09/11/79    18 49 35    PAGE      1

0000100       4A4F423139333500       000000009000BE00       0004009E00009D80       0007000000009D72 JOB1935
0000104       0000000000000000       8000000000000000       0000000000000000       0000000000000000
0000110       0000000000000000       0000000000000000       0000000000000000       0000000000000000
        *****
0000164       0000000000000000       0000000000000000       30392F31312F3739       31383A34393A3335                 09/11/7918 49 35
0000170       0000000000000000       0000000000000000       0000000000000000       0000000000000000
0000174       0000000000000000       30392F31312F3739       31383A34393A3335       0000000000000000         09/11/7918 49 35
0000200       44554D50204F4F52       4D41542054595045       5300000000000000       30392F31312F3739 DUMP FORMAT TYPES        09/11/79
0000204       31383A34393A3335       0000000000000000       FFFFFFFFFFFFFFFF       A72EE0A72EE0A72E 18 49 35
0000210       6666666666666665       112210F47DE98115       C037D63B1A8D129C       3FCA800000000000          "        ? ;     ?
0000214       4000000240400083       0800400000034040       0084080040000000       0800484948494849 @    @@    @    @@    @      HIHIHI
          *** END OF DUMP ***
```

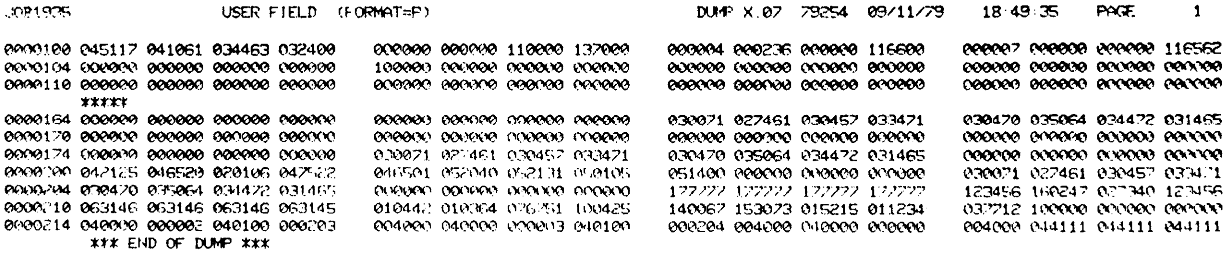
Format G specified on the same dump portion:



```
JOB1935              USER FIELD  (FORMAT=G)                              DUMP X.07  79254  09/11/79    18:49:35    PAGE      1

0000100    0.677213997398+794    0.000000000000        0.000000000000        0.000000000000      JOB1935
0000104    0.000000000000        0.000000000000        0.000000000000        0.000000000000
0000110    0.000000000000        0.000000000000        0.000000000000        0.000000000000
         *****
0000164    0.000000000000        0.000000000000       0.254376726086-1216   0.181639066368-1139                      09/11/7918:49:35
0000170    0.000000000000        0.000000000000        0.000000000000        0.000000000000
0000174    0.000000000000       0.254376726086-1216   0.181639066368-1139    0.000000000000                 09/11/7918:49:35
0000200    0.210028247902+334   0.815530850627+1021    0.000000000000       0.254376726086-1216 DUMP FORMAT TYPES        09/11/79
0000204   0.181639066368-1139    0.000000000000                      R   -0.158008307942-1912 18:49:35
0000210                      R   0.000000000000       -0.301503151190E+17    0.227555756156E-16            "          7 ;      ?
0000214    0.343471770172E-04    0.000000000000        0.000000000000        0.000000000000      @    @@    @    @@    @      HIHIHI
         *** END OF DUMP ***
```

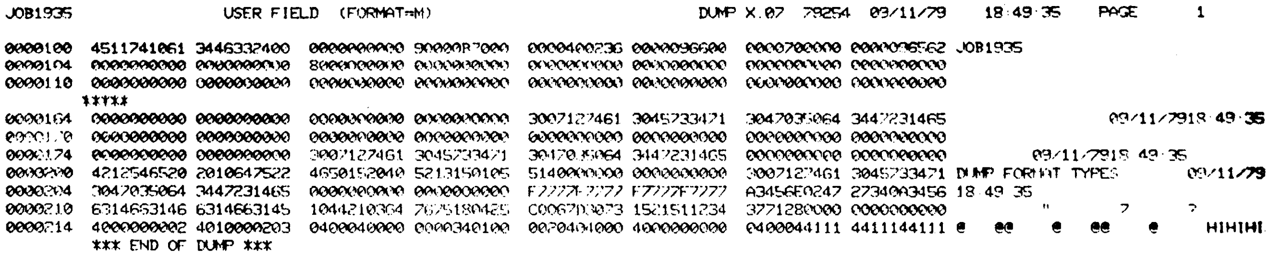
The same portion of the dump in format P:



```
JOB1935              USER FIELD  (FORMAT=P)                              DUMP X.07  79254  09/11/79    18:49:35    PAGE      1

0000100 045117 041061 034463 032400    000000 000000 110000 137000    000004 000236 000000 116600    000007 000000 000000 116562
0000104 000000 000000 000000 000000    100000 000000 000000 000000    000000 000000 000000 000000    000000 000000 000000 000000
0000110 000000 000000 000000 000000    000000 000000 000000 000000    000000 000000 000000 000000    000000 000000 000000 000000
        *****
0000164 000000 000000 000000 000000    000000 000000 000000 000000    030071 027461 030457 033471    030470 035064 034472 031465
0000170 000000 000000 000000 000000    000000 000000 000000 000000    000000 000000 000000 000000    000000 000000 000000 000000
0000174 000000 000000 000000 000000    030071 027461 030457 033471    030470 035064 034472 031465    000000 000000 000000 000000
0000200 042125 046520 020106 047522    046501 052040 052131 050105    051400 000000 000000 000000    030071 027461 030457 033471
0000204 030470 035064 034472 031465    000000 000000 000000 000000    177777 177777 177777 177777    123456 160247 027340 123456
0000210 063146 063146 063146 063145    010442 010364 076751 100425    140067 153073 015215 011234    037712 100000 000000 000000
0000214 040000 000002 040100 000203    004000 040000 000003 040100    000204 004000 040000 000000    004000 044111 044111 044111
         *** END OF DUMP ***
```

The same portion of the dump in format M:



```
JOB1935              USER FIELD  (FORMAT=M)                              DUMP X.07  79254  09/11/79    18:49:35    PAGE      1

0000100  4511741061 3446332400  0000000000 90000B7000  0000400236 0000096600  0000700000 0000096562 JOB1935
0000104  0000000000 0000000000  8000000000 0000000000  0000000000 0000000000  0000000000 0000000000
0000110  0000000000 0000000000  0000000000 0000000000  0000000000 0000000000  0000000000 0000000000
        *****
0000164  0000000000 0000000000  0000000000 0000000000  3007127461 3045733471  3047035064 3447231465                  09/11/7918:49:35
0000170  0000000000 0000000000  0000000000 0000000000  6000000000 0000000000  0000000000 0000000000
0000174  0000000000 0000000000  3007127461 3045733471  3047035064 3447231465  0000000000 0000000000            09/11/7918:49:35
0000200  4212546520 2010647522  4650152040 5213150105  5140000000 0000000000  3007127461 3045733471 DUMP FORMAT TYPES        09/11/79
0000204  3047035064 3447231465  0000000000 0000000000  F7777F7777 F7777F7777  A3456E0247 2734003456 18:49:35
0000210  6314663146 6314663145  1044210364 7675180425  C00670B073 1521511234  3771280000 0000000000            "          7       ?
0000214  4000000002 4010000203  0400040000 0000340100  0020404000 4000000000  0400044111 4411144111 @    @@    @    @@    @      HIHIHI
         *** END OF DUMP ***
```

DEBUG - PRODUCE SYMBOLIC DUMP

The symbolic debug utility routine, DEBUG, provides a means of dumping portions of memory and interprets the dump in terms of FORTRAN or CAL symbols. DEBUG is normally used when a job aborts after an EXIT, DUMPJOB sequence, however it may be used anywhere provided that a valid version of $DUMP exists.

To be useful, both CFT and CAL must write special tables, which the loader (LDR) augments with a version of the load map. The loader writes this information on a dataset called $DEBUG, which gives the FORTRAN or CAL symbol names associated with memory addresses. This is initiated by specifying the ON=Z option for CFT or the SYM option for CAL. DEBUG reads $DEBUG and $DUMP and prints out variable names and values in a format appropriate for the variable type.

The following example shows the conventional use of DEBUG:

```
JOB, ... .
CFT,ON=Z.
LDR.
EXIT.
DUMPJOB.
DEBUG.
   .
   .
   .
```

The library routine SYMDEBUG may be called from either FORTRAN or CAL with one argument, which is a Hollerith string containing any of the DEBUG parameters. SYMDEBUG produces output similar to that produced by DUMP but interprets the memory of the running program rather than $DUMP.

Format:

```
DEBUG,I=idn,O=odn,DUMP=ddn,TRACE=n,SYMS=sym,NOTSYMS=nysm,

    MAXDIM=dim,BLOCKS=blk,NOTBLKS=nblk,PAGES=np,COMMENTS='string'.
```

Parameters are in keyword form.

I=*idn* Name of dataset containing debug symbol tables. The default is $DEBUG, which is created by the loader from the symbol tables produced by CFT and CAL.

O=*oan* Name of dataset to receive the listing output from the symbolic debug routine. The default is $OUT.

DUMP=*ddn* Name of dataset containing the dump of the user field. This dataset is created by the DUMPJOB control statement. *ddn* is used when the symbolic debug routine is invoked after an abort. The default is $DUMP.

TRACE=*n* Number of routine levels to be looked at in symbolic dump. DEBUG traces back through the active subprograms the number of levels specified by *n*. If this parameter is omitted or if TRACE is specified without a value, the default is 50.

SYMS=*sym* List of symbols to be dumped by DEBUG. Up to 20 symbols may be specified; symbols are separated by a colon. A shorthand notation as described in the AUDIT statement may be used; thus, a dash represents any character or characters or no character, and an asterisk represents any single character. For example:

```
... ,SYMS=ABC:X-:B**, ...
```

requests a dump of the symbol ABC, all symbols that start with X, and all 3-character symbols beginning with B. This parameter applies to all blocks dumped. The default is all symbols.

NOTSYMS=*nsym*

List of symbols to be skipped. Up to 20 symbols may be specified; symbols are separated by a colon. The shorthand notation as described under the SYMS parameter may be used. This parameter applies to all blocks dumped. The default is that no symbols are to be skipped. This parameter takes precedence over the SYMS parameter.

MAXDIM=*dim*

Maximum number of each dimension of the arrays to be dumped. This parameter allows the user to sample the contents of arrays without creating huge amounts of output. For example:

```
... ,MAXDIM=3:2:3, ...
```

causes the following elements to be dumped from an array dimensioned as A(10,3,6):

```
A(1, 1, 1)  A(2, 1, 1)  A(3, 1, 1)  A(1, 2, 1)  A(2, 2, 1)
A(3, 2, 1)  A(1, 1, 2)  A(2, 1, 2)  A(3, 1, 2)  A(1, 2, 2)
A(2, 2, 2)  A(3, 2, 2)  A(1, 1, 3)  A(2, 1, 3)
A(3, 1, 3)  A(1, 2, 3)  A(2, 2, 3)  A(3, 2, 3)
```

This parameter applies to all blocks dumped. The default is MAXDIM=20:5:2:1:1:1:1. The arrays are dumped in storage order.

BLOCKS=*blk*

List of common blocks to be included in the symbolic dump. A maximum of 20 blocks may be specified. The shorthand notation as described under the SYMS parameter may be used. All symbols (qualified by the SYMS and NOTSYMS parameters) in the blocks named here are to be dumped. If BLOCKS is specified without a value, all common blocks are dumped.

NOTBLKS=*nblk*

List of common blocks to be excluded from the symbolic dump. A maximum of 20 blocks may be specified. The shorthand notation as described under the SYMS parameter may be used. The default is to exclude no blocks. NOTBLKS specified without a value excludes all but the subprogram block. This parameter takes precedence over the BLOCKS parameter.

PAGES=*np* Page limit for the symbolic debug routine. The default is 70 pages.

COMMENT='*string*'

Identifier to be printed on the DEBUG output title line. Up to 8 ASCII characters may be specified.

DSDUMP - DUMP DATASET

The DSDUMP control statement dumps specified portions of a dataset to another dataset. The dump may be made in one of two formats: blocked or unblocked.

In the blocked format, a group of words within a record, a group of records within a file, and a group of files within a dataset may be selected. Initial word number, initial record number, and initial file number begin with 1 and are relative to the current dataset position. Specifying an initial number greater than one causes words, records, or files to be skipped starting from the current position. Since the initial word, record, or file number is relative to the current position of the dataset, the dataset must be positioned properly prior to calling DSDUMP. A rewind of the dataset prior to calling DSDUMP makes the initial word, record, and file numbers relative to the beginning of the dataset. When DSDUMP is completed, the input dataset is positioned after the last record dumped.

The unblocked format is used for dumping a dataset without regard to whether it is blocked. It is possible to dump a blocked dataset in unblocked format (by sectors). A group of sectors within the dataset or a group of words within each sector may be selected. The initial word and initial sector numbers begin with one and are always relative to the beginning of the dataset. Specifying an initial sector greater than 1 causes sectors to be skipped from the beginning of the dataset; specifying an initial word greater than one causes words to be skipped from the beginning of each sector. Following a dump in unblocked format, the dataset is closed.

Format:

```
DSDUMP,I=idn,O=odn,DF=df,IW=n,NW=n,IR=n,NR=n,IF=n,NF=n,IS=n,NS=n.
```

Parameters are in keyword form; the only required parameter is I.

I=*idn* (or DN=*idn*)
Name of dataset to be dumped. This is a required parameter.

O=*odn* (or L=*odn*)
Name of dataset to receive the dump. The default is $OUT.

DF=*df* Dump format. The default is B.
B Blocked
U Unblocked

IW=*n* Decimal number (*n*) of initial word for each record/sector on *idn*. The default is 1.

NW=*n* Decimal number (*n*) of words per record/sector to dump. Specifying NW without a value dumps all words to the end of a record/sector. The default is 1.

IR=*n* Decimal number (*n*) of initial record for each file on *idn*. Applicable only if DF=B. The default is 1.

NR=*n* Decimal number (*n*) of records per file to dump. Specifying NR without a value dumps all records to the end of the file. Applicable only if DF=B. The default is 1.

IF=*n* Decimal number (*n*) of initial file for dataset on *idn*. Applicable only if DF=B. The default is 1.

NF=*n* Decimal number (*n*) of files on *idn* to dump. Specifying NF without a value dumps all files to the end of the dataset. Applicable only if DF=B. The default is 1.

IS=*n* Decimal number (*n*) of initial sector on *idn*. Applicable only if DF=U. The default is 1.

NS=*n* Decimal number (*n*) of sectors to dump. Specifying NS without a value dumps all sectors to the end of the dataset. Applicable only if DF=U. The default is 1.

For blocked format, each record from *idn* dumped to *odn* is preceded by a header specifying the file and record number. For unblocked format, each sector is preceded by a header specifying the sector number.

Format of each dump record:

Word count	Octal interpretation of four words	Character interpretation of four words

A row of five asterisks indicates that one or more groups of four words has not been formatted because they are identical to the previous four. Only the first group is formatted. The number of words not formatted can be determined from the word counts of the formatted lines before and after the asterisks. The final group of four or less words is always formatted.

COMPARE - COMPARE DATASETS

The COMPARE control statement compares two blocked datasets and lists all differences found. The output consists of a listing of the location of each discrepancy, the contents of the differing portions of the datasets, and a message indicating the number of discrepancies. Refer to the CRAY-OS Message Manual, publication SR-0039.

Keyword parameters allow the user to specify the maximum number of errors and the amount of context to be listed.

If only parts of two datasets are being compared, the parts must first be copied before using a COMPARE statement; COMPARE compares complete datasets only.

COMPARE rewinds both input datasets before and after the comparison.

Format:

COMPARE,A=*adn*,B=*bdn*,L=*ldn*,DF=*df*,ME=*maxe*,CP=*cpn*,

CS=*csn*,CW=cw_1,:cw_2,ABORT=*ac*.

Parameters are in keyword form; both A and B must be specified.

A=*adn* and B=*bdn*
Input dataset names. If *adn*=*bdn*, an error message is issued and the job is aborted. A and B are required parameters.

L=*ldn* Dataset name for list of discrepancies. *ldn* must be different from *adn* and *bdn*. The default is $OUT.

DF=*df* Input dataset format. The default is T.

df is a 1-character alpha code as follows:

B Binary. The input datasets are compared logically to verify that they are identical. If they are not identical, the differing words are printed in octal and as ASCII characters. The location printed is a word count in decimal. The first word of each dataset is called word 1.

T Text. The input datasets are compared to see if they are equivalent as text. For example, a blank-compressed record and its expansion are considered equivalent. If the two datasets are not equivalent, the differing records are printed as text. The location is printed as a record count in decimal. The first record of each dataset is called record 1.

ME=*maxe* Maximum number of differences printed. The default is 100.

CP=*cpn* Amount of context printed. *cpn* records to either side of a difference are printed. The CP parameter applies only if DF=T; if DF=B and CP are specified, an error message is generated. The default is 0.

CS=*csn* Amount of context scanned. *csn* records to either side of a discrepancy are scanned for a match. The CS parameter applies only if DF=T; if DF=B and CS are specified, an error message is generated. The default is 0.

If a match is found within the defined range, subsequent comparisons are made at the same interval. That is, if record 275 of dataset A is equivalent to record 277 of dataset B, the next comparison is between record 276 of dataset A and record 278 of dataset B.

NOTE

If identical records occur within *csn* records of each other, the pairing is ambiguous and COMPARE may match the wrong pair.

CW=*cw* or CW=cw_1:cw_2

Compare width. If CW=*cw* is specified, columns 1 through *cw* are compared. If CW=cw_1:cw_2 is specified, columns cw_1 through cw_2 are compared. Specifying CW without a value is not permitted. The default is to compare columns 1 through 133, but this can be changed by installation option. The CW parameter applies only if DF=T; if DF=B and CW are specified, an error message is generated.

ABORT=*ac* If *ac* or more differences are found, the job step aborts. Specifying ABORT alone is equivalent to ABORT=1 and causes an abort if any differences are found. Specifying ABORT does not prevent the listing of up to *maxe* differences.

PRINT - WRITE VALUE OF EXPRESSION TO LOGFILE

The PRINT control statement writes the value of an expression on the logfile. The value of the expression is written in three different formats: as a decimal integer, as a 22-digit octal value, and as an ASCII string. PRINT is a system verb.

Format:

```
PRINT(expression)
```

Parameters:

expression
Any JCL expression (part 1, section 4). This parameter is required.

Format in the logfile:

FT060 *decimal octal ASCII*

FT060	Message code indicating origin is PRINT statement
decimal	16-digit decimal representation of evaluated expression
octal	22-digit octal representation of evaluated expression
ASCII	8-character ASCII representation of evaluated expression

FLODUMP - FLOW TRACE RECOVERY DUMP

The FLODUMP control statement recovers and dumps flow trace tables when a program aborts with flow tracing active. The flow trace tables are dumped in the FORTRAN flow trace format.

FLODUMP is invoked by specifying the F option on the CFT control statement and including the FLODUMP control statement in the COS control statement file. (Refer to the CRAY-1 FORTRAN (CFT) Reference Manual, publication SR-0009, for more information on the F option.)

Format:

```
FLODUMP.
```

Parameters: none

The following example illustrates the use of the FLODUMP control statement.

```
    JOB,....
    CFT, ON=F.
    LDR.
    EXIT.
    DUMPJOB.
    FLODUMP.
    .
    .
    .
```

A flow trace summary is illustrated in figure 8-1; a flow trace recovery dump is shown in figure 8-2.

The examples in figures 8-1 and 8-2 show that the total time reported for the main program, ONF, is larger for the flow trace recovery dump than for the flow trace summary. This difference is because the time reported with FLODUMP includes the main program's execution time, the time required to abort the program, and the time required to recover the flow trace tables.

```
 FLOW TRACE --- SUMMARY
        ROUTINE          TIME        %    CALLED    AVERAGE T


  1  ONF            0.000053    5.42         1     0.000053
                                                            CALLS SUB1
  2  SUB1           0.000323   32.80         9     0.000036 CALLED BY ONF
                                                            CALLS SUB2
  3  SUB2           0.000322   32.75         9     0.000036 CALLED BY SUB1
                                                            CALLS SUB3
  4  SUB3           0.000286   29.04         9     0.000032 CALLED BY SUB2
***     TOTAL       0.000985
***  OVERHEAD       0.000712


SUBROUTINE LINKAGE OVERHEAD SUMMARY            28 CALLS

              MINIMUM    MAXIMUM    AVERAGE    CYCLES   SECONDS          %
T REGISTERS         1          2        2.0       838  1.05E-05     1.0640
B REGISTERS         2          3        3.0       894  1.12E-05     1.1351
  ARGUMENTS         0          0        0.0         0  0.00E+00     0.0000
      TOTAL                                      1732  2.17E-05     2.1991
 MAXIMUM SUBROUTINE DEPTH =  4
```

Figure 8-1. Example of a flow trace summary

```
 FLOW TRACE RECOVERY DUMP --- RECOVER WITH ONFDMP   ACTIVE
 FLOW TRACE --- SUMMARY
        ROUTINE          TIME        %  CALLED   AVERAGE T

    1 ONFDMP       0.000328  26.04        1     0.000328
                                                          CALLS SUB1
    2 SUB1         0.000323  25.64        9     0.000036 CALLED BY ONFDMP
                                                          CALLS SUB2
    3 SUB2         0.000322  25.61        9     0.000036 CALLED BY SUB1
                                                          CALLS SUB3
    4 SUB3         0.000286  22.70        9     0.000032 CALLED BY SUB2
***      TOTAL     0.001259
***   OVERHEAD     0.000712


SUBROUTINE LINKAGE OVERHEAD SUMMARY               28 CALLS

              MINIMUM    MAXIMUM    AVERAGE      CYCLES   SECONDS
T REGISTERS         1          2        2.0         838  1.05E-05     0.83
B REGISTERS         2          3        3.0         894  1.12E-05     0.88
  ARGUMENTS         0          0        0.0           0  0.00E+00     0.00
      TOTAL                                        1732  2.17E-05     1.71
 MAXIMUM SUBROUTINE DEPTH =  4
```

Figure 8-2. Example of a flow trace recovery dump

SYSREF - GENERATE GLOBAL CROSS-REFERENCE LISTING

The SYSREF utility generates a global cross-reference listing for a group of CAL or APML programs. The number of CAL or APML programs that can be included in such a group is limited by the amount of CRAY-1 memory allocated to a user.

SYSREF reads special binary symbol tables written by CAL or APML and produces a single cross-reference listing for the program modules represented in the tables. When the X parameter appears on a CAL or APML statement, a record is written for each program unit assembled. The records are written to a dataset specified by the X parameter ($XRF by default if X appears alone). Each record has a header containing the name of the program unit. The rest of the record consists of cross-reference information for every global symbol used in that program.

Format:

```
SYSREF,X=xdn,L=ldn.
```

Parameters:

X=*xdn* Name of dataset whose first file (normally the only file) contains one or more symbol records written by CAL and/or APML. The default is $XRF.

L=*ldn* Name of output dataset. The default is $OUT.

USE OF SYSREF

SYSREF is usually used to process symbol records written by CAL and/or APML earlier in the same job. To do so, add X parameters to each CAL or APML control statement and follow them with a SYSREF control statement:

```
CAL,X.
APML,X.
CAL,X.
SYSREF,L=XROUT.
```

$XRF is used as default in all cases.

To process symbol records written in an earlier job, the following sequence is used:

The first job:

```
    CAL,X.
    APML,X.
    SAVE,DN=$XRF,ID=XX.
```

The second job:

```
    ACCESS,DN=$XRF,ID=XX,UQ.
    DELETE,DN=$XRF.
    SYSREF,L=XROUT.
```

To add more symbol records before invoking SYSREF, use:

```
    ACCESS,DN=$XRF,ID=XX,UQ.
    DELETE,DN=$XRF.
    SKIPR,DN=$XRF,NR.
    CAL,X.
    SYSREF
```

The format above has the same effect as if the CAL step had been done before the SAVE step.

GLOBAL CROSS-REFERENCE LISTING FORMAT

The global cross-reference listing contains only global symbols. A symbol is global if it is any one of the following:

- Named in an ENTRY or EXTERNAL statement
- Defined before an IDENT statement and after the preceding END statement
- Defined within a system text such as $SYSTXT
- Defined within a section of source code bracketed by TEXT and ENDTEXT pseudo instructions

The order of the symbols in the global cross-reference listing is lexicographic, based first on the symbol name and then (within each symbol name) on the module name. An exception to the order is made for symbol names beginning with N@, S@, or W@. These symbol names are sorted as if @ is the most significant (leftmost) character and the N, S, or W is the least significant character. The listing displays the symbol name correctly. The effect is a grouping of all the N@, S@, and W@ symbols that refer to the same field in a table.

The global cross-reference listing consists of 13 columns:

Column	Heading	Contents
1	Value	The symbol's value
2	Symbol	The symbol's name
3	Origin	The IDENT of the system text in which the symbol is defined; or the label of the TEXT block in which the symbol is defined; or *GLOBAL*, if the symbol is defined outside any program unit; or blank.
4	Module	The IDENT of the module within or before which the symbol is defined or referenced
5-13	References	A list of the lines on which the symbol is defined or referenced

The symbol's name, value, and references appear in the same format as in a CAL or APML listing. The page number in each reference is a local page number which starts at 1 for each module. In a CAL or APML listing, this is the page number that appears in parentheses to the right of the second title line on each page.

RELOCATABLE LOADER 9

The COS relocatable loader is a utility program that executes within the user field and provides the loading and linking in memory of relocatable modules from datasets on mass storage.

The relocatable loader is called through the LDR control statement when a user requires loading of a program in relocatable format. Absolute load modules can also be loaded. The design of the COS loader tables and relocatable loader allows program modules to be loaded, relocated, and linked to externals in a single pass over the dataset being loaded. This minimizes the time spent in loading activities on the CRAY-1. The loader allows the immediate execution of the object module or the creation of an absolute binary image of the object module on a specified dataset. Loader features are governed by parameters of the LDR control statement.

The relocatable loader can also generate a partially relocated module. This module is referred to as a relocatable overlay and is described at the end of this section.

LDR CONTROL STATEMENT

The loader is called into execution by the LDR control statement. Parameters of the control statement determine the functions to be performed by the loader.

Format:

```
LDR,DN=dn,LIB=ldn,NOLIB=ldn,LLD,AB=adn,MAP=op,SID='string',T=tra,

          NX,DEB=l,C,OVL=dir,CNS,NA,USA,L=ldn,SET=val,E=n,I=sdir.
```

Parameters are in keyword form.

DN=*dn* Dataset containing modules to be loaded. The default is $BLD. Loading continues until an end-of-file is reached. Modules are loaded according to block name as determined by a CAL IDENT card or a CFT PROGRAM, SUBROUTINE, BLOCK DATA, or FUNCTION statement. Duplicate blocks are skipped and an informative message is issued.

Multiple files from the same dataset may be loaded by specifying the dataset name multiple times separated by colons. A maximum of eight files may be indicated.

Datasets specified by the DN parameter are closed at the end of the load process. Closing a dataset has the effect of rewinding the dataset and releasing I/O tables and buffers.

Modules to be loaded may be relocatable or absolute. However, the two types of modules may not be mixed.

For example,

```
DN=LOAD1:LOAD2:$BLD
```

causes the loading of all modules in the first file of datasets LOAD1, then LOAD2, and then $BLD.

Normally the dataset is rewound before loading; however, consecutive occurrences of a dataset name inhibit subsequent rewind operations. Therefore, the statement

```
DN=LOAD3:LOAD3
```

causes the loading of all modules in the first two files of dataset LOAD3.

The DN parameter takes on a special quality when OVL is specified: only one *dn* may be specified. The dataset named will be the initial LOAD file used by the overlay loader. (See the description of the overlay loader, part 2, section 10 for more information.)

LIB=*ldn* The LIB parameter names the dataset from which unsatisfied externals are loaded. A maximum of eight datasets can be named, with the dataset names separated by colons. All datasets listed are automatically accessed if not already local; therefore, no ACCESS statement is required.

Any default libraries are automatically included in the library list unless the NOLIB parameter is specified. The loader accesses the default libraries if they are not local to the job; no ACCESS statement is required.

Datasets specified by the LIB parameter are closed at the end of the load process. Closing a dataset has the effect of rewinding the dataset and releasing I/O tables and buffers.

NOTE

These datasets should be generated using the BUILD utilities to prevent unnecessary overhead in the loader.

The libraries cannot be tape resident.

NOLIB=*ldn*

The NOLIB parameter value names the specific default library to be excluded from the load. Selecting NOLIB with no value specifies the exclusion of all default system libraries. If NOLIB is not specified, any default libraries that a site has are automatically included in the library list, along with any libraries specified on the LIB parameter.

LLD Specifying the LLD parameter causes any libraries included in the load to be retained as local datasets at load completion. These local datasets remain open. If the LLD parameter is not specified, the loader closes all libraries at load completion. Datasets automatically accessed are not released at load completion.

AB=*adn* Absolute binary object module generation. Use of this parameter causes an absolute binary object module to be written to the named dataset after the load process is completed. Selecting AB does not imply NX (no execution). Unless NX is also selected, the loaded program begins execution after the binary is generated. Specifying AB without *adn* causes the module to be written on a dataset named $ABD, the default dataset. Some other dataset may be specified by AB=*adn*. The dataset is not rewound before or after the file is written.

If the AB parameter is omitted, no binary generation occurs.

If OVL is specified on the loader statement, the OVLDN directive replaces AB; any value specified for AB is ignored in overlay mode. (See part 2 section 10 for a detailed description.)

MAP=*op* Map control. The MAP parameter causes the loader to produce a map of the loaded program on the specified dataset. MAP can take any of the following values:

ON	Produces a block list and an entry list including all cross references to each entry.
FULL	Same as MAP=ON.
OFF	No map is produced. MAP=OFF is the default.
PART	Produces a block list only. Equivalent to MAP with no value specified.

SID='*string*'

Debug routine loading. The SID parameter indicates the system debugging routines (SID) are to be loaded with the code. These routines comprise an additional binary dataset loaded after all DN specified datasets and before any libraries.

The '*string*', if provided, is passed to SID for evaluation as a control statement. The verb and initial separator are not required. For example, SID='I=IN,ECH=ELIST.' is a proper string specification (the period is a required terminator). For a complete description of SID parameters, see the Symbolic Interactive Debugger (SID) User's Guide, CRI publication SG-0056. If only SID is specified, all keyed default SID control statement parameter values are used.

T=*tra* Transfer name. The T parameter allows specification of an entry name where the loader transfers control at completion of the load. The T parameter also specifies the entry included in absolute binary object modules.

The entry name is a maximum of eight characters. If no T parameter is specified, the loader begins object program execution at either the entry specified by the first encountered START pseudo from a CAL routine or at the entry of the first main program in CFT compiled routines. If no START entries are encountered, a warning message is issued and the first entry of the first relocatable or absolute module is used.

NOTE

When the SID parameter is used, the load transfer is to the system debugger; the T parameter is ignored and a warning message is issued to the user logfile.

NX	No execution. Inclusion of this parameter inhibits execution of the loaded program.

DEB=*l*	Job Communication Block (JCB) length. The default length is 200_8. Specifying DEB without a value changes the JCB length to 3000_8.

C	Compressed load. This parameter causes loading of each module to begin at the next available location after the previous module. If this parameter is omitted, loading of modules begins on 20_8-word boundaries only (optional load).

OVL=*dir*	Overlay load. The OVL parameter indicates an overlay load sequence is specified on *dir*. (See part 2 section 10 for a detailed description of the overlay load.) If the OVL keyword is specified without a value, the loader examines the next file of $IN for an overlay load sequence. The default is no overlay load. Selecting OVL implies NX (no execution).

CNS	Crack next control statement record image. This feature allows the loader to pass parameters on to the loaded program for analysis and use during execution of the loaded program. The control statement cracked follows the LDR control statement and is not available for processing by the Control Statement Processor (CSP) after processing by the loaded program.

NOTE

When the SID parameter is specified, the CNS parameter is ignored and a warning message is written to the user logfile. SID prompts for the control statement for the code being debugged.

NA — No abort. If this parameter is omitted, a caution or higher level loader error causes the job to abort.

USA — Unsatisfied external abort. When USA is specified, the loader aborts at the end if it finds one or more unsatisfied externals. A load map listing all unsatisfied externals is produced, if called for.

L=*ldn* — Listing output. This parameter allows the user to specify the name of the dataset to receive the map output. If L=0, all output is suppressed. The default is $OUT.

SET=*val* — Memory initialization. Variables, named and blank common blocks, and storage areas defined by DIMENSION statements are set to 0, -1, or an out-of-range floating-point value during loading. The default is an installation option.

SET=ZERO Memory is set to binary zeros.

SET=ONES Memory is set to -1 (all bits set in word).

SET=INDEF Memory is set to a value that causes an out-of-range error if the word is referenced as a floating-point operand. The 1's complement of each memory address is placed in the lower 24 bits of the respective word to aid in reading register and memory dumps. An example, in octal, of the value loaded into memory word 13216 is: 0605050037740177764561.

E=*n* Lists error messages. This parameter indicates which level of loader-produced error messages are not to be listed. The user may specify one of five levels of severity, where *n* is the highest level to be suppressed. The default for this parameter is E=2.

Level	Type	Description
1	COMMENT	Error does not hinder program execution.
2	NOTE	Error probably hinders program execution.
3	CAUTION	Job aborts when load process completes unless NA is selected; program might not execute properly.
4	WARNING	Job aborts when load process completes unless NX is selected; program execution is not possible.
5	FATAL	Job aborts immediately.

Example:

E=2 suppresses COMMENT and NOTE messages and allows CAUTION, WARNING, and FATAL messages to appear. FATAL messages are never suppressed.

I=*sdir* Selective load. Modules from other datasets may be loaded according to a set of directives. *sdir* indicates the dataset containing the directives. If the I keyword is specified without a value, the directives are taken from the next file of $IN. The selective load directives are described later in this section.

LOADER ERRORS

Following is a list of the errors encountered by the loader. The errors are listed by class.

Comment:

Blank common redefined
Named common redefined smaller
Generating BUILD directory for Library

All files searched
Name included before
Name excluded before

Note:

Overlay member not found
Multiple load datasets ignored in overlay mode
Illegal map value
No start address found - first entry used
Duplicate entry loaded and ignored
Duplicate program block name encountered and skipped
Bad directory format on library dataset
Unsatisfied external
Disabled parameter selected and ignored
Dataset replaced by file DN
Invalid read, try again
No selective modules from dataset
Skip dataset included before
Invalid selective file

Caution:

Blank common address not large enough
Dataset name too long
Named common defined larger
Relocatable load module in absolute mode
Member error
Directive error
Illegal character in overlay directive
Compile error
Transfer is to SID; T parameter ignored
SID loaded; CNS parameter ignored.
Absolute load module in relocatable load

Warning:

Start entry not found
Bad XI field in External Relocation Table (XRT) table

Fatal:

More than one internal relocation block
Invalid table type
Unable to open specified dataset
Null file or abnormal table found
Invalid program block name
Initial table not Program Description Table (PDT)

LOAD MAP

Each time the loader is called, the user has the option of requesting a listing that describes where each module is loaded and what entry points and external symbols were used for loading. This listing is called a load map.

The user may specify the contents of the map or the dataset to receive the map by setting parameters of the LDR control statement to the desired values. The MAP parameter of the LDR control statement allows the user to specify the contents of the map requested. MAP=ON or MAP=FULL produces a block list and an entry list. The block list gives the names, beginning addresses and lengths of the program and subroutines loaded on this loader call; the entry list includes all cross references to each entry. MAP=PART supplies a partial map, that is, the block map only.

The load map is printed when requested even if fatal errors abort the load. In this case, the map contains only those modules loaded up to the point where the fatal load error occurred.

Figure 9-1 illustrates the load map generated by the following LDR statement:

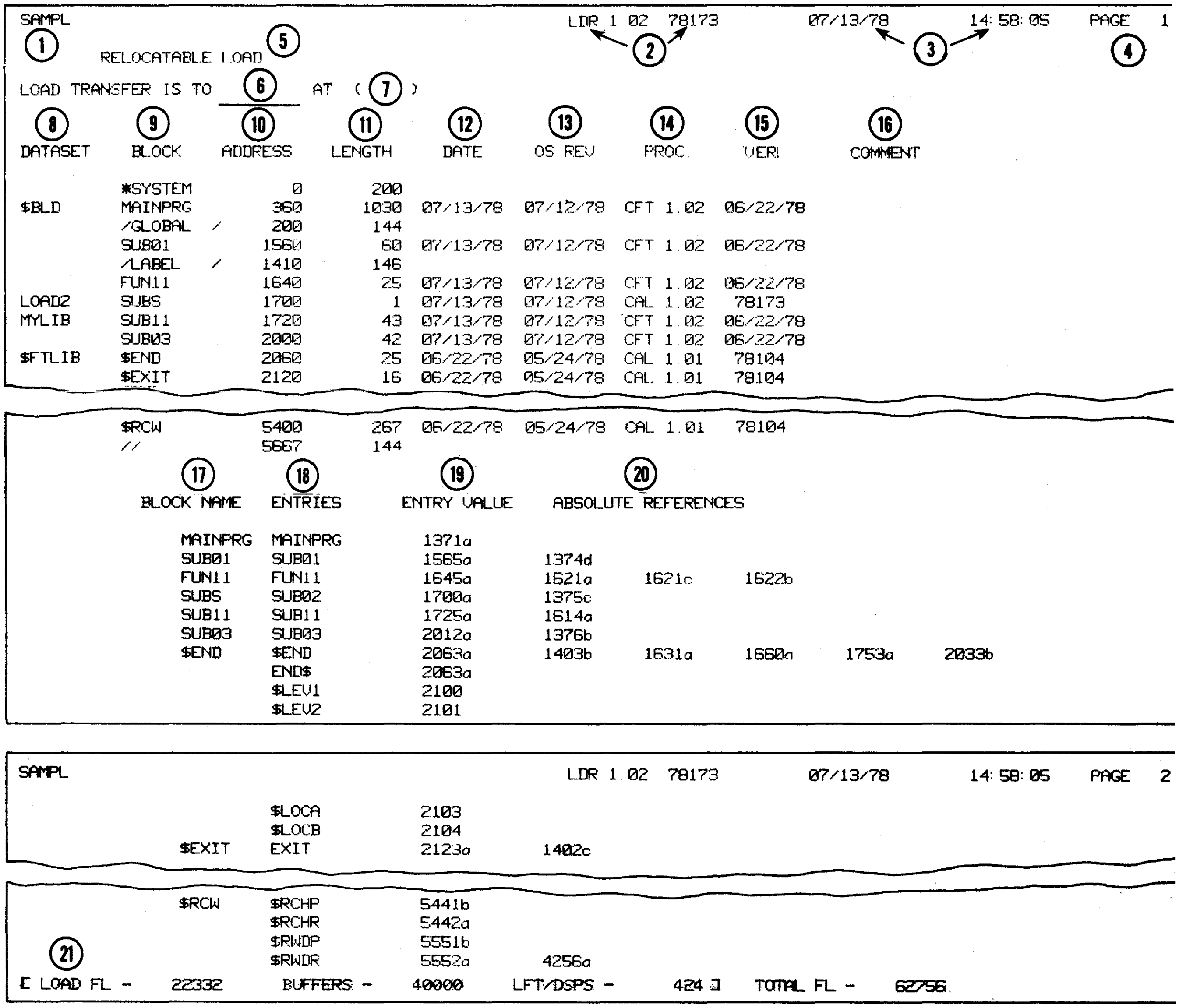
```
    LDR,DN=$BLD:LOAD2,LIB=MYLIB:$FTLIB,MAP=FULL.
```

The block list consists of items 1-16 in figure 9-1; the entry list includes items 17-21.

1. Job name from the JOB control statement
2. Loader level and Julian date of assembly of the loader
3. Date and time of loader execution
4. Page number
5. Load type; either relocatable, absolute, or overlay
6. Entry name to which initial transfer is given
7. Entry address where initial transfer is made
8. Name of load or library dataset containing modules to be loaded
9. Names of blocks loaded from the named dataset. These are common blocks (identified by the slashes around their names, for example, /LABEL/) or are names of program blocks.

*SYSTEM is always the first block listed in a relocatable load. It consists of the first 200_8 words of the user field, which is reserved for the Job Communication Block (JCB). For an absolute load, *SYSTEM is not allocated. Therefore, the CAL user must set the origin to 200_8 via an ORG pseudo instruction to allow space for the JCB. If this is not done, the job will abort.

Blank common, indicated as //, is allocated last and appears at the end of the list (if it has been defined).


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

```
SAMPL                                                       LDR 1 02  78173          07/13/78          14:58:05     PAGE   1

          RELOCATABLE LOAD

LOAD TRANSFER IS TO ______   AT (   )

DATASET     BLOCK      ADDRESS    LENGTH     DATE       OS REV      PROC.      VER!          COMMENT

            *SYSTEM          0       200
$BLD        MAINPRG        360      1030   07/13/78   07/12/78   CFT 1.02   06/22/78
            /GLOBAL  /     200       144
            SUB01         1560        60   07/13/78   07/12/78   CFT 1.02   06/22/78
            /LABEL   /    1410       146
            FUN11         1640        25   07/13/78   07/12/78   CFT 1.02   06/22/78
LOAD2       SUBS          1700         1   07/13/78   07/12/78   CAL 1.02    78173
MYLIB       SUB11         1720        43   07/13/78   07/12/78   CFT 1.02   06/22/78
            SUB03         2000        42   07/13/78   07/12/78   CFT 1.02   06/22/78
$FTLIB      $END          2060        25   06/22/78   05/24/78   CAL 1.01    78104
            $EXIT         2120        16   06/22/78   05/24/78   CAL 1.01    78104

            $RCW          5400       267   06/22/78   05/24/78   CAL 1.01    78104
            //            5667       144

             BLOCK NAME   ENTRIES       ENTRY VALUE   ABSOLUTE REFERENCES

               MAINPRG    MAINPRG         1371a
               SUB01      SUB01           1565a        1374d
               FUN11      FUN11           1645a        1621a      1621c      1622b
               SUBS       SUB02           1700a        1375c
               SUB11      SUB11           1725a        1614a
               SUB03      SUB03           2012a        1376b
               $END       $END            2063a        1403b      1631a      1660a      1753a      2033b
                          END$            2063a
                          $LEV1           2100
                          $LEV2           2101

SAMPL                                                       LDR 1 02  78173          07/13/78          14:58:05     PAGE   2

                          $LOCA           2103
                          $LOCB           2104
               $EXIT      EXIT            2123a        1402c

               $RCW       $RCHP           5441b
                          $RCHR           5442a
                          $RWDP           5551b
                          $RWDR           5552a        4256a
[ LOAD FL -    22332     BUFFERS -     40000     LFT/DSPS -      424 ]   TOTAL FL -     62756.
```

Figure 9-1. Example of a load map

10. Octal starting address of the block

11. Octal word length of the block

12. Date the object module was generated

13. Operating system revision date at the time the object module was generated

14. Name and revision level of the processor that generated the object module

15. Revision date of the processor that generated the object module

16. Comment (if any) from CAL COMMENT pseudo included in the load module

17. Name of program block referenced

18. Entry points in the program block

19. Word address, parcel address, or value of each entry point

20. Absolute parcel addresses of references to each entry point. Eight references are listed per line; some entry points have no references.

21. Size of loaded binary, amount of memory used for I/O buffers, amount of memory used for LFTs and DSPs, and total amount of memory used. Total is the minimum amount of memory needed for the program.

SELECTIVE LOAD

If the I keyword is present on the LDR control statement, one or more INCLUDE and/or EXCLUDE directives are examined in the specified dataset.

Formats:

INCLUDE,SDN=*sdn*,FN=*fn*,MOD=md_1,:md_2:...:md_{50}.

EXCLUDE,SDN=*sdn*,FN=*fn*,MOD=md_1:md_2:...:md_{50}.

Parameters are in keyword form.

SDN=*sdn* Name of dataset containing modules to be selectively loaded. If SDN is specified without a value, the first dataset specified on the DN parameter of the LDR statement is the default. If the SDN parameter is omitted, an error message results, and the directive is skipped; the load does not abort. The SDN and FN parameters must refer to the same dataset.

FN=*fn* File number of the specified dataset. A number from 0 through 7. *fn* refers to the file by its numerical position in SDN or in the DN parameter of the LDR statement.

For example, if DN=D1:D1:D2, the first file of D1 has an *fn* of 0, and the second file of D1 has an *fn* value of 1. If FN is specified without a value, the default is 0. If FN is omitted, the whole of *sdn* is searched for the correct module; a message is issued for a complete *sdn* search. The SDN and FN parameters must refer to the same dataset.

To load a module from the first file of D1, the directive may include the parameter FN=0; however, if FN is specified without a value, the default is to load a module from the first file.

MOD=*md* Module name or entry point to a module to be included or excluded from the load. Up to 50 modules can be specified; the modules must be separated by colons. If the MOD parameter is omitted, an error message results, and the directive is skipped.

Example: Given the LDR statement

```
LDR,DN=D1:D1:D2,...
```

A directive to load a module from the second file of dataset D1 would include the following directive in the next file of $IN:

```
INCLUDE,SDN=D1,FN=1,MOD=....
```

Selective load messages are never suppressed.

RELOCATABLE OVERLAYS

When a binary module is defined as a relocatable overlay, the loader can generate an image of the module that has been only partially relocated. The image of the binary module contains sufficient information for a user program to relocate all address references within the module program block according to the actual address at which the user program determines that the module should be executed.

The relocatable overlay is useful because program modules are generated in such a way that a common memory pool can execute the overlay and also any of several overlays can execute at any address within the pool.

GENERATION OF RELOCATABLE OVERLAYS

The CAL assembler defines a module as a relocatable overlay at assembly time with the MODULE pseudo-op.

Format:

Location	Result	Operand
ignored	MODULE	*type*

Parameters:

type A keyword parameter identifying the type of module being defined. RELOCOVL is the only type currently available.

When the relocatable overlay is defined by the assembler, COS sets a special flag in the Program Descriptor Table (PDT) for use by the relocatable loader.

The loader, recognizing that the current module being loaded is a relocatable overlay, performs limited relocation of the address references in the module. That is, all references to labeled COMMON blocks and all references to entry points defined within other modules are adjusted according to the address at which the other module resides in the memory image being constructed. References to blank COMMON are illegal. It is also illegal for any other module to make any reference to any entry point which is defined to be within the relocatable overlay module. References from within the module to addresses within the module are not adjusted at this time. Instead, a copy of the necessary Block Relocation Table (BRT) entries is included in the memory image of the module. All BRT entries not needed for satisfying internal references are deleted.

The absolute memory image of the program constructed by the loader will contain the loaded programs, including all relocatable overlay modules.

The relocatable overlays are physically located at the end of the memory image; all nonrelocatable overlay modules are loaded contiguously in the order in which they are encountered. Relocatable overlay modules can appear at any point in the load sequence and can be contained in libraries. The loader moves modules in memory as required to order the relocatable overlays at the end of the image. This placement of the overlays makes it possible for a user program to locate the images of each overlay and to copy the overlays to mass storage, if it is desired, in order to make the memory space used by the overlay images available for use by the program.)

MEMORY LAYOUT WHEN RELOCATABLE OVERLAYS EXIST

When the loader has detected the existence of one or more relocatable overlays, memory is laid out in the following manner.

1. All nonrelocatable modules, in the order they were encountered on load datasets or in libraries

2. Labeled COMMON blocks interspersed among the nonrelocatable modules so that a labeled COMMON block precedes the absolute image of the first block encountered which defines the block

3. All labeled COMMON blocks which are first defined within a relocatable overlay module and which are not defined within any other type of module

4. The images of all relocatable overlays in the order in which they are encountered on load datasets or in libraries

5. Unsatified external (USX) program which is the loader's internal program for processing unsatisfied external references

6. Blank COMMON if defined by any program module

Note that the placement of USX and blank COMMON can defeat the purpose of relocatable overlays, since the overlay images must remain reserved. With proper care, the program can use the space occupied by the overlay images for internal tables and other data with nonallocated space.

MEMORY LAYOUT OF A RELOCATABLE OVERLAY IMAGE

When the loader completes constructing the image of the complete program being loaded, the relocatable overlay portions have a different structure than do the nonrelocatable overlay portions. Normal modules are loaded as an absolute image with all loader-related tables removed. All address references, both internal to the module and to other modules, are adjusted so that the code executes correctly. If the C parameter is specified when the loader is called into execution, individual modules may begin immediately after the previous module, or they may begin at the next 16-word (decimal) boundary.

Because relocatable overlay modules are expected by the loader to be moved to a different address for execution, the C specification has no meaning to a relocatable overlay module, and the first and subsequent such modules begin immediately after the last word of the previous module.

Relocatable overlay module images also contain loader-relocated tables. These tables are required so that the user program can adjust address references within a relocatable overlay when it has determined the address at which the overlay will execute. The tables are:

- Program Description Table (PDT)
- Text Table (TXT)
- Block Relocation Table (BRT)

The PDT contains information regarding the number of entry points defined and the number of blocks and external references. The TXT contains a count of the words in the actual image of the code, followed by the semi-absolute image of the code. The BRT contains information necessary for adjusting address references within the module. If the user program wants to write the overlays to mass storage, the information in the PDT can be used to construct a directory or similar table for locating specific overlays or entry points, and then can be discarded. TXT and BRT must be retained in the mass storage copy for future relocation of address references.

ADDRESS RELOCATION

When a relocatable overlay has been loaded into the desired execution area, the BRT information must be used to locate all address references within the overlay. Information in the BRT includes a header with a word count and a number of words containing two relocation specifiers. Some words may contain only one specifier which must be in the left position because of the way in which the loader gathers references for the image.

The format of the header follows:

0	4	28 63
15	*wc*	//////////////////////////////

Field	Bits	Description
	0-3	Table type; 15-block relocation table.
wc	4-27	Number of words in the table, including the header

There are *wc*-1 words of relocation specifiers. Each relocation specifier word contains two 32-bit values, the format of which is:

0		8	32 63
/////	*q*	*qwa*	//////////////////////////////////

Field	Bits	Description
q	7	Relocation mode: 0 Reference requires a word address value 1 Reference requires a parcel address value
qwa	8-31	Quarter word address; indicates the parcel address of a field relative to the beginning of the overlay code image which must be modified. It consists of a 22-bit word address and a 2-bit field specifying the parcel within the word. Parcels are located within words as follows:

Field   Bits   Description

```
Parcel     Word Addr       Parcel Location

0          0-1             ----- ----- ----- ---**
           0  ***** ----- ----- -----

1          0  ---** ***** ----- -----
2          0  ----- ---** ***** -----
3          0  ----- ----- ---** *****
```

Relocation is 22-bits wide, and occurs across a word boundary if the parcel number is 0.

OVERLAY LOADING 10

INTRODUCTION

Very large programs may not fit in the available user memory space or may not use large portions of memory while other parts of the program are in execution. For such programs, the COS relocatable loader includes the ability to define and generate *overlays* -- separate modules that the user creates and then calls and executes as necessary.

Two types of overlays are available to the user, classified as either Type 1 or Type 2 depending on the directives used. *Type 1 overlays* are generated by using the generation directives ROOT, POVL, and SOVL. Two levels of overlays in addition to the root overlay are allowed with calls to a maximum of 999 adjacent overlays. *Type 2 overlays* are generated by using the generation directive OVLL. Ten levels of overlays in addition to the root overlay are allowed with calls to a maximum of 63 adjacent overlays.

The overlay loader can also generate a partially relocated module. This module is referred to as a relocatable overlay. It is described in part 2, section 9.

The overlay structure, rules for overlay generation and overlay calls for both types are described in this section. The control statements used to generate the overlay and the directives common to both types of overlays are described first. Specific rules for generation of Type 1 and Type 2 overlays are described separately in the following subsections.

OVERLAY GENERATION

Overlay generation consists of a load operation in which the loader performs relocatable loading and writes the resulting binary image to disk. One named absolute binary record is written per root and each overlay.

If the LDR control statement (part 2, section 9) has the parameter OVL=*dir*, the loader finds the overlay generation directives on the named dataset, *dir*. If no dataset is given (that is, OVL), then the loader reads overlay generation directives from $IN.

The format of the control statement is:

```
LDR,...,OVL=dir,....
```

OVERLAY DIRECTIVES

An overlay directive consists of a keyword and a parameter. A blank, comma, or open parenthesis must separate the keyword from the parameter. A period, closed parenthesis, or two consecutive blanks serve as the terminator. A caret ( ) at the end of the directive line indicates that the next line is a continuation of the current directive. The caret cannot be preceded by a blank; it must immediately follow the last character of the line.

FILE directive

The FILE directive indicates the dataset, *dn*, containing the routines to be loaded. This directive's function is similar to that of the DN parameter on the LDR control statement. It is generally the first directive on the directives dataset but may appear at any time and as often as necessary thereafter. If no FILE directive appears, the loading proceeds from the dataset specified on the DN parameter of the LDR control statement (see part 2, section 9). If that, too, has been omitted, loading initially occurs from $BLD. This directive is common to both overlay types.

Format:

```
FILE,dn.
```

OVLDN directive

The function of this directive is similar to that of the AB parameter on the LDR control statement. This directive names the dataset, *dn*, on which overlays are written. The *dn* parameter must be present. If no OVLDN directive is present, the default overlay binary dataset ($OBD) is assigned. All overlays generated following an OVLDN directive reside as separate binary records on dataset *dn*. OVLDN directives may appear as often as desired. This directive is common to both overlay types.

Format:

OVLDN,*dn*.

SBCA directive

The SBCA directive sets the blank common starting address to the specified address. This directive allows the user to place blank common after all load modules in the current overlay structure. The address specified must be larger than any address used in the overlay structure. This directive must appear before any overlay generation directive, such as ROOT or OVLL.

Format:

SBCA,*address*.

where *address* is the octal address assigned to blank common.

TYPE 1 OVERLAY STRUCTURE

Each Type 1 overlay is identified by a pair of decimal numbers, each from 0 through 999. There must be one and only one root overlay; its level numbers are (0,0). This root remains in memory throughout program execution. Primary overlays all have level numbers (*n*,0) where *n* is in the range 1 through 999.

Primary overlays are called at various times by the root and are loaded at the same address immediately following the root. A secondary overlay is associated with a specific primary overlay. The secondary level numbers are (*n*,*m*), where *n* is the primary level, and *m* is in the range 1 through 999. All secondary overlays associated with a given primary (i.e., the same *n*) are loaded at the same address immediately following that primary.

Only the root, one primary overlay, and one secondary overlay can be in memory at one time.

Figure 10-1 is a diagram of a sample Type 1 overlay loading. The primary and secondary overlays are shown in time sequence. The sequence of generation does not imply that the programs are loaded into memory in the same sequence or that they remain in memory for a set period of time when they are executed.

All external references must be directed toward an overlay nearer to the root. For example, overlay (1,0) may contain references to the root (0,0) but not to overlay (1,1). Overlay (1,1) may contain references to both (1,0) and (0,0).

The loader places named common prior to the routine that first references it. All named common references must be directed toward a lower level routine. The lowest level routine with a named common block must contain data statements for that block.

For example, in figure 10-1,

MAIN	can reference named common A only
SUB1 and SUB2	can reference named common A and B only
TEST	can reference named common A, B, and C

The loader allocates blank common immediately after the first overlay in which it is declared. If blank common is declared in the root overlay (0,0), it is allocated at the highest address of the root overlay and is accessible to all overlays. If blank common is first declared in primary overlay (1,0) and not declared in the root (0,0), then it is accessible only to the (1,*x*) overlays. Allocation and placement of blank common may also be manipulated by the user through the SBCA director.

JCHLM is set to the highest address of the root overlay prior to loading. If a subsequent overlay module requires additional memory, JCHL is reset to the highest address of that module

Type 1 overlay generation directives

The overlay generation directives define the structure of the overlay. Included in this class are the ROOT, POVL, and SOVL directives.

ROOT directive - This directive defines programs, subroutines, and/or entry points comprising the load from *dn*. For programs written in CAL, list each entry referenced. FORTRAN programs need the program name only. All members for this directive reside on the same dataset, *dn*, as defined by the FILE directive.

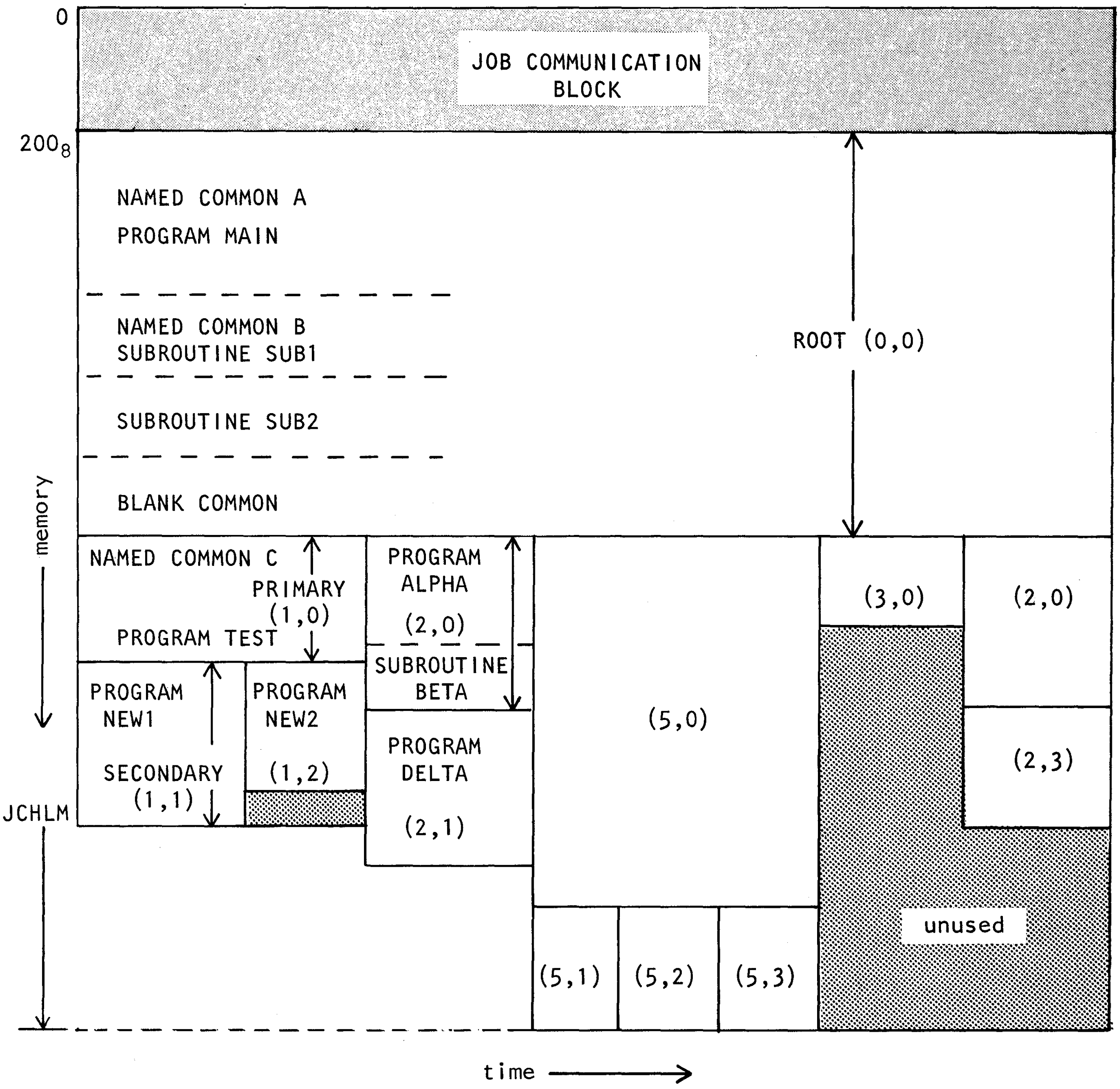


Figure 10-1. Example of Type 1 overlay loading

Format:

ROOT,*member*$_1$,*member*$_2$,*member*$_n$.

<u>POVL directive</u> - This directive causes relocatable loading of the named blocks to the primary overlay with the name *plevel*:000. The size of the root determines the base location. All members for this directive reside on the same dataset, *dn*. The first member in the list is the one that receives control when the overlay is loaded. For programs written in CAL, the first entry point of the first routine receives control.

Format:

POVL,*plevel*, *member*$_1$,*member*$_2$, ...,*member*$_n$.

where *plevel* is between 1 and 999.

<u>SOVL directive</u> - This directive causes relocatable loading of the named blocks to the secondary overlay with the name *plevel*:*slevel*. The length of POVL (*plevel*:000) determines the base location. All members for this directive reside on the same dataset, *dn*. The first member in the list is the one that receives control when the overlay is loaded. For programs written in CAL, the first entry point of the first routine receives control.

Format:

SOVL,*slevel* ,*member*$_1$,*member*$_2$,...,*member*$_n$.

where *slevel* is between 1 and 999.

Rules for Type 1 overlay generation

1. Overlay members are loaded from datasets named in FILE directives. Members are searched for in the most recently mentioned dataset only. In the absence of a FILE directive, members are loaded from the dataset specified on the LDR control statement. If that is also omitted, loading will initially occur from $BLD. Currently, the relocatable modules of all members for any overlay level must reside on the same file.

2. The overlays are generated in the order of the directives.

3. There must be one and only one root.

4. Level hierarchy must be maintained. The ROOT overlay must be generated first; hence the ROOT directives appear first. Following the ROOT generation, a primary overlay (POVL) is generated. No limitation is placed on which primary overlay number (*plevel*) is generated; however, all secondary overlays (SOVL) associated with the *plevel* must follow. The secondary overlay *slevels* may be generated in any order following their respective primary level.

5. An end-of-file in the directives file ends the input of overlay directives; hence overlay generation.

6. Any directive other than FILE, OVLDN, SBCA, ROOT, POVL, or SOVL causes a fatal error.

7. The list of members may be continued to another line by using a caret (∧) immediately following the last character at the end of the directive line (that is, no blanks). The ∧ does not replace a separator and must not appear within a member name.

8. Any number of lines may be used to name the members of an overlay.

Example of Type 1 overlay generation directives

In the following example,

DSET1 contains routines THETA, TEST, GAMMA, SUB1, MAIN, SUB2.

DSET2 contains routines NEW2, ALPHA, OVER, NEW1, DELTA, EPSILON, SIGMA, BETA.

Format of the control statement that initializes overlay generation:

```
LDR,...,OVL=OVLIN,....
```

Dataset OVLIN contains the following directives:

```
FILE,DSET1.             Loader selectively loads from dataset DSET1.

OVLDN,LEV00.            The following overlay modules are written to the
                        dataset LEV00.

ROOT,MAIN,SUB1          The absolute binary of MAIN,SUB1,SUB2 is
,SUB2.                  written as the first record on dataset LEV00.

POVL,1,TEST.            The binary of TEST is named 001:000 and is
                        binary record 2 on dataset LEV00.

FILE,DSET2.             Loader selectively loads from dataset DSET2.

SOVL,1,NEW1.            The binary of NEW1 is named 001:001 and is
                        binary record 3 on dataset LEV00.

OVLDN,LEV12.            The subsequent overlay modules are written to
                        the dataset LEV12.

SOVL,2,NEW2.            The binary of NEW2 is named 001:002 and is
                        binary record 1 on dataset LEV12.

POVL,2,ALPHA,BETA.      The binary of ALPHA,BETA is named 002:000 and is
                        record 2 on dataset LEV12.
  .
  .
  .
<eof>                   End of overlay load sequence
```

Execution of Type 1 overlays

A control statement call of the dataset containing the ROOT overlay initiates its loading and execution. If no OVLDN directives are used before generating the ROOT, the dataset $OBD will contain the ROOT overlay.

The following sequence executes the root overlay after generation:

```
LDR,...,OVL=dir,....
$OBD.
```

During overlay generation the members are loaded from the FILE dataset in the order they appear on the dataset, regardless of their order of appearance in the members list. The entry for POVL and SOVL overlays is defined by the first member listed on the generation directive. Control is transferred to this address after loading by the $OVERLAY routine during program execution. The ROOT entry may be named using the T parameter on the LDR control statement (see part 2, section 9).

Type 1 overlay calls

The user calls for the loading of overlays from within the program, and the method by which they are called depends on the program language in use (FORTRAN or CAL). OVERLAY is a subroutine of the root overlay and is loaded into memory with the root.

FORTRAN Language Call

A FORTRAN program calls for the loading of overlays as follows:

CALL OVERLAY(*n*L*dn*,level_1,level_2,*r*)

n Number of characters in the name

L Left-adjusted; zero filled

dn Name of the dataset on which this overlay resides

level_1 Primary level number of the overlay

level_2 Secondary level number of the overlay

r An optional recall parameter. If the user wishes to re-execute an overlay without reloading it, 6LRECALL may be entered. If it is not currently loaded, it will be loaded.

CAL Language Call

A sample call sequence from a CAL program is as follows:

Location	Result	Operand
	EXT	OVERLAY
	⋮	⋮
	S1	OVLDN
	S2	PLEV
	S3	SLEV
	W.OVERLAY-1,0	S1
	W.OVERLAY-2,0	S2
	W.OVERLAY-3,0	S3
	R	OVERLAY
	⋮	⋮
OVLDN	CON	A'LEV12'L
PLEV	CON	2
SLEV	CON	0

where OVLDN is the address of the dataset name, PLEV is the address of the primary level, and SLEV is the address of the secondary level. If recall is desired, the address of the literal RECALL is transmitted to W.OVERLAY-4.

Example:

Location	Result	Operand
	S4	='RECALL'L
	W.OVERLAY-4,0	S4

For both FORTRAN and CAL language calls, during execution of the ROOT(0,0) program MAIN, the statement

CALL OVERLAY(5LLEV12,2,0) or the above CAL sample call

causes OVERLAY to search dataset LEV12 for the absolute binary named 002:000. OVERLAY positions the dataset LEV12 to the location of the absolute binary named 002:000 using information supplied by the loader, loads the overlay, and transfers control to the first member specified on the POVL or SOVL directive. After execution of the overlay, control returns to the statement in MAIN immediately following the CALL statement. Following the load, dataset LEV12 is positioned immediately after the end of record for the overlay (2,0). If overlay (2,0) is not on dataset LEV12, a fatal error results.

Placing a call for a secondary overlay for which the corresponding primary overlay is not already loaded causes OVERLAY to load both overlays. Control transfers to the secondary after both overlays are in memory. A fatal error results if the primary and secondary overlays are not both on the named *ovldn*. If the overlays reside on different datasets, the user must place separate calls to load the overlays in the correct order.

Log of Type 1 overlay generation

When MAP is specified on the LDR control statement, a listing is generated that describes where each module is loaded and what entry points and external symbols were used for loading. This listing is an overlay load map and is similar to the map of a non-overlay load (part 2, section 9). A log of the directives used follows the map of the last overlay generated. If overlay loading aborts, the directives are not listed.

TYPE 2 OVERLAY STRUCTURE

A Type 2 overlay is identified by a pair of decimal numbers that indicate the overlay level and the number of the overlay within that level. The overlay notation is of the form (*level*,*number*) where the value of *level* is in the range 1 through 10 and the value of *number* is in the range 1 through 63. Only one root overlay exists; its level number is 0. The root overlay remains in memory during the entire program execution and may call only level one overlays.

Level one overlays are called at various times by the root overlay; each call loads the named overlay at the same address, which is immediately following the location of the root. The first level overlay must be called by the root; each upper level overlay may be called by the associated overlay at the adjacent lower level. A hierarchy exists among overlay levels; an upper level overlay is subordinate to the proximate lower level overlay. An upper level overlay associated with overlay (2,1) might be (3,2), (3,3) or (3,4).

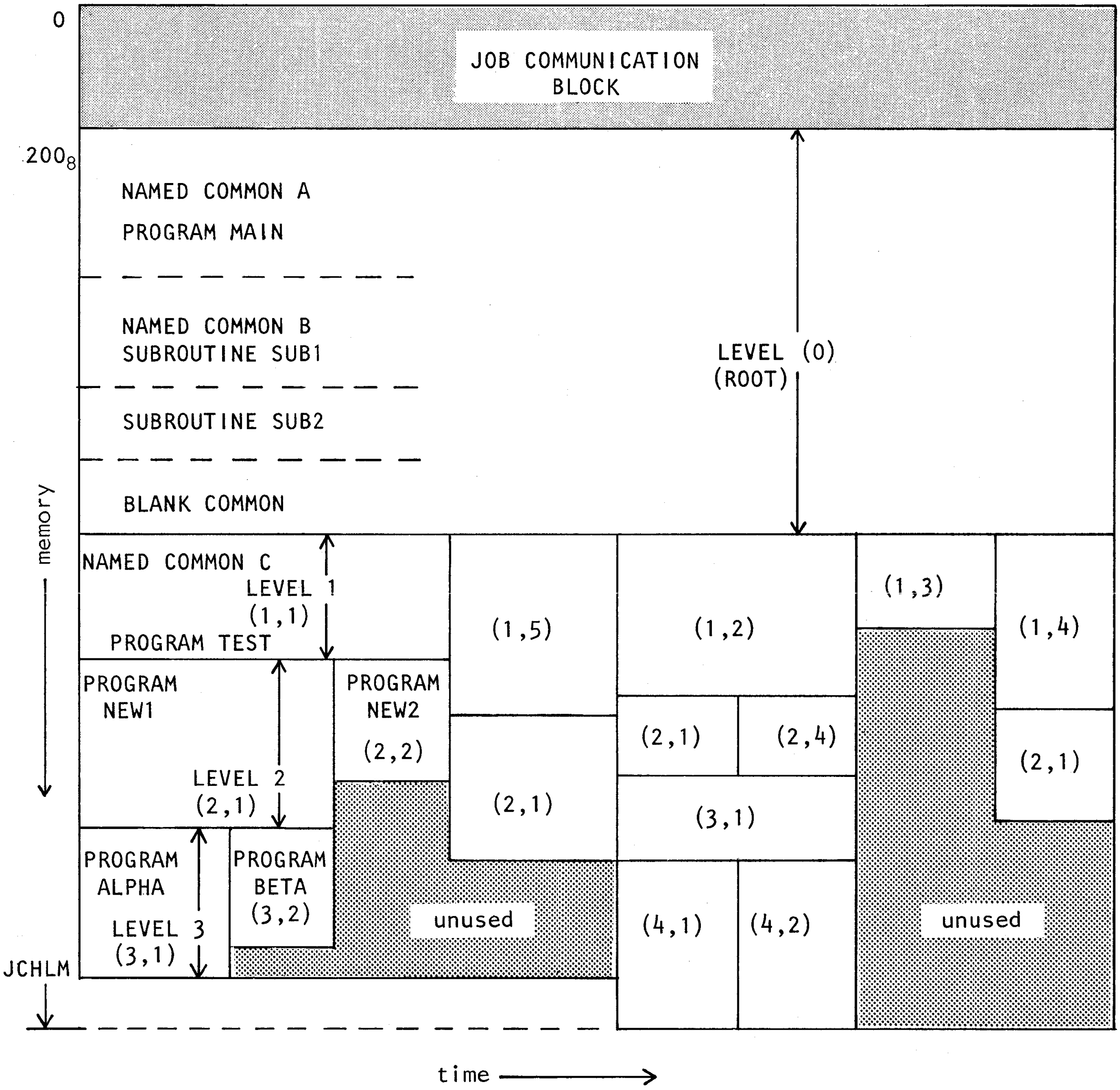


Figure 10-2. Example of Type 2 overlay loading

An overlay can call into memory any overlay in the next higher level; it cannot call an overlay more than one level above it in the hierarchy. For example, overlay (2,1) can call (3,1) through (3,63), but it cannot call (4,1). Each call for an overlay loads the named overlay at the same address location, which immediately follows the location of the calling overlay. Only the root and one overlay at each level can be in memory concurrently.

All external references must be directed toward an overlay nearer the root overlay. Overlay (1,1) may contain references to the root overlay but not to overlay (1,2) or overlay (2,1). The (2,1) overlay may reference externals in both the (1,1) overlay and the root overlay.

The loader places named common blocks prior to the routine that first references it. All named common references must be directed toward a lower level routine (toward the root overlay). If blank common is declared in the root overlay, it is allocated at the highest address of the root and is accessible to all overlays. If blank common is declared first in a level one overlay, for example, and is not declared in the root overlay, it is accessible only to level one and upper level overlays.

JCHLM is set to the highest address of the root overlay prior to loading. If a subsequent overlay module requires additional memory, JCHLM is reset to the highest address of that module.

Figure 10-2 shows a sample Type 2 overlay loading diagram. The overlays are shown in time sequence. The sequence of generation does not imply that the programs are loaded into memory in the same sequence or that they remain in memory for a set period of time when they are executed.

Type 2 overlay generation directive

The Type 2 overlay directive defines the structure of the overlay within the directive format.

OVLL directive - This directive causes relocatable loading of the named blocks of an overlay. The size of the lower level overlays in the group determines the base location. All members for this directive reside on the same dataset, dn, specified by the FILE directive. The first member in the list is the one that receives control when the overlay is loaded. For programs written in CAL, the first entry point of the first routine receives control.

Format:

```
OVLL,level,number,member1,member2,...,membern.
```

level Level number of the overlay ($1 \leq$ level ≤ 10). If *level* is 0, the root phase is generated and the *number* must be omitted.

number Number of the overlay within the level ($1 \leq$ *number* ≤ 63).

member Module names for the individual overlays

Rules for Type 2 overlay generation

1. Overlay members are loaded from datasets named in FILE directives. Members are searched for in the most recently mentioned dataset only. In the absence of a FILE directive, members are loaded from the dataset specified on the LDR control statement. If that is also omitted, loading initially occurs from $BLD.

2. The overlays are generated in the order of the directives.

3. There must be one and only one root per dataset.

4. Level hierarchy must be mai tained. The root overlay must be generated first. Following the root generation, a first level overlay is generated. No limitation is placed on which overlay number is generated; however, all overlays associated with that first level overlay must follow. The overlays may be generated in any order; the same restrictions apply for all levels of overlays (1 through 10).

5. An end-of-file ends the input of overlay directives.

6. Any directive other than FILE, OVLDN, SBCA or OVLL causes a fatal error.

7. The list of members can be continued to another line by using a caret (∧) immediately following the last character at the end of the directive line (that is, no blanks). The ∧ does not replace a separator and must not appear within a member name.

8. Any number of lines can be used to name the members of an overlay.

Example of Type 2 overlay generation directives

In the following example,

DSET1 contains routines THETA, TEST, GAMMA, SUB1, MAIN, SUB2.

DSET2 contains routines NEW2, ALPHA, OVER, NEW1, DELTA, EPSILON, SIGMA, BETA.

Format of the control statement that initializes overlay generation:

```
LDR,...,OVL=OVLIN,...
```

Dataset OVLIN contains the following directives:

FILE,DSET1.	Loader selectively loads from dataset DSET1.
OVLDN,LEV00.	The following overlay modules are written to the dataset LEV00.
OVLL,0,MAIN,SUB1, SUB2.	The absolute binary of MAIN,SUB1,SUB2 is named 0 and is the first record on dataset LEV00.
OVLL,1,1,TEST.	The binary of TEST is named 1 and is binary record 2 on dataset LEV00.
FILE,DSET2.	Loader selectively loads from dataset DSET2.
OVLL,2,1,NEW1.	The binary of NEW1 is named 101_8 and is binary record 3 on dataset LEV00.
OVLDN,LEV12.	The subsequent overlay modules are written to the dataset LEV12.
OVLL,2,2,NEW2.	The binary of NEW2 is named 201_8 and is binary record 1 on dataset LEV12.
OVLL,3,1,ALPHA.	The binary of ALPHA is named 10201_8 and is binary record 2 on dataset LEV12.

OVLL,3,2,BETA.   The binary of BETA is named 20201_8 and is binary record 3 on dataset LEV12.

.
.
.

<eof>   End of overlay load sequence.

Execution of Type 2 overlays

A control statement call of the dataset containing the root overlay initiates its loading and execution. If no OVLDN directives are used before generating the root, the dataset $OBD will contain the root overlay. All overlays reside on the datasets specified on the overlay directives. The entry for higher level overlays is defined by the first member listed on the generation directive. Control is transferred to this address after loading by the $OVERLAY routine during program execution. The root entry may be named using the T parameter on the LDR control statement (see part 2, section 9).

The following sequence executes the root overlay after generation:

```
LDR,...,OVL=dir,....
$OBD.
```

When the program is to be executed, the root overlay is brought into memory as a result of a control statement call in the job deck. Thereafter, additional overlays are called into memory by the executing program. Overlay loading allows any overlay to call for the loading of an adjacent upper level overlay.

Type 2 overlay calls

The user calls for the loading of Type 2 overlays from within the program, and the method by which they are called depends on the program language in use (FORTRAN or CAL). OVERLAY is a subroutine of the root overlay and is loaded into memory with the root.

FORTRAN Language Call

A FORTRAN program calls for the loading of Type 2 overlays as follows:

```
CALL OVERLAY(nLdn,level,number,r)
```

n Number of characters in the name

L Left-adjusted, zero filled

dn Name of the dataset on which this overlay resides

level Level number of the overlay

number Number of the overlay within the level

r Optional recall parameter. If the user wishes to re-execute an overlay without reloading it, 6LRECALL may be entered. If it is not currently loaded, it will be loaded.

CAL Language Call

Location	Result	Operand
	EXT	OVERLAY
	⋮	⋮
	S1	OVLDN
	S2	LEVEL
	S3	NUMBER
	W.OVERLAY-1,0	S1
	W.OVERLAY-2,0	S2
	W.OVERLAY-3,0	S3
	R	OVERLAY
	⋮	⋮
OVLDN	CON	A'LEV12'L
LEVEL	CON	1
NUMBER	CON	2

where OVLDN is the address of the dataset name, LEVEL is the address of the overlay level, and NUMBER is the address of the number within the level. If recall is desired, the address of the literal RECALL is transmitted to W.OVERLAY-4.

Example:

Location	Result	Operand
	S4	='RECALL'L
	W.OVERLAY-4,0	S4

For both FORTRAN and CAL language calls, during execution of the ROOT program MAIN, the statement

CALL OVERLAY(5LLEV12,1,2), or above CAL sample call

causes OVERLAY to search dataset LEV12 for the absolute binary named 2. OVERLAY positions the dataset LEV12 to the location of the absolute binary named 2 using information supplied by the loader, loads the overlay, and transfers control to the first member specified on the OVLL directive. After execution of the overlay, control returns to the statement in MAIN immediately following the CALL statement. Following the load, dataset LEV12 is positioned immediately after the end of record for the overlay 2. If overlay 2 is not on dataset LEV12, a fatal error results.

Log of Type 2 overlay generation

When MAP is specified on the LDR control statement, a listing is obtained that describes where each module is loaded and what entry points and external symbols were used for loading. This listing is an overlay load map and is similar to the map of a non-overlay load (part 2, section 9). A log of the directives used will follow the map of the last overlay generated. If overlay loading aborts, the directives are not listed.

BUILD 11

INTRODUCTION

BUILD is an operating system utility program for generating and maintaining library datasets. A *library dataset* is a dataset containing a program file followed by a directory file. Library datasets are designed primarily to provide the loader with a means of rapidly locating and accessing program modules. The program file is composed of loader tables for one or more absolute or relocatable program modules. The directory file contains an entry for each program. The entry contains the name of the program module; the relative location of the program module in the dataset; and block names, entry names, and external names.

The BUILD program constructs a library from one or more input datasets named by the user when BUILD is called. A library dataset created by a BUILD run may be used as input to a subsequent BUILD run. Through BUILD directives, the user designates the program modules to be copied from the input datasets to the new library and the order in which they are to be placed in the library. However, no directives or control statement parameters are needed for the most frequent application of BUILD, which is to add new binaries from $BLD to an existing library of binary programs, replacing the old binaries where necessary.

PROGRAM MODULE NAMES

BUILD directives refer to program modules by their names as given in the directory or, if the directory is missing or is unreadable, by the names given in the program modules.

PROGRAM MODULE GROUPS

In the COPY and OMIT directives, program modules whose names contain one or more identical groups of characters may be specified together, with the variable parts of each name relaced by one or more hyphens. For example, XYZ- represents all names beginning with XYZ, including XYZ itself. In the extreme case, a name consisting of only a hyphen represents all possible names.

In addition, up to eight asterisks may be used anywhere in a name as wild characters matching any character other than a blank. For example, GE* specifies a group of modules having three-character names including GET and GEM but not GE or GEMS.

PROGRAM MODULE RANGES

In order to facilitate the copying of large numbers of contiguous program modules, the COPY directive allows a range specifier to be used instead of a single name or group specifier. The range specifier has the general form:

(*first,last*)

which means: skip to the first module and then copy all modules from that first one up to and including the last module.

FILE OUTPUT SEQUENCE

If the SORT parameter appears in the BUILD control statement, all modules are copied alphabetically according to their new names. In the absence of a SORT parameter, modules are written in the order in which they are read originally from the input datasets.

The order of the entries in the directory is always the same as the order of the modules themselves.

FILE SEARCHING METHOD

The user need not be aware of the order of modules in the input dataset unless (1) there are two or more modules with the same name or (2) a range is specified in a COPY directive.

If two or more modules with the same name are in the input datasets, the last of the modules read is the one that survives, unless the user specifically omits that last module while its original dataset is the currently active input dataset.

The concept of *current position* in the input file is used to interpret range specifiers in which the first name is omitted as in (,   ) or (,). In such cases, the current position is defined to be either immediately after the last module copied or at the beginning of the dataset if no modules have yet been copied.

BUILD CONTROL STATEMENT

Format:

```
BUILD,I=ddn,L=ldn,OBL=odn,B=bdn,NBL=ndn,SORT,NODIR,REPLACE.
```

Parameters are in keyword form.

I=*idn* Name of dataset containing BUILD directives, if any. Directives may be included in the $IN dataset, or they may be submitted in a separate dataset.

If the I parameter appears alone or is omitted, all directives are taken from the $IN dataset, starting at its current position and stopping when an end-of-file is read.

If I=*ddn*, all directives are taken from the specified dataset, *ddn*, stopping when an end-of-file is read.

If I=0, no directives are read. The most common condition is to merge the modules from *odn* (the OBL dataset) with those from *bdn* (the B dataset), replacing OBL modules with B modules whenever the names conflict, and to write the output to *ndn* (the NBL dataset). Note that the input dataset specified by the B parameter corresponds to the binary output from CAL and CFT, also designated by B.

L=*ldn* Name of list output dataset.

If the L keyword appears alone or is omitted, list output is written to $OUT.

If L=*ldn*, list output is written to *ldn*.

If L=0, no list output is written.

OBL=*odn* Name of the first input dataset, which is usually a previously created program library.

If the OBL parameter is omitted or appears alone, the first dataset read is $OBL.

If OBL=*odn*, the first dataset read is *odn*.

If OBL=0, no old binary library exists. This is a creation run.

B=*bdn* Name of the second input dataset, whose modules will be added to or will replace the modules in the first dataset.

If the B parameter appears alone or is omitted, the second dataset read is $BLD.

If B=*bdn*, the second dataset read is *bdn*, which is read to the first end-of-file.

If B=0, no modules are being added. This run edits an old library.

BUILD stops at end-of-file; *bdn* is not required.

NBL=*ndn* Name of the output dataset, which is usually considered to be a new program library.

If the NBL parameter appears alone or is omitted, output is written to $NBL.

If NBL=*ndn*, output is written to *ndn*.

If NBL=0, no output is written. This usage is intended for checking out BUILD directives.

SORT Specifies that all modules are to be listed alphabetically according to their new names. The default is to list the modules in the order they were first read.

NODIR Specifies that no directory is to be appended to the output dataset, resulting in an ordinary sequential dataset like $BLD. The default is to append the directory.

REPLACE   Specifies that the output library is to contain modules in the same order as the old library. If omitted, the new library contains modules from the old library which were not replaced by modules from the input binary dataset, followed by modules from the input dataset.

Any of the following errors causes BUILD to abort:

- A module specified explicitly in a COPY or OMIT directive is not in the current input dataset.
- A module specified explicitly in a COPY directive has already been selected for output.
- Improper syntax is used in the BUILD control statement or in the directive dataset.
- An unrecognized directive or control statement keyword is used.
- A dataset name or module name is too long or contains illegal characters.

BUILD DIRECTIVES

BUILD is controlled through directives in a dataset defined by the I parameter on the BUILD control statement. A directive consists of a keyword and, if the keyword requires it, a list of dataset names or module names. When names are required, the keyword must be separated from the first name by a blank; subsequent names (if any) in the list are separated from each other by commas. Extra blanks are optional except within the keyword.

A line can contain more than one directive; periods or semicolons are used to separate directives on the same line from each other. A directive cannot be continued from one directive line to the next.

Examples of directives:

```
OMIT ENCODE,DECODE

COPY **CODE.
```

Examples of multiple directives on one line:

```
FROM OLDLIB; LIST; OMIT ENCODE,DECODE,XLATE

FROM $BLD. LIST.
```

FROM DIRECTIVE

A FROM directive may name a single dataset, which is thus established as the input dataset for succeeding COPY, OMIT, and LIST directives, or it may list several datasets that (except for the last dataset in the list) are to be copied in their entirety to the output dataset ($NBL). The last dataset in the list is established as the current input dataset, just as if it were specified alone in the FROM directive. If no COPY or OMIT directive follows, the last dataset is also copied in its entirety to the output dataset.

An input dataset may be a library (with a directory) or an ordinary sequential dataset (such as $BLD). BUILD always determines whether a directory is present at the end of the dataset and attempts to use it if it is there. A library dataset is treated as sequential if its directory file is unreadable for any reason.

Format:

FROM $dn_1,dn_2,\ldots,dn_n$

The following general rule allows the user to copy several datasets from one FROM directive or to omit COPY (which means "copy all") when it would be the only directive (except for OMIT directives) in the range of a particular FROM directive:

> If any dataset named on a FROM directive are not acted on by any LIST or COPY directive, then BUILD copies all of the modules belonging to that dataset. BUILD takes this action when it encounters the next FROM dataset name or the end of the directive file, whichever comes first.

If there are two input datasets to be read as soon as BUILD begins to execute (that is, if neither OBL=0 nor B=0 is specified), the modules from these two datasets are treated as if they belong to a single dataset as far as the OMIT, COPY, and LIST directives are concerned. However, if either of them is named in a FROM directive, it is treated as a separate dataset and OMIT, COPY, and LIST directives apply only to whichever is the current input dataset.

OMIT DIRECTIVE

The OMIT directive allows a user to specify that certain modules otherwise included in a group be omitted from the group on subsequent copy operations. An OMIT affects modules on the current input dataset only; its effect ends when a FROM directive is encountered.

Format:

```
OMIT fn1,fn2,...,fnn
```

Each fn_i may be one of the following:

- A single name, such as $AB@CDEF or CAB22, by which binary records can be explicitly prevented from being copied
- A group name, such as F$- or *AB**, by which binary records are prevented from being copied unless they are specified explicitly (i.e., singly) in a COPY directive (see AUDIT statement for description of * and -)

If an *fn* parameter specifies a module not in the input dataset or a group of modules having no representatives in the input dataset, a diagnostic message is included in the list output and BUILD aborts.

COPY DIRECTIVES

COPY directives cause BUILD to select the specified modules for copying from current input dataset to the output dataset. The user may specify single modules, groups of modules, or ranges of modules to be copied. If the user specifies a module that is not in the current input dataset, a diagnostic message is included in the list output and BUILD aborts.

Format:

```
COPY fn1,fn2,...,fnn
```

Each fn_i may be either of the two forms that are valid in OMIT directives:

- A single module name, by which modules are explicitly selected for copying, even if they belong to a group named in a previous OMIT directive
- A group specifier, by which all the modules in the group are selected for copying unless they were specified either explicitly or implicitly in a previous OMIT directive

In addition, two special forms are allowed for each fn_i in COPY directives:

- A form to rename a single module whose old name is specified explicitly; for example, OLDNAME=NEWNAME. (The name is changed both in the output directory and in the PDT.)
- A form to copy an inclusive range, as in (FIRST,LAST), by which all the modules in the range are selected for copying unless they were specified either explicitly or implicitly in a previous OMIT directive.

These two forms are mutually exclusive. A module copied by being included in a range cannot at the same time be renamed. Nor can either form accept a hyphen or asterisk specifying a group of modules.

Examples:

BUG=ROACH	Copies BUG, renaming it to ROACH
(LOKI,THOR)	Copies all modules from LOKI through THOR
(THOTH,)	Copies all modules from THOTH to the end of the input dataset
(,ISIS)	Copies all modules from the current dataset position through ISIS
(,)	Copies all modules from the current dataset position to the end of the input dataset

The current dataset position is defined as the beginning of the input dataset if no modules have been selected for copying yet, or else as the beginning of the record immediately after the last module that has been selected for copying.

LIST DIRECTIVE

The LIST directive tells BUILD to list the characteristics of the modules in the current input dataset. Its effect is immediate. (BUILD's standard list output describes the contents of the output dataset and is produced at the end of the run so as not to interfere with output triggered by LIST directives.)

Format:

```
LIST
```

EXAMPLES

The following are examples of various uses of the BUILD program:

- Creating a new library dataset, using as input whatever binary modules have been written out to $BLD by CAL and/or CFT.

  Control statements:

  ```
  BUILD,OBL=0,I=0.
  SAVE,DN=$NBL,PDN=MLIB.
       .
       .
  ```

- Adding one or more modules to an already existing library dataset, again taking the input from $BLD.

  Control statements:

  ```
  ACCESS,DN=$OBL,PDN=MYLIB.
  BUILD,I=0.
  SAVE,DN=$NBL,PDN=MYLIB.
       .
       .
  ```

  Any modules whose names were already in the directory of MYLIB are replaced by the new binaries from $BLD in the new edition of MYLIB that is created by BUILD and saved by the SAVE control statement.

- Merging several libraries.

  Control statements:

  ```
  ACCESS,DN=LIBONE,PDN=HERLIB.
  ACCESS,DN=LIBTWO,PDN=HISLIB.
  ACCESS,DN=ANOTHER,PDN=ITSLIB.
  ACCESS,DN=LASTONE,PDN=MYLIB.
  BUILD,I,OBL=0,B=0.
  SAVE,DN=$NBL,PDN=NEWLIB.
        .
        .
  ```

  Directives:

  ```
  FROM LIBTWO,ANOTHER,LIBONE,LASTONE
  ```

  The order of the dataset names in the FROM directives, not the order of the ACCESS control statements, determines the order of processing. If two datasets contain modules of the same name, the surviving module is the one in the dataset whose name occurs later in the FROM directive. (Any module could be renamed before input from a succeeding dataset is begun, in order to prevent it from being discarded.)

- Deleting a program module from a library.

  Control statements:

  ```
  ACCESS,DN=$OBL,PDN=MYLIB.
  BUILD,B=0.
  SAVE,DN=$NBL,PDN=MYLIB.
        .
        .
  ```

  Directive:

  ```
  OMIT BADPROG
  ```

- Extracting a program module from a library for input to the system loader, using the local dataset name $BLD as the intermediate file.

  Control statements:

  ```
  ACCESS,DN=XXX,PDN=MYLIB.
  BUILD,I,OBL=XXX,B=0,NBL=$BLD,NODIR.
        .
        .
  ```

  Directive:

  ```
  COPY RUNPROG
  ```

PART 3

MACRO INSTRUCTIONS

CONTENTS
PART 3 MACRO INSTRUCTIONS

INTRODUCTION 1

Included with the CRAY-1 Operating System is a set of macro instructions that provide the user with a means of communicating with COS. These macro instructions are available only when programming in the CAL assembler language and are processed by the assembler using macro definitions defined in the system text, $SYSTXT. The code generated by the macros represents a call to a system task or a system-provided subroutine, or it generates a table.

The format for a macro instruction is:

Location	Result	Operand
loc	*name*	$a_1,a_2,\ldots,a_j,f_1=b_1,f_2=b_2,\ldots,f_k=b_k$

loc Location field argument. Certain macros require an entry in this field. For other macros, *loc* is an optional symbolic program address. Macros that generate a table are assigned a word address; macros that generate executable code are assigned a parcel address.

name Name of macro as given in system text

a_i Actual argument string corresponding to positional parameter in prototype. Two consecutive commas indicate a null string.

$f_j=b_j$ Keyword and actual argument; these entries can be in any order. A space or comma following the equal sign indicates a null string.

{ } Stacked items within braces signify that one and only one of the listed items must be entered.

A parameter shown in all UPPERCASE letters must be coded literally as shown. A parameter presented in *italics* must be supplied with a value, a symbol, an expression, or a register designator as indicated in the text following the format for each macro.

A macro can be coded through column 72 of a line. It can be continued on the next line by placing a comma in column 1 of the next line and resuming the parameter list in column 2, with no intervening blanks at the end of the first line.

NOTE

Use the A0 and S0 registers as parameters with care. When a macro that includes A0 or S0 as a parameter is expanded, special syntax values are used rather than the value of the contents of A0 or S0.

SYSTEM ACTION REQUEST MACROS 2

The system action request macros are a subset of the system function requests. Each macro generates a function code that is a call to the operating system. The octal function value is stored in register S0; S1 and S2 provide additional arguments for some requests. The function is enabled when the program exit instruction is executed. The contents of the registers used are not restored after the call is completed.

See the COS EXEC/STP/CSP Internal Reference Manual, CRI publication SM-0040 for more information on system function codes.

The system action request macros can be divided into five main classes: those involved in job control, those related to dataset management, those representing requests for time or date, those that are debugging aids, and miscellaneous. Any macro that generates executable code can have a label.

JOB CONTROL

Several system action request macros allow the user to set operating characteristics and control job processing. These include MEMORY, MESSAGE, CSECHO, MODE, SWITCH, JTIME, RECALL, DELAY, ABORT, SETRPV, ENDRPV, ROLL, ENDP, NORERUN, RERUN, IOAREA, and DUMPJOB.

MEMORY - REQUEST MEMORY

The amount of memory assigned to the job may be determined or changed by the memory request. If the user area is expanded, the additional memory is set to an installation-defined value before control returns to the user. The job is aborted if filling the request would exceed the maximum allowable memory for the job.

Format:

Location	Result	Operand
	MEMORY	*address*

address A symbol or an A, S, or T register (except A0 or S0) that contains the request word address

The format of the word at location *address* is as follows:

0		2		7		16	40 63
M	/	L	///	T	////	DEL	WC

Field	Bits	Description
M	0	Maximum memory flag. If set by the caller, the system returns in WC the maximum allowable amount of memory (in words) not including the Job Table Area (JTA). No memory is allocated.
L	2	Limit flag. The system sets this flag when the job has received the maximum allowable amount of memory.
T	7	Total flag. If T is set, WC represents the total memory requested (excluding the JTA) rather than an increment or decrement. If T is specified, DEL is ignored.

Field	Bits	Description
DEL	16-39	Deletion pointer. If the caller wants an increase in memory, DEL must equal 0. If the caller wants a decrease in memory, DEL must contain the address relative to the user's base address of the beginning of the area to be deleted.
WC	40-63	Word count. Here, if T=0, the caller must supply the absolute number of words to be added to or deleted from the user area. If T=1, the caller must supply the total field length desired. If T=0 and WC=0, no action is taken other than to return the user's field length as described below. In the memory request word, L may be set by the system as described above. When WC and T equal 0, the system sets WC to the current total number of words in the user's field length. The total number of words in the user's field length does not include the Job Table Area but does include the I/O buffers and tables.

MESSAGE - ENTER MESSAGE IN LOGFILE

The printable ASCII message at the location specified in the macro call is entered in the job and system logfile. The message must be 1-80 characters terminated by a zero byte. A flag, *loc*, indicates the destination for the message.

Format:

Location	Result	Operand
	MESSAGE	*address,loc,msgclass,override*

address A symbol or an A, S, or T register (except A0, S0, and S2) that contains the starting address of the ASCII message *loc* Destination for message. Can be any of the following:

U	User logfile only
S	System logfile only
US	User and system logfiles; default if *loc* is blank

loc can be a symbol or an A, S, or T register (except A0, S0, S3, or S4).

msgclass Class where the message is to be assigned. Only current class is JCLMSG. *msgclass* can be a symbol or an A, S, or T register (except A0, S0, S2, S3, or S4) containing the message class.

override Message suppression override flag; if present message is to go to $LOG regardless of ECHO status.

CSECHO - ECHO A CONTROL STATEMENT TO THE LOGFILE

The control statement at the specified location is entered into the system log and user logfile. This macro will not echo the control statement to the user logfile if the statement originated as terminal input from an interactive job. Echoing is also governed by the current ECHO state for JCL statements. (See part 2, section 2, ECHO control statement.)

Format:

Location	Result	Operand
	CSECHO	*address*

address A symbol or an A, S, or T register (except S0, S1, S2) that contains the base address of the control statement image. This is a required parameter.

MODE - SET OPERATING MODE

The MODE macro sets the floating-point error flag in the M register of the job's exchange package. This flag controls whether or not a floating-point error will cause an interrupt flag to be set in the Flags (F) register. An exit from the program occurs on a floating-point error only when the floating-point error flag has been set.

Format:

Location	Result	Operand
	MODE	M=*mode*

M=*mode* Operating mode. May be any of the following:

DFI, 1, or 2	Disable floating-point interrupt
EFI, 3, or 4	Enable floating-point interrupt

SWITCH - SET OR CLEAR SENSE SWITCH

The SWITCH macro allows a user to turn on (set) or turn off (clear) pseudo sense switches.

Format:

Location	Result	Operand
	SWITCH	*n*,*x*

n Number of switch (1-6) to be set or cleared

x Switch position

ON	Switch *n* is turned on; set to 1
OFF	Switch *n* is turned off; set to 0

JTIME - REQUEST ACCUMULATED CPU TIME FOR JOB

The accumulated CPU time for the job is returned at the location specified in the macro call. The time in seconds is expressed in floating-point form.

Format:

Location	Result	Operand
	JTIME	*address*

address A symbol or an A, S, or T register that contains the address at which to return the accumulated CPU time

RECALL - RECALL JOB UPON I/O REQUEST COMPLETION

This function removes a job from processing. The job does not become a candidate for processing until the previously issued I/O request for the specified dataset is completed or partially completed, that is, the job is resumed when another block of data is transferred to or from the user's buffer or when the I/O request is completed.

Format:

Location	Result	Operand
	RECALL	*address*

address Symbolic address of the Open Dataset Name Table (ODN) or Dataset Parameter Area (DSP) for this dataset or an A, S, or T register containing the ODN or DSP address. See description of OPEN macro (this section) and DSP table (Appendix A).

DELAY - DELAY JOB PROCESSING

This function removes a job from execution and delays the job from becoming a candidate for processing until the number of milliseconds (specified in the word at the given address) has elapsed.

Format:

Location	Result	Operand
	DELAY	*address*

address A symbol or an A, S, or T register containing the address of the word that contains the number of milliseconds to delay

ABORT - ABORT PROGRAM

The ABORT request provides for abnormal termination of the current program. Processing resumes with the first job control statement following the next EXIT statement unless reprieve processing is enabled. If no such statement exists, the job is terminated.

Format:

Location	Result	Operand
	ABORT	

SETRPV - SET JOB STEP REPRIEVE

The SETRPV request enables the user's current job step to maintain control when a job step abort error condition occurs or upon normal termination of the job step. Once enabled by the user, reprieve processing remains in effect until the job step terminates, a selected error condition occurs, or the user clears the reprieve processing capability.

If a selected error condition occurs, the user is reprieved from the normal or abnormal job step termination. The reprieve processing code that is given control may attempt a recovery or continue with the normal or abort termination.

I/O errors from $SYSLIB or $FTLIB are not readily recognizable or correctable. At the $FTLIB level, FORTRAN I/O usually involves three steps: initialization, transfer, and termination. I/O errors almost always occur at the transfer stage; because termination does not occur in this case, any further attempts at initialization fail, thus hampering correction. Any errors reported by the logical I/O routines look like user-requested aborts.

Two types of error conditions are related to a job step: nonfatal and fatal. Nonfatal error conditions may be reprieved any number of times per job step by the user. Each fatal error condition may be reprieved only once per job step. The second occurrence of the same fatal error condition results in an immediate termination of the job step.

Refer to Appendix F for system error codes and the mask value for each code.

Format:

Location	Result	Operand
	SETRPV	*entry,xpsave,mask*

entry Address to which control is passed if reprieve processing is selected for the respective error conditions.

xpsave First word address (FWA) of the area into which the system copies the user's exchange package when control is passed to the user's reprieve processing code. This area is formatted as follows, and the contents are those at the time of the error.

1 . . . 16	XP
17	VMR
18	ESW
19	SEC
20 . . . 40	Reserved for system use

XP User exchange package (refer to Appendix E)

VMR User vector mask register

ESW Error status word. Contains the octal value of the error category reprieved (refer to description of mask).

SEC Actual system error code (refer to Appendix F)

mask Address of a user specified octal value indicating the class(es) of error condition(s) for which to enable reprieve processing. Any number of classes may be specified by combining the appropriate octal *mask* values.

Class (Octal mask value)	Reprievable Error Condition
0	Disable user reprieve processing
1	Normal job step termination
2	User requested abort
4	System abort
10	Operator DROP
20	Operator RERUN
40	Memory error
100	Floating point error
200	Time limit
400	Mass storage limit exceeded
1000	Memory limit exceeded
2000	Link transfer error
4000	Security violation
10000	Interactive console 'attention' interrupt

NOTE

The system disables reprieve processing once the user's reprieve processing code gains control. To be reprieved from future error conditions the user must issue another SETRPV request.

CONTRPV - CONTINUE FROM REPRIEVE CONDITION

The CONTRPV macro continues normal job processing from within a reprieve subroutine. The program address to continue processing and all A, S, and VL register values are taken from the user-supplied exchange package.

Format:

Location	Result	Operand
	CONTRPV	XP

ENDRPV - END REPRIEVE PROCESSING

The ENDRPV request is used to return to job step termination processing. If the step completed normally, normal termination is completed. If the step aborted, abort processing is resumed.

Format:

Location	Result	Operand
	ENDRPV	

ROLL - ROLL A JOB

A user can protect a job against system interruption via the ROLL request. Rolling a job causes it to be written to disk so that the job at that point in time can be recovered in the event of a system interruption. Once a job has been rolled, it remains recoverable unless it loses the recoverable status (by altering a permanent dataset, for example). Once a job loses its recoverable status, the user may request another ROLL to continue to protect the job against system interruption.

Format:

Location	Result	Operand
	ROLL	

ENDP - END PROGRAM

The ENDP request is used for normal termination of the current program. Processing resumes with the next job control statement if reprieve processing is not enabled for normal job step termination. If no such statement exists, the job is terminated.

Format:

Location	Result	Operand
	ENDP	

NORERUN - CONTROL DETECTION OF NONRERUNNABLE FUNCTIONS

The NORERUN request instructs the system to begin or cease monitoring of user operations for nonrerunnable functions. This request determines whether execution of such functions will make the job become nonrerunnable but does not affect the current rerunnability of the job.

Format:

Location	Result	Operand
	NORERUN	*parameter*

parameter

ENABLE causes the system to begin (or continue) monitoring user functions for nonrerunnable operations

DISABLE causes the system to stop monitoring user operations for nonrerunnable functions

A symbol identifying a location or an A, S, or T register containing the address of a location which contains either a 0 for ENABLE or a 1 for DISABLE.

RERUN - UNCONDITIONALLY SET JOB RERUNNABILITY

The RERUN request instructs the system to mark the job as either rerunnable or nonrerunnable regardless of functions previously performed. The future declaration of nonrerunnability is not affected.

Format:

Location	Result	Operand
	RERUN	*parameter*

parameter

ENABLE causes the system to mark the job rerunnable

DISABLE causes the system to mark the job not rerunnable

A symbol identifying a location or an A, S, or T register containing the address of a location which contains either a 0 for ENABLE or a 1 for DISABLE.

IOAREA - CONTROL USER ACCESS TO I/O AREA

The IOAREA request instructs the system to either allow or deny access to the user's I/O and Dataset Parameter Area (DSP) areas. This request can also be used to restore the status of these areas to their initial status. Initially, the user I/O area is assumed to be unlocked.

Format:

Location	Result	Operand
	IOAREA	*key*, *save*

key May equal any of the following:

LOCK Denies access to the user's I/O buffers and DSP area. The Limit Address is set to the address specified in JCDSP. (All user logical I/O calls which require access to the DSP area or I/O buffers involve an exchange to the operating system before and after I/O processing.)

UNLOCK Gives full access to the user's I/O buffers and DSP area. The Limit Address is set to the value specified in JCFL.

RESTORE Reserved for use by the FORTRAN library. If UNLOCK was used previously to unlock the I/O area, then RESTORE locks the area.

save Symbolic address where lock status is to be stored; required only if RESTORE is to be used. The current status of *key* is stored in one word.

DUMPJOB - DUMP JOB IMAGE

The DUMPJOB request causes the current job image (including the Job Table Area) to be written to a specified local dataset. If the dataset already exists, it is rewound; otherwise, a new dataset is created for the dump. The dump is formatted as suitable for the dump utility.

Format:

Location	Result	Operand
	DUMPJOB	DN=*dn*

DN=*dn* A symbol or an A, S, or T register (not A0 or S0) containing the address of a dataset name. If *dn* is not specified, $DUMP is assumed. If location *dn* is not defined, the DUMPJOB macro generates the symbolic location.

DATASET MANAGEMENT

The system action request macros involved with dataset management allow the user to open datasets; set up tables; and close, release, or dispose datasets. System action request macros available include DSP, OPEN, CLOSE, RELEASE, DISPOSE, and SUBMIT.

DSP - CREATE DATASET PARAMETER AREA

The DSP macro creates a table in the user field called the Dataset Parameter Area (DSP). This table holds information concerning the status of the named dataset and the location of the I/O buffer for the dataset. The DSP is illustrated in Appendix A of this manual.

The DSP macro should be used only when the user needs the DSP and I/O buffer in the user-managed memory portion of the job. Normally a DSP and buffer for a dataset are created in the high end of the job's memory (above JCHLM) by execution of an OPEN macro.

When using the DSP macro, the user must also set up a two-word Open Dataset Name Table (ODN). This ODN must be defined before using an OPEN macro specifying this dataset.

The DSP macro is not executable; it merely sets up a DSP table with the dataset name, first, in, out, and limit fields initialized. An OPEN macro must be executed to make the DSP known to the system.

Format:

Location	Result	Operand
loc	DSP	*dn,first,nb*

loc Symbolic address of DSP. If *loc* is not specified, the address of the dataset name is generated.

dn Dataset name

first Address of the first word of the user-allocated buffer for this dataset

nb Number of 512-word blocks in the dataset buffer

Example:

Location	Result	Operand	Comment
1	10	20	35
x	DSP	XFIL,BUF,1	
ODN	CON	'XFIL'L	ASCII name
	CON	XFIL@	Address of DSP
	.		
	.		
	.		
	OPEN	ODN,I	

OPEN - OPEN DATASET

The OPEN macro prepares a dataset for processing. When an OPEN macro is executed, the dataset is made known to the system if it is not an existing dataset. I/O tables are created in the high end of the job's memory; included are the Dataset Parameter Area (DSP) and the Logical File Table (LFT). An I/O buffer is created if the dataset is COS blocked format, but not for an unblocked dataset. The address or offset of the DSP table is returned to the user.

An OPEN macro may be executed on a dataset that is already open.

Format:

Location	Result	Operand
	OPEN	*dn,pd,ldt*

dn Dataset name. The OPEN macro generates a 2-word Open Dataset Name Table (ODN) the first time an OPEN of the dataset is encountered, unless the user has previously generated an ODN for the dataset. (The ODN is illustrated in Appendix A.) The *dn* becomes the symbolic address of the ODN. It is used in all references to the dataset in other I/O requests.

As an alternative, *dn* may be an A, S, or T register (not A0, S0, or S2) containing the ODN address.

pd Processing direction. May be any of the following:

I	Dataset opened for input
O	Dataset opened for output
IO	Dataset opened for input/output (default)

pd may alternatively be an S or T register (but not an A register) with bit 0 set for input and/or bit 1 set for output.

ldt Label Definition Table (LDT); an optional parameter that is the name of a previously defined LDT for tape processing. The pointer to this field will be placed in the ODN built by the macro. The parameter applies to tape datasets only.

If the DSP pointer in the ODN is negative or zero, the OPEN call returns the negative DSP offset in the DSP field of the ODN. The actual DSP address is equal to (JCDSP) - negative DSP offset, where (JCDSP) is the value of the JCDSP field of the Job Communication Block. The negative DSP offset of a dataset does not change when a job's field length changes or as additional datasets are opened or closed.

If the DSP pointer in the ODN is positive and greater than zero, OPEN assumes the DSP field contains the address of the user's own DSP in the user field between the Job Communication Block and (JCHLM) (the value in the JCHLM field of the JCB). The system uses the DSP indicated and does not allocate an additional DSP or buffer in the job's I/O table area. The DSP indicated must already contain the buffer pointers and must indicate a buffer also within the user field. If the dataset is memory resident, this buffer should be large enough to contain the entire dataset plus one block.

Examples:

1. In the following example, the OPEN generates an ODN for dataset DSETONE unless one has been previously generated for that dataset. The dataset is opened for input/output processing.

Location	Result	Operand	Comment
1	10	20	35
L	OPEN	DSETONE,IO	

2. In this example, the address of the ODN generated by this OPEN call is passed via register S1; S2 contains processing direction information.

Location	Result	Operand	Comment
1	10	20	35
	OPEN	S1,S2	

3. In this example, the dataset ATAPE is opened for output with LABELX as the Label Definition Table. An ODN for ATAPE has not yet been defined.

Location	Result	Operand	Comment
1	10	20	35
	OPEN	ATAPE,O,LABELX	

CLOSE - CLOSE DATASET

CLOSE releases the buffer and the Dataset Parameter Area (DSP) for a COS-managed dataset. Disk space is not released and the dataset remains attached to the job.

The buffers are flushed if all of the following conditions are true for the dataset:

1. The dataset is currently opened for output.
2. No end-of-data is written.
3. The dataset is being written sequentially.
4. The dataset's DSP is managed by COS.
5. It has COS blocked dataset structure.
6. It is not memory resident.

Format:

Location	Result	Operand
	CLOSE	*dn*

dn Dataset name. Symbolic address of the Open Dataset Name Table (ODN) for this dataset or an A, S, or T (not A0 or S0) register containing the address of the ODN. See description of OPEN macro.

RELEASE - RELEASE DATASET TO SYSTEM

The dataset whose Dataset Parameter Area (DSP) address is at the location specified in the macro call is returned to the system. The dataset is closed and the Dataset Name Table (DNT) entry is released. Additional system action depends on the type of dataset. Output datasets are routed to a front end. If a dataset is not a permanent dataset, the disk space associated with that dataset is returned to the system. The dataset is no longer attached to the job.

Format:

Location	Result	Operand
	RELEASE	*address*,HOLD

address Symbolic address of the Open Dataset Name Table (ODN) or Dataset Parameter Area (DSP) for this dataset or an A, S, or T register containing the ODN or DSP address. See description of OPEN and DSP macros (this section) and of DSP format (Appendix A).

HOLD Hold generic device; optional parameter. If specified, the generic system resource associated with this dataset will not be made available to another job when the dataset is released. This parameter is for tape datasets only and is ignored for mass storage datasets.

DISPOSE - DISPOSE DATASET

The DISPOSE macro places a dataset in the appropriate queue as defined by the PDD macro.

Format:

Location	Result	Operand
	DISPOSE	*pddtag*

pddtag Address of PDD macro call

SUBMIT - SUBMIT JOB DATASET

The SUBMIT macro places a job dataset into the CRAY-1 job input queue.

Format:

Location	Result	Operand
	SUBMIT	*pddtag*

pddtag Address of PDD macro call

TIME AND DATE REQUESTS

Several system action request macros inform the user of the current time or date and the Julian date. These include TIME, DATE, and JDATE.

TIME - GET CURRENT TIME

The current time in ASCII is returned at the location specified in the macro call. The format of the time is as follows:

0 15	23	39	47	63
h h	:	m m	:	s s

Format:

Location	Result	Operand
	TIME	*address*

address A symbol or an A, S, or T register that contains the destination address of the current time

DATE - GET CURRENT DATE

The current date in ASCII is returned at the location specified in the macro call. The format of the date is as follows:

0 15	23	39	47	63
m m	/	d d	/	y y

The order can be changed to day, month, and year (the European format) through an installation parameter.

Format:

Location	Result	Operand
	DATE	*address*

address A symbol or an A, S, or T register that contains the destination address of the current date

JDATE - RETURN JULIAN DATE

The current Julian date in ASCII is returned at the location specified in the macro call. The format of the date is as follows:

0					40		63
y	y	d	d	d	Δ	Δ	Δ

Five ASCII characters are left-adjusted with blank fill in the reply word. The first two characters are the year; the next three are the number of the day in the year.

Format:

Location	Result	Operand
	JDATE	*address*

address A symbol or an A, S, or T register that contains the destination address of the current Julian date

DEBUGGING AIDS

The system action request macros in this category permit the user to selectively read or write information during a program run to aid in the debugging process. Included are the SNAP, DUMP, INPUT, OUTPUT, FREAD, FWRITE, UFREAD, UFWRITE, SAVEREGS, and LOADREGS macros. The first four of these macros can be made conditional using the label DEBUG.

SNAP - TAKE SNAPSHOT OF SELECTED REGISTERS

The SNAP macro writes the contents of selected registers under the control of FORTRAN-style formats selected by the user.

The macro generates exactly three words of inline code; the rest of the logic is in a unique subroutine created by the macro.

The DEBUG option allows conditional execution of the SNAP macro. If the label on the SNAP statement is DEBUG, no label is defined for the generated code. Instead, unless the symbol DEBUG has been set to 1 by a previously assembled SET or equate statement, code generation within the macro is suppressed entirely.

Format:

Location	Result	Operand
	SNAP	(*list*),UNIT=*unit*,AF=*fmt*,BF=*fmt*, SF=*fmt*,TF=*fmt*,VF=*fmt*,VL=*n*

list A list of registers and register groups separated by commas. The list need not be enclosed in parentheses if it contains only one element. Within the list, null elements are ignored so that each element can be preceded and followed by blanks. However, an element cannot contain embedded blanks. Each element of the list that is not null must have one of the following forms:

R Writes the contents of all *R* registers (where *R* is A, B, S, T, or V)

R_i Writes the contents of register R_i (for example, A7)

R_{i-j} or R_i-R_j
Writes the contents of registers R_i through R_j (for example, A1-A4 or A1-4)

Each i or j must be either an octal number or a previously defined register designator (for example, B.SEP).

There is no limit to the number of elements in the list or to the number of occurrences of a particular register. If the list is empty, no output is produced except for the usual header. The header, which is always produced, shows the contents of P and B0 as parcel addresses.

UNIT=*unit* A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. The default is $OUT.

AF=*fmt* A register format in decimal; the default is (8(3XO8))

BF=*fmt* B register format in decimal; the default is (8(3XO8))

SF=*fmt* S register format in decimal; the default is (4O25)

TF=*fmt* T register format in decimal; the default is (4O25)

VF=*fmt* V register format in decimal; the default is (4O25)

VL=*n* Number of V register elements to be snapped. The default is VL=VL. The caller can also specify VL=VL+1 or an absolute expression. If VL is 0 or 64, then VL=VL+1 means 64 rather than 65. The default radix of *n* is decimal unless a BASE O or BASE M is in effect.

RETURN CONDITIONS:
All registers are saved, including the vector registers and VL.

DUMP - DUMP SELECTED AREAS OF MEMORY

The DUMP macro performs a formatted dump of selected memory areas.

The macro generates exactly three words of inline code; the rest of the logic is in a unique subroutine created by the macro.

The DEBUG option allows conditional execution of the DUMP macro. If the label on the DUMP statement is DEBUG, no label is defined for the generated code. Instead, unless the symbol DEBUG has been set to 1 by a previously assembled SET or equate statement, code generation within the macro is suppressed entirely.

Format:

Location	Result	Operand
	DUMP	(*list*),UNIT=*unit*

list A list of memory ranges separated by commas. The list need not be enclosed in parentheses if it contains only one range. There is no limit to the number of ranges in the list. Within the list, null elements are ignored, so that each memory range can be preceded and followed by blanks. However, a memory range cannot contain embedded blanks. Each non-null range must have one of the following forms:

f..*l*	Dump memory from address *f* to address *l*-1
f	Dump memory word *f*
f(n)	Dump *n* words starting at memory address *f*

f, *l*, or *n* can be numbers, labels, register names, or a combination of labels and numbers. Indirect addressing, using the at sign (@) as a prefix, is allowed. For numbers, the default radix is decimal unless a BASE O or BASE M is in effect.

Examples:

(O'200..O'400)	Words 200_8 through 377_8
(0(D'128))	Words 0 through 177_8 (the Job Communication Block)
(R.A1(R.A2))	The starting address is given in A1 and the word count is given in A2
(@R.A1(@R.A2))	The starting address is given in the memory word addressed by A1 and the word count is in the memory word addressed by A2

(R.A1..R.A2) The address given in A1 through the address immediately before the address given in A2

(@R.A1..@R.A2) The address given in the memory word addressed by A1 through the address immediately before the address given in the memory word addressed by A2

(TABLE(R.A.BU)) The first *n* words of TABLE, where *n* is held in register A.BU

(TTT-1(@R.B77)) The first *n* words following and including TTT-1, where *n* is held in the memory addressed by register B77

(@PTR(@LTH)) The word addressed by PTR is the start, and the word count is in the word addressed by LTH

(@P..@Q,@A(@L)) Two ranges are dumped. The first range is from the word addressed by P through the word immediately before the word addressed by Q; the second begins at the word addressed by A and includes the number of words given by the value contained in the memory cell addressed by L. Only the low-order 24 bits in P, Q, A, and L are considered in determining the addresses.

UNIT=*unit* A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. The default is $OUT.

RETURN CONDITIONS: All registers are saved and restored, including the vector registers and VL.

INPUT - READ DATA

The INPUT macro reads data resident on a dataset or characters already located in memory and assigns values to variables, words of an array, or registers. Its syntax is as close as possible to the syntax of the INPUT statement in SKOL.

The macro generates its code either inline or in a unique subroutine created by the macro. In the latter case, exactly three words of code are generated inline.

The DEBUG option allows conditional execution of the INPUT macro. If the label on the INPUT statement is DEBUG, no label is defined for the generated code. Instead, unless the symbol DEBUG has been set to 1 by a previously assembled SET or equate statement, code generation within the macro is suppressed entirely.

Format:

Location	Result	Operand

	INPUT	(*list*),SV=$\left\{\begin{matrix}\text{YES}\\ \text{NO}\end{matrix}\right\}$,IN=$\left\{\begin{matrix}\text{YES}\\ \text{NO}\end{matrix}\right\}$,UNIT=*unit*,
		STRING=*addr*,LTH=*length*,END=*addr*,
		ERR=*addr*

list A list of input elements, each of which may include a variable name, an array specifier, and a format item. The list need not be enclosed in parentheses if it contains only one element. If it consists of more than one element, the elements are separated by commas. Null elements are ignored, so that each list element may be preceded and followed by blanks. However, an element cannot contain embedded blanks. Each element non-null must have one of the following forms:

:fmt A format item not associated with a variable, such as :2x or :/

var:fmt A variable name and the format used to read a value into it

The format can contain any of the edit descriptors available to the CRAY-1 FORTRAN (CFT) user. The format cannot contain commas unless the entire list item is enclosed in parentheses.

The variable can refer to a single word, to an array, to a single register, or to an array of registers, and can take any of the following forms:

addr Change the contents of a single word (for example, LABEL-2 or W.177

addr(*count*)
Read values for *count* words beginning at *addr*

addr(*count*!*incr*)
: Read values for *count* words beginning at *addr* and applying an increment of *incr* after each word. The default value for *incr* is 1.

R.*rn* Change the contents of register *rn* (where *r* is A, B, S, or T and *n* is an octal register number or a register designator of the form .*name*.)

R.VL or R.VM
: Change the current vector length or vector mask

R.*rn*(*count*)
: Change *count* registers starting with *rn*, as in R.A1(5)

R.V*n*(*count*)
: Change the first *count* elements of V*n*

R.V*n*+*e* Change the *e*th element in V*n*

R.V*n*+*e*(*count*)
: Change *count* elements, beginning at the *e*th element in V*n*

In all of the above, *n* must be either an octal number or a previously defined register designator. *count* and *e* may be represented by any absolute expression, in which the default radix is determined by the calling program.

The variable can also refer indirectly to a word or to an array, using a saved register or a word in memory as a pointer. The forms begin with the at sign (@) and include:

@*addr* Modify the word addressed by *addr*

@*addr*(*count*)
: Modify *count* words beginning with the word addressed by *addr*

@*addr*(*count*!*incr*)
: Modify *count* words beginning with the word addressed by *addr*, applying an increment of *incr* after each word

@R.*rn* Modify the word addressed by register *rn*

@R.*rn*(*count*)
: Modify *count* words beginning with the word addressed by register *rn*

SV=$\begin{Bmatrix}\text{YES}\\\text{NO}\end{Bmatrix}$ Save flag. If SV=NO, the SAVEREGS and LOADREGS macros are not invoked, and registers cannot be used for input values; IN=YES must also be specified when SV=NO. The default is SV=YES, which saves and restores all registers.

IN=$\begin{Bmatrix}\text{YES}\\\text{NO}\end{Bmatrix}$ Inline code flag. If IN=YES, all the code necessary to perform the INPUT (except the standard subroutines called by the SAVEREGS and LOADREGS macros) is generated inline. The default is IN=NO, which causes 3 words of code to be generated inline; the rest is contained in a subroutine created by the macro.

UNIT=*unit* A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. The default is $IN.

STRING=*string*
Address of a packed character string that resides in memory. When used in conjunction with the LTH parameter, the STRING parameter allows input (decoding) from the string. The END and ERR parameters cannot be used with STRING and LTH.

LTH=*length*
Number of characters to be decoded from *string*

END=*addr* Optional address to which a branch occurs if an end-of-file is encountered

ERR=*addr* Optional address to which a branch occurs if an error is encountered during the read

RETURN CONDITIONS: All registers, including the vector registers and the vector length register, are saved and restored when SV=YES (the default).

OUTPUT - WRITE DATA

The OUTPUT macro transfers variable values and character strings from a user's data area to a dataset or to an area in memory. Its syntax is as close as possible to the syntax of the OUTPUT statement in SKOL.

The macro generates its code either inline or in a unique subroutine created by the macro. In the latter case, exactly three words of code are generated inline.

The DEBUG option allows conditional execution of the OUTPUT macro. If the label on the OUTPUT statement is DEBUG, no label is defined for the generated code. Instead, unless the symbol DEBUG has been set to 1 by a previously assembled SET or equate statement, code generation within the macro is suppressed entirely.

Format:

Location	Result	Operand
	OUTPUT	(*list*),SV=$\begin{Bmatrix}\text{YES}\\\text{NO}\end{Bmatrix}$,IN=$\begin{Bmatrix}\text{YES}\\\text{NO}\end{Bmatrix}$,UNIT=*unit*, BUFFER=*addr*,LTH=*length*

list A list of variable names, array names, format items, and string constants separated by commas. The list need not be enclosed in parentheses if it contains only one element. If it consists of more than one element, the elements are separated by commas. Null elements are ignored, so that each element can be preceded and followed by blanks. However, an element cannot contain embedded blanks unless it is enclosed in a second level of parentheses. Each non-null element must have one of the following forms:

'*string*' or **string**
Represents any character string. The list item must be enclosed in parentheses if the string contains any blanks or commas. If the string is delimited by apostrophes, any inner apostrophes must be doubled. If it is delimited by asterisks, no inner asterisks are allowed.

:*fmt* Represents a format item that is not associated with any variable (for example, :2X, :/, or ::). The list item must be enclosed in parentheses if *fmt* contains any commas or blanks.

$PAGE, $SKIP, and $LINE
These special format items do not require a colon prefix. They generate FORTRAN-style carriage control characters at the beginning of a line. When $SKIP or $PAGE is the first list element, the appropriate literal character (0 or 1) becomes the first element of the OUTPUT format. $LINE is assumed to be present by default unless the first list element is a format item (:*fmt*). If $LINE, $SKIP, or $PAGE occurs later in the list, a comma and a slash are inserted before the carriage control literal in order to force a new line.

var:fmt Represents a variable name and the format to be used for its output

var::fmt The same as *var:fmt*, except that the variable's name and value are output together

var The same as *var*::O22

var(...) The same as *var*(...)::(4025)

The variable can refer to a single word, to an array, to a single register, or to an array of registers and can take any of the following forms:

addr Write the contents of a single word (for example, LABEL-2 or W.177)

addr(*count*)
Write *count* words beginning at *addr*

addr(*count*)
Write *count* words beginning at *addr*

addr(*count*!*incr*)
Write *count* words beginning at *addr* and applying an increment of *incr* after each word. The default value for *incr* is 1.

R.*rn* Write the contents of register *rn* (where *r* is A, B, S, or T and *n* is an octal register number or a register designator of the form .*name*)

R.VL or R.VM
Write the current vector length or vector mask

R.*rn*(*count*)
Write *count* registers starting with *rn*, as in R.A1(5)

R.V*n*(*count*)
Write the first *count* elements of V*n*

R.V*n*+*e* Write the *e*th element in V*n*

R.V*n*+*e*(*count*)
Write *count* elements, beginning at the *e*th element in V*n*

In all of the above, *n* must be either an octal number or a previously defined register designator. *count* and *e* may be represented by any absolute expression.

The variable can also refer indirectly to a word or to an array, using a saved register or a word in memory as a pointer. The forms begin with the at sign (@) and include:

@*addr* Write the word addressed by *addr*

@*addr*(*count*)
Write *count* words beginning with the word addressed by *addr*

@*addr*(*count*!*incr*)
Write *count* words beginning with the word addressed by *addr*, applying an increment of *incr* after each word

@R.*rn* Write the word addressed by register *rn*

@R.*rn*(*count*)
Write *count* words beginning with the word addressed by register *rn*

$\text{SV=}\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$ Save flag. If SV=NO, the SAVEREGS and LOADREGS macros are not invoked, and registers cannot be used for output; IN=YES must also be specified if SV=NO. The default is SV=YES, which saves and restores all registers.

$\text{IN=}\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$ Inline code flag. If IN=YES, all the code necessary to perform the OUTPUT (except the standard subroutines called by the SAVEREGS and LOADREGS macros) is generated inline. The default is IN=NO, which means that exactly three words of code are generated inline; the rest is contained in a subroutine created by the macro.

UNIT=*unit*

A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. The default is $OUT.

UNIT=$LOG is treated as a special parameter value rather than as a dataset name. If UNIT=$LOG, the OUTPUT macro automatically encodes the data (using its own buffer) rather than writing it directly, and uses the MESSAGE macro to write it to both the user log and the system log.

OUTPUT looks at the first 8 characters of the formatted line. The content of the first 8 characters of a message ID is:

▨	▨	▨	▨	▨	Δ	-	Δ

where is a space. If the first 8 characters do not match the above, OUTPUT inserts the string:

Δ	Δ	Δ	Δ	Δ	Δ	-	Δ

BUFFER=*addr*

Address of a packed character buffer used instead of an external dataset to accept the output

LTH=*length*

Number of characters to be encoded (output) into the buffer

RETURN CONDITIONS: All registers, including the vector registers and the vector length register, are saved and restored when SV=YES (the default).

FREAD - READ DATA

The FREAD macro permits a FORTRAN-like read statement that can make use of a previously defined format.

Format:

Location	Result	Operand
	FREAD	*fmt*,(*list*),SV=$\begin{Bmatrix}\text{YES}\\\text{NO}\end{Bmatrix}$,UNIT=*unit*,END=*addr*, ERR=*addr*

fmt Format; takes one of the following forms:

addr Address of a format, possibly defined with the DATA pseudo instruction, as in:

```
          DATA     '(F10.0)'
```

((*string*))

A character string enclosed in a double set of parentheses

The default is (5025).

(*list*) List of addresses for which values are to be read. Even if there is only one item, the list must be enclosed in parentheses. Each item in the list specifies either the address of a single word or the address of an array.

An array is handled by enclosing the array base address, the word count, and an optional increment in an additional set of parentheses. Examples: ((A,10)) or ((B,LTH,3))

The CAL statement

```
       FREAD    ,((A,10),(B,LTH,3))
```

is equivalent to the FORTRAN statements

```
       READ 20, (A(I), I=1,10), (B(3*(I-1)+1), I=1, LTH)
    20 FORMAT (5025)
```

An array or a single word described as a one-word array can be addressed indirectly by using the at sign (@) and the name of a variable containing the indirect address instead of an array name. For example:

((@C,10))	Reads values for the first 10 words of an array beginning at an address held in variable C
((@E,1))	Reads a value for the single word specified by the address held in variable E

To pass a numeric address, use a W prefix (for example, W.177).

SV=$\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$ Save flag. If SV=NO, the SAVEREGS and LOADREGS macros are not invoked. If SV=YES, all registers are saved and restored. The default is SV=NO.

UNIT=*unit*
A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. The default is $IN.

END=*addr* Optional address to which a branch occurs if an end-of-file is encountered

ERR=*addr* Optional address to which a branch occurs if an error is encountered during the read

FWRITE - WRITE DATA

The FWRITE macro permits a FORTRAN-like write statement that can make use of a previously defined format.

Format:

Location	Result	Operand
	FWRITE	*fmt*,(*list*),SV=$\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$,UNIT=*unit*

fmt Format; takes one of the following forms:

addr Address of a format, possibly defined with the DATA pseudo instruction, as in:

```
          DATA     '(F10.0,''TEXT'')'
```

((*string*))

A character string enclosed in a double set of parentheses (for example, ((F10.0,''TEXT'')))

The default is (5O25).

(*list*) List of addresses whose contents are to be written. Even if there is only one item, the list must be enclosed in parentheses. Each item in the list specifies either the address of a single word or the address of an array.

An array is handled by enclosing the array base address, the word count, and an optional increment in an additional set of parentheses. Examples: ((A,10)) or ((B,LTH,3))

The CAL statement

```
       FWRITE   ,((A,10),(B,LTH,3))
```

is equivalent to the FORTRAN statements

```
       PRINT 20, (A(I), I=1,10), (B(3*(I-1)+1), I=1, LTH)
   20  FORMAT (5O25)
```

An array or a single word described as a one-word array can be addressed indirectly by using the at sign (@) and the name of a variable containing the indirect address instead of an array name. For example:

((@C,10))	Reads values for the first 10 words of an array beginning at an address held in variable C
((@E,1))	Reads a value for the single word specified by the address held in variable E

To pass a numeric address, use a W prefix (for example, W.177).

$\text{SV=}\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$ Save flag. If SV=NO, the SAVEREGS and LOADREGS macros are not invoked. If SV=YES, all registers are saved and restored. The default is SV=NO.

UNIT=*unit*

A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. The default is $IN.

UFREAD - UNFORMATTED READ

The UFREAD macro performs a FORTRAN-like unformatted read.

Format:

Location	Result	Operand
	UFREAD	*unit*,(*list*),SV=$\begin{Bmatrix}\text{YES}\\\text{NO}\end{Bmatrix}$,END=*addr* ERR=*addr*

unit A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. There is no default.

(*list*) List of addresses for which values are read. Even if there is only one item, the list must be enclosed in parentheses. Each item in the list specifies either the address of a single word or the address of an array.

An array is handled by enclosing the array base address, the word count, and an optional increment in an additional set of parentheses. Examples: ((A,10)) or ((B,LTH,3))

The CAL statement

```
      UFREAD   ,((A,10),(B,LTH,3))
```

is equivalent to the FORTRAN statement

```
      READ (A(I), I=1,10), (B(3*(I-1)+1), I=1, LTH)
```

An array or a single word described as a one-word array can be addressed indirectly by using the at sign (@) and the name of a variable containing the indirect address instead of an array name. For example:

((@C,10)) Reads values for the first 10 words of an array beginning at an address held in variable C

((@E,1)) Reads a value for the single word specified by the address held in variable E

To pass a numeric address, use a W prefix (for example, W.177).

SV=$\left\{\begin{matrix}\text{YES}\\ \text{NO}\end{matrix}\right\}$ Save flag. If SV=NO, the SAVEREGS and LOADREGS macros are not invoked. If SV=YES, all registers are saved and restored. The default is SV=NO.

END=*addr* Optional address to which a branch occurs if an error is encountered during the read

ERR=*addr* Optional address to which a branch occurs if an error is encountered during the read

UFWRITE - UNFORMATTED WRITE

The UFWRITE macro performs a FORTRAN-like unformatted write of output items separated by commas.

Format:

Location	Result	Operand
	UFWRITE	*unit*,(*list*),SV=$\left\{\begin{matrix}\text{YES}\\ \text{NO}\end{matrix}\right\}$

unit A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. There is no default value.

(*list*) List of addresses whose contents are to be written. Even if there is only one item, the list must be enclosed in parentheses. Each item in the list specifies either the address of a single word or the address of an array.

An array is handled by enclosing the array base address, the word count, and an optional increment in an additional set of parentheses. Examples: ((A,10)) or ((B,LTH,3))

The CAL statement

```
      UFWRITE    $OUT,((A,10),(B,LTH,3))
```

is equivalent to the FORTRAN statement

```
      PRINT (A(I), I=1,10), (B(3*(I-1)+1), I=1, LTH)
```

An array or a single word described as a one-word array can be addressed indirectly by using the at sign (@) and the name of a variable containing the indirect address instead of an array name. For example:

((@C,10)) Writes the first 10 words of an array beginning at an address held in variable C

((@E,1)) Writes the single word specified by the address held in variable E

To pass a numeric address, use a W prefix (for example, W.177).

SV=$\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$ Save flag. If SV=NO, the SAVEREGS and LOADREGS macros are not invoked. If SV=YES, all registers are saved and restored. The default is SV=NO.

SAVEREGS - SAVE ALL REGISTERS

The SAVEREGS macro saves all of the A, B, S, T, V, VL, and VM registers. Additionally, it sets up words containing VL+1, P/4, parcel(P), B0/4, and parcel(B0) so that SNAP can handle the VL=VL+1 option and so that SNAP, DUMP, and OUTPUT can output P and B0 in parcel-address format. (Here, parcel(x) means the 2 low-order bits of x.)

Format:

Location	Result	Operand
	SAVEREGS	[*region*], INLINE=$\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$

region Label of the first word of a region where registers are to be saved. The default is QZH44HZQ. If the *region* is defined, it must be with a BSS O'1230; if it is not defined, the SAVEREGS macro defines it. If SAVEREGS requests are nested, each request must specify a different *region*.

INLINE= $\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$

Inline code flag. If INLINE is omitted (as when SAVEREGS is invoked directly), A0 is saved in word O'1200 and B0 is saved in word O'1000 of the region. If INLINE=YES, both A0 and B0 are lost. If INLINE=NO, B0 is saved in word O'1223 of the region and A0 is lost.

LOADREGS - RESTORE ALL REGISTERS

The LOADREGS macro restores the A, B, S, T, V, VL, and VM registers that were saved by a previously-executed SAVEREGS macro.

Format:

Location	Result	Operand
	LOADREGS	[*region*],INLINE= $\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$

region The *region* used previously in a corresponding SAVEREGS. If no value is specified, the default is QZH44HZQ. If the region is defined, it must be with a BSS O'1230; if it is not defined, the LOADREGS macro defines it. If LOADREGS requests are nested, each request must specify a different *region*.

INLINE= $\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$

Inline code flag. If INLINE is omitted (as when LOADREGS is called by a user), A0 and B0 are restored from words O'1200 and O'1000 of the region. If INLINE=YES, both A0 and B0 are lost. If INLINE=NO, B0 is restored from O'1223, but A0 is lost.

MISCELLANEOUS

Macros that do not fit in the other categories are the SYSID, GETMODE, GETSWS, and INSFUN macros.

SYSID - REQUEST SYSTEM IDENTIFICATION

The identification of the current system is returned at the location specified in the macro call. The identification is returned as two words; the first contains the COS revision level in ASCII and the second contains the COS assembly date in ASCII.

Format:

Location	Result	Operand
	SYSID	*address*

address A symbol or an A, S, or T register (not A0 or S0) containing the address where the system ID is returned

GETMODE - GET MODE SETTING

The GETMODE macro obtains the mode setting from the user's exchange package and returns it in the S1 register.

Format:

Location	Result	Operand
	GETMODE	

GETSWS - GET SWITCH SETTING

The GETSWS macro allows the user to determine whether a specified sense switch number is set or not. GETSWS returns the setting of the switch number specified in the S1 register. S1=1 if set; 0 if not set.

Format:

Location	Result	Operand
	GETSWS	*n*

n Number of the switch (1-6) to be tested

INSFUN - CALL INSTALLATION-DEFINED SUBFUNCTION

The INSFUN macro allows the user to call any one of the installation-defined subfunctions defined in a subfunction table (INSTAB). Control is transferred to the indicated subfunction.

Format:

Location	Result	Operand
	INSFUN	*n,p*

n A symbol or an A, S, or T register (not A0 or S0) containing the subfunction code

p An optional symbol, A, S or T register (not S2), containing the address of a parameter list to be passed to the installation-dependent subfunction.

LOGICAL I/O MACROS 3

The logical I/O macros generate calls to I/O subroutines to be loaded from the subroutine library and executed as part of the user program. The logical I/O macros apply only to blocked datasets. Datasets referenced by these macros must have been opened previously by an OPEN macro.

There are four main categories of logical I/O macros: synchronous read/write, asynchronous (buffered) read/write, unblocked read/write, and positioning.

SYNCHRONOUS READ/WRITE

The synchronous read/write logical I/O macros allow the user to read and write words or characters and to write an end-of-file or an end-of-data. Control does not return to the user program until all requested data transfers are completed.

Upon termination of the READ/WRITE function, register contents are modified as detailed under the description of each macro. A or S registers not specifically mentioned should not be assumed to have any meaningful contents, and will not contain the same values as before the function request. Registers B0, B70-B77, and T70-T77 may be changed, as well as VL, VM, V0, and V1. Other B, T, and V registers will not be changed.

Issuing a synchronous I/O macro for an unblocked dataset produces an error.

READ/READP - READ WORDS

The READ and READP macros transfer words of data that are resident on a dataset into the user's data area. Blank compression characters are not recognized, and the compressed blanks are not expanded (see part 1, section 2).

The READ macro generates a return jump to the $RWDR subroutine, thus causing one record at a time to be processed. Each macro call causes the dataset to be positioned after the end-of-record that terminated the read.

The READP macro generates a return jump to the $RWDP subroutine. Words are transmitted to the user's data area as requested by the user. Each call is terminated by reaching an end-of-record or by satisfying the word count, whichever comes first.

No blank decompression is performed.

When end-of-record is reached as a result of reading in word mode, the unused bit count from the end-of-record RCW is placed in the field DPBUBC of the Dataset Parameter Area (DSP). Also, the unused bits are zeroed in the user's record area.

Unrecovered data errors do not abort the job; instead control is returned to the caller. The caller can use the good data read, (A2) through (A4)-1, and then abort. The caller can also skip or accept the bad data.[§] If the caller does nothing, the job aborts when the next read request occurs. See the Library Reference Manual, CRI publication SR-0014, for detailed descriptions of SKIPBAD and ACPTBAD.

When a READ or READP macro refers to a memory resident dataset, the first such macro causes the dataset to be loaded into the buffer from mass storage, if it exists there. If it does not exist on mass storage, the system I/O routines set the DSP so that it appears that the buffer is filled with data and no attempt is made to read data. Note that the I/O routines cannot distinguish between the cases (1) an existing dataset is declared memory resident, read in, modified in the buffer, rewound, and read again, and (2) no modification of data in the buffer occurs. In either case, the first read following a REWIND reads the unmodified data from disk. If an existing dataset is declared memory resident and is to be modified and reread, use backspace positioning macros rather than REWIND to reposition to beginning-of-data to preserve the modifications. This is necessary only when a memory resident dataset already exists on mass storage.

Formats:

Location	Result	Operand
	READ	*dn,uda,ct*

Location	Result	Operand
	READP	*dn,uda,ct*

[§] Deferred implementation

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Area (DSP) address or negative DSP offset relative to JCDSP

uda User data area first word address (FWA) or an A, B, or S register (not A1) containing the *uda* address

ct Word count or an A, B, or S register (not A1 or A2) containing the number of words to be read

RETURN CONDITIONS:

(A1) DSP address

(A2) FWA of user data area (*uda*)

(A3) Requested word count (*ct*)

(A4) Actual LWA+1 of data transferred to *uda*. A4=A2 if a null record was read.

(S0) Condition of termination

- < 0 End of record encountered
- = 0 Null record, end of file, end of data, or unrecovered data error encountered§
- > 0 User-specified count (A3) exhausted before end-of-record RCW is encountered

(S1) Error status§

- = 0 No errors encountered
- = 1 Unrecovered data error encountered

(S6) Contents of the RCW if S0≤0 and S1=0; otherwise meaningless. Note that for READ/READP, the unused bit count may also be obtained from S6 if S0<0. Unused bits are not meaningful for READC/READCP, since the unused characters will be reflected in the number of characters transferred (A4-A2).

§ Deferred implementation

READC/READCP - READ CHARACTERS

The READC and READCP macros transfer character data from a dataset into the user data area.

The READC macro generates a return jump to the $RCHR subroutine, thus causing one record at a time to be processed. Each macro call causes the dataset to be positioned after the end-of-record that terminated the read.

The READCP macro generates a return jump to the $RCHP subroutine. Characters are transferred to the user data area as requested by the user. Each call is terminated by reaching an end-of-record or by satisfying the character count, whichever occurs first.

One character from the record is placed, right-adjusted, zero-filled, in each word of the data area. Blank-compressed fields are recognized and expanded, one blank per word.

Unrecovered data errors do not abort the job if the dataset is tape resident; instead control is returned to the caller. The caller can use the good data read, (A2) through (A4)-1, and then abort. The user can also skip or accept the bad data.§ If the caller does nothing, the job aborts when the next read request occurs. See Library Reference Manual, CRI publication SR-0014, for detailed descriptions of SKIPBAD and ACPTBAD.

Memory resident datasets are treated as described for READ/READP macro.

Formats:

Location	Result	Operand
	READC	*dn,uda,ct*

Location	Result	Operand
	READCP	*dn,uda,ct*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Area (DSP) address or negative DSP offset

uda User data area first word address (FWA) or an A, B, or S register (not A1) containing the *uda* address

§ Deferred implementation

ct Character count or an A, B, or S register (not A1 or A2) containing the character count

RETURN CONDITIONS: Same as for READ/READP, except that the requested count (A3) and data-transfer length (A4-A2) is in characters rather than words

WRITE/WRITEP - WRITE WORDS

The WRITE macro generates a return jump to either the $WWDR or $WWDS subroutine, depending on whether an unused bit count is specified. Words are written from the user's data area. An end-of-record RCW is written following each WRITE. The end-of-record RCW indicates how many bits in the last words are unused, if any.

The WRITEP macro generates a return jump to the $WWDP subroutine. Words are written from the user's data area as requested by the user. No end-of-record is written.
No blank compression is performed.

If the dataset is memory resident and the WRITE or WRITEP causes the buffer to become full, the memory resident flags are cleared and the buffers are flushed to mass storage.

To write only an end-of-record RCW, the WRITE macro with word count of 0 is used.

Formats:

Location	Result	Operand
	WRITE WRITE	*dn,uda,ct,ubc* *dn,uda,ct*

Location	Result	Operand
	WRITEP	*dn,uda,ct*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Area (DSP) address or negative DSP offset

uda User data area first word address (FWA) or an A, B, or S register (not A1) containing the *uda* address

ct Word count or an A, B, or S register (not A1 or A2) containing the word count

ubc Unused bit count or an A, B, or S register (not A1, A2, or A3) containing the unused bit count or null. If null, record contains no unused bits.

RETURN CONDITIONS: (A1) DSP address

(A2) FWA of user data area (*uda*)

(A3) Requested word count (*ct*)

WRITEC/WRITECP - WRITE CHARACTERS

The WRITEC and WRITECP macros transfer characters from the user's data area to the dataset.

The WRITEC macro generates a return jump to the $WCHR subroutine, thus causing one record at a time to be processed. An end-of-record RCW is written following each WRITEC.

The WRITECP macro generates a return jump to the $WCHP subroutine. Characters are written from the user's data area as requested by the user. No end-of-record is written.

One character is taken from bits 56-63 of each word of the data area and packed into the record, eight characters per word. Blank compression (see part 1, section 2) occurs.

Memory resident datasets are handled as described for WRITE/WRITEP.

To write only an end-of-record RCW, the WRITEC macro with character count of 0 is used.

Formats:

Location	Result	Operand
	WRITEC	*dn,uda,ct*

Location	Result	Operand
	WRITECP	*dn,uda,ct*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Area (DSP) address or negative DSP offset

uda User data area first word address (FWA) or an A, B, or S register (not A1) containing the *uda* addresss

ct Character count or an A, B, or S register (not A1 or A2) containing the character count

RETURN CONDITIONS: Same as for WRITE/WRITEP except that the requested count (A3) is in characters rather than words

WRITEF - WRITE END OF FILE

The WRITEF macro generates a return jump to the $WEOF subroutine, causing an end-of-record RCW (if not previously written) and an end-of-file RCW to be written.

If the WRITEF macro causes the buffer for a memory resident dataset to be full, the memory resident flags are cleared and the buffers are flushed to mass storage.

Format:

Location	Result	Operand
	WRITEF	*dn*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Area (DSP) address or negative DSP offset

RETURN CONDITIONS: (A1) DSP address

WRITED - WRITE END OF DATA

The WRITED macro generates a return jump to the $WEOD subroutine, causing an end-of-record RCW (if not previously written), an end-of-file RCW (if not previously written), and an end-of-data RCW to be written.

The WRITED macro causes buffers to be flushed. If the dataset is memory resident, buffers are flushed to mass storage only if the end of data occurs within the last block of the buffer; in this case the memory resident flags will also be cleared.

Format:

Location	Result	Operand
	WRITED	*dn*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Area (DSP) address or negative DSP offset

RETURN CONDITIONS: (A1) DSP address

ASYNCHRONOUS READ/WRITE

The asynchronous read/write logical I/O macros allow the user to read and write words and to write an end of file or an end of data. These macros provide the CRAY Assembly Language (CAL) programmer with the same capabilities as the FORTRAN BUFFER IN/BUFFER OUT statements.

Control returns to the user immediately. It is the user's responsibility to ensure that requested data transfers are complete and error-free by examining the DSP before attempting to process input data or requesting additional writes. The macro BUFCHECK is provided to make the necessary checks.

All of the asynchronous blocked I/O macros use registers A0, A1, A2, S0, S1, and S2. Other A and S registers, and all B, T, and V registers remain unchanged (except B0). Unblocked I/O processing also uses registers A6, S3, and S4. In all cases, after the I/O function completes, A1 contains the DSP address. The other registers used are not meaningful. All status responses must be obtained from the DSP.

Asynchronous requests for unblocked datasets require that the *uda* parameter specify the address of an area in the user's program. Also, the *ct* parameter must be a value that is a multiple of 512.

Memory resident datasets are handled the same as for the synchronous read/write macros. See the description of the READ, WRITE, WRITEF, and WRITED macros for the handling of BUFIN(P), BUFOUT(P), BUFEOF, and BUFEOD respectively.

BUFIN/BUFINP - TRANSFER DATA FROM DATASET TO USER RECORD AREA

The BUFIN and BUFINP macros transfer words of data from a dataset to a user record area. Both macros generate a system call to $CBIO.

The BUFIN macro transfers data from the current position to end-of-record or until the specified word count is exhausted. The dataset is positioned after the end of the current record. Field DPBUBC indicates the count of unused bits in the last word of the record. If the word count is exhausted before end of record, the unused bit count is set to zero.

The BUFINP macro transfers data from the current position to end-of-record or until the specified word count is exhausted. The dataset remains positioned midrecord if the word count is exhausted before end-of-record is reached. The unused bit count is set in the same way as for BUFIN.

In both cases, control returns to the user program immediately, giving the user the responsibility of monitoring the proper DSP fields to determine when the transfer is complete and whether any errors occurred.

If the dataset is unblocked, the specified word count is transferred. RCWs and BCWs are ignored.

Formats:

Location	Result	Operand
	BUFIN	*dn,uda,ct,rcl*

Location	Result	Operand
	BUFINP	*dn,uda,ct,rcl*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset). It is the Dataset Parameter Area (DSP) address if and only if *dn*=A1. The ODN address may be given in any other A register except A0, any S register except S0, S1 or S2, or any B register.

uda User's record area or A, S, or T register (not A0, S0, S1, or S2) containing the *uda* address

ct Word count or A, S, or T register containing word count (not S1 or S2)

rcl Optional recall flag. If not null, the macro expansion contains a RECALL loop until the I/O is completed.

Registers S1 and S2 are used to construct the parameter word (W@DPBIO) and may not contain parameter address or values.

When I/O is completed, for both BUFIN and BUFINP, the actual number of words transferred can be obtained from the DPBWC field of the DSP.

BUFOUT/BUFOUTP - TRANSFER DATA FROM USER RECORD AREA TO DATASET

The BUFOUT and BUFOUTP macros transfer data from a user's record area to a dataset using the system F$BIO function.

The BUFOUT macro transfers the specified number of words and writes an end-of-record RCW on the dataset. Optionally, an unused bit count may be specified, giving the number of bits in the last word of data that are not to be considered as part of the data. The end-of-record RCW will contain this unused bit count.

The BUFOUTP macro transfers the specified number of words but does not write an end-of-record RCW. Subsequent BUFOUTP macro calls continue to construct the record. A subsequent BUFOUT macro terminates the record with an end-of-record. Unused bits are meaningless for BUFOUTP.

In both cases, control returns to the user program immediately, giving the user the responsibility of monitoring the proper DSP fields to determine when the transfer is complete and whether any errors occurred.

If the dataset is unblocked, the specified word count is transferred. RCWs and BCWs are ignored.

Formats:

Location	Result	Operand
	BUFOUT	*dn,uda,ct,ubc,rcl*

Location	Result	Operand
	BUFOUTP	*dn,uda,ct,ubc,rcl*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset). It is the Dataset Parameter Area (DSP) address if and only if *dn*=A1. The ODN address may be given in any other A register except A0, any S register except S0, S1, or S2, or any B register.

uda User record area or A, S, or T register containing record area address (not A0, S0, S1, or S2)

ct Word count or A, S, or T register containing word count (not A0, S0, S1 or S2)

ubc Optional unused bit count or A, S, or T register containing unused bit count (not A0, S0, or S2) or null. If null, record contains no unused bits. This field is ignored for BUFOUTP.

rcl Optional recall flag. If not null, the macro expansion contains a RECALL loop until the I/O is completed.

Registers S1 and S2 are used to construct the parameter word (W@DPBIO) and may not contain parameter addresses or values, except that S1 may contain the unused bit count.

BUFEOF - WRITE END OF FILE ON DATASET

The BUFEOF macro is used to write an end-of-file on a dataset. Control returns immediately to the user program, giving the user the responsibility of monitoring the DPBIO field. An end-of-record is written if the dataset is mid-record.

Issuing a BUFEOF macro for an unblocked dataset produces an error.

Format:

Location	Result	Operand
	BUFEOF	*dn,rcl*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset). It is the Dataset Parameter Area (DSP) address if and only if *dn*=A1. The ODN address may be given in any other A register except A0, any S register except S0, S1, or S2, or any B register.

rcl Optional recall flag. If not null, the macro expansion includes a RECALL loop until the I/O is completed.

BUFEOD - WRITE END OF DATA ON DATASET

The BUFEOD macro writes an end-of-data to a dataset. Control returns immediately to the user and it is the user's responsibility to monitor the DPBIO field. An end-of-record and an end-of-file will also be written, if necessary.

Issuing a BUFEOD macro for an unblocked dataset produces an error.

Format:

Location	Result	Operand
	BUFEOD	*dn,rcl*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset). It is the Dataset Parameter Area (DSP) address if and only if *dn*=A1. The ODN address may be given in any other A register except A0, any S register except S0, S1, or S2, or any B register.

rcl Optional recall flag. If not null, the macro expansion includes a RECALL loop until the I/O is completed.

BUFCHECK - CHECK BUFFERED I/O COMPLETION

The BUFCHECK macro requests the system to wait until the buffered I/O on a dataset has completed and, optionally, to go to an error address if the DSP status contains any error flags when the I/O completes.

Format:

Location	Result	Operand
	BUFCHECK	*dn,err*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset). It is the Dataset Parameter Area (DSP) address if and only if *dn*=A1. The ODN address may be given in any other A register except A0, any S register except S0, S1, or S2, or any B register.

err Optional error address. If any error bits are set in the DSP on completion of the I/O, control is transferred to *err*, if specified. If *err* is not specified, it is the user's responsibility to detect any errors. Note that for this purpose, DPEOI does not constitute an error bit.

RETURN CONDITIONS: If *err* is specified, S1 will contain the DSP field DPERR, right-justified.

UNBLOCKED READ/WRITE

The unblocked dataset read and write macros allow the user to read and write data directly into or from a buffer supplied by a program rather than by the system. The system waits for I/O to complete.

The system does no blocking or deblocking of unblocked datasets.

Upon termination of the READ/WRITE function, register contents are modified as detailed under the description of each macro. A or S registers not specifically mentioned should not be assumed to have any meaningful contents, and will not contain the same values as before the function request. Registers B0, B70-B77, and T70-T77 may be changed, as well as VL, VM, V0, and V1. Other B, T, and V registers will not be changed.

READU - TRANSFER DATA FROM DATASET TO USER'S AREA

The READU macro transfers words of data from an unblocked dataset into an area specified by the caller. The READU macro generates a return jump to the $RLB subroutine.

Format:

Location	Result	Operand
	READU	*dn,uda,ct*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register (except A0 or S0) containing the Dataset Parameter Area (DSP) address or negative DSP offset

uda User data area first word address (FWA) or an A, B, or S register (except A0, A1, or S0) containing the *uda* address

ct Word count or an A, B, or S register (except A0, A1, A2, or S0) containing the number of words to be transferred. *ct* must be a multiple of 512.

RETURN CONDITIONS:

(A1) DSP address

(A2) FWA of user data area (*uda*)

(A3) Requested word count (*ct*)

(A4) Actual LWA+1 of data transferred

(S0) Completion status. One of the following:

-1.0	Operation complete, no errors
0.0	Attempt to read past allocated data
+1.0	Parity error§
+2.0	Unrecovered hardware error§

§ Deferred implementation

WRITEU - TRANSFER DATA FROM USER'S AREA TO DATASET

The WRITEU macro transfers data from the user's area to an unblocked dataset. The WRITEU macro generates a return jump to the $WLB subroutine.

Format:

Location	Result	Operand
	WRITEU	*dn,uda,ct*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Area (DSP) address or negative DSP offset

uda User data area first word address (FWA) or an A, B, or S register (not A0, A1, or S0) containing the *uda* address

ct Word count or an A, B, or S register (not A1 or A2) containing the number of words to be transferred. *ct* must be a multiple of 512.

RETURN CONDITIONS: (A1) DSP address

(A2) FWA of user data area (*uda*)

(A3) Requested word count (*ct*)

(S0) Completion status. One of the following:

-1.0	Operation complete, no errors
0.0	Attempt to read past allocated data
+1.0	Parity error§
+2.0	Unrecovered hardware error§

POSITIONING

The user can rewind datasets, backspace records or files, get the current dataset position, and position datasets using the positioning logical I/O macros. See each macro description for register contents on return. Other registers mentioned as used by READ/WRITE will be meaningless on return.

§ Deferred implementation

When a dataset is positioned backward and the last operation on the dataset was a write operation, an end-of-data is written (and an end-of-record and end-of-file, if necessary). (See the WRITE, WRITEF, and WRITED macro descriptions for handling of memory resident datasets during the end-of-data processing.) If the last operation was not a write operation, backward positioning has no special effect on a dataset.

REWIND - REWIND DATASET

The REWIND macro generates a return jump to the $REWD subroutine causing the dataset to be positioned at beginning-of-data.

The REWIND macro causes all buffer pointers in the DSP to be reset to indicate an empty buffer. For memory resident datasets, the next read will cause the pointers to be reset. If the memory resident dataset previously existed on mass storage, any changes made to the contents of the buffer prior to the rewind will be lost. This is because the disk copy of the dataset is reread without the changes being flushed. If the dataset did not previously exist on disk, any changes in the buffer contents are preserved across the rewind and read sequence. To preserve changed buffer contents for a memory resident dataset that previously existed on disk, use BKSPF to reposition the dataset.

Format:

Location	Result	Operand
	REWIND	*dn*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Area (DSP) address or negative DSP offset

RETURN CONDITIONS: (A1) DSP address

BKSP - BACKSPACE RECORD

Th BKSP macro generates a return jump to the $BKSP subroutine. The dataset is backspaced one record. If the dataset is at beginning-of-data, no action occurs.

Because the backspace operation occurs within the buffer for memory resident datasets, such datasets receive special handling only if an end-of-data must be written. Changes made in the buffer contents are preserved.

Issuing a BKSP macro for an unblocked dataset produces an error.

BKSP applies to mass storage datasets only; it is illegal on tape datasets.

Format:

Location	Result	Operand
	BKSP	*dn*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register containing the Dataset Parameter Area (DSP) address or negative DSP offset

RETURN CONDITIONS: (A1) DSP address

(S6) The RCW after which the dataset was left-positioned; equals 0 if beginning-of-data is encountered.

BKSPF - BACKSPACE FILE

The BKSPF macro generates a return jump to the $BKSPF subroutine. The dataset is backspaced one file. If the dataset is at beginning-of-data, no action occurs.

Because the backspace operation occurs within the buffer for memory resident datasets, such datasets receive special handling only if an end-of-data must be written. Changes made in the buffer contents are preserved.

Issuing a BKSPF macro for an unblocked dataset produces an error. BKSPF applies to mass storage datasets only; it is illegal on tape datasets.

Format:

Location	Result	Operand
	BKSPF	*dn*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register containing the Dataset Parameter Area (DSP) address or negative DSP offset

RETURN CONDITIONS: Same as for BKSP

GETPOS - GET CURRENT DATASET POSITION

The GETPOS macro generates a return jump to the $GPOS subroutine. This subroutine returns the current dataset position in S1. The dataset position is the number of words between the beginning-of-data and the present position, not counting BCWs but including RCWs.

Format:

Location	Result	Operand
	GETPOS	*dn*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register (except A0 or S0) containing the Dataset Parameter Area (DSP) address or negative DSP offset

RETURN CONDITIONS: (A1) DSP address

(S1) For a blocked dataset, S1 contains dataset position flags. Bits 0-2 indicate position within records or files, bits 31-63 indicate physical word address within the file, including RCWs. At beginning-of-data, S1=0. Bit 0=1 if dataset is positioned immediately following a RCW. Bit 2=1 if the RCW is an end-of-file RCW. If bit 0=0, bit 2 also will equal zero, and the dataset is midrecord or at beginning-of-data. Bit 1 is unused.

For an unblocked dataset, S1 returns the relative position of the current block within the dataset.

(S2) For an unblocked dataset, S2 contains the same address as is contained in bits 31-63 of S1 for blocked datasets.

For a blocked dataset, S2 contains the physical word address relative to the beginning of the dataset, including RCWs.

SETPOS - POSITION DATASET

The SETPOS macro generates a return jump to the $SPOS subroutine. The dataset is positioned at the word indicated by the word offset specified, which must be at a record boundary (at beginning-of-data, or following end-of-record or end-of-file, or before end-of-data).

For an unblocked dataset, the DSP is updated to reflect the specified position within the dataset. No I/O request is actually issued. SETPOS applies to mass storage datasets only; it is illegal for tape datasets.

Format:

Location	Result	Operand
	SETPOS	*dn* ,*pos*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset) or an A, B, or S register containing the Dataset Parameter Area (DSP) address or negative DSP offset

pos Dataset position. May be any of the following:

EOD Position the dataset preceding end-of-data

BOD Position the dataset at beginning-of-data

S*n* or T*n*
Position the dataset to the word address contained in the specified S or T register. If *pos* is not S1, S1 is destroyed.

RETURN CONDITIONS: (S1) Dataset position (See GETPOS for meaning of flags)

(S6) Record control word after which dataset is positioned, or 0 at beginning-of-data

POSITION - POSITION TAPE

The POSITION macro generates an F$POS call to position a tape dataset.

Format:

Location	Result	Operand
	POSITION	*dn,subcode*

dn Symbolic address of the ODN table for this dataset or an A, S, or T register containing a pointer to the ODN table

subcode REWIND is the only *subcode* currently supported.

RETURN CONDITIONS: None

PERMANENT DATASET MACROS 4

The permanent dataset macro instructions are a subset of the system function requests. Each macro generates a function code that is a call to COS. The function code octal value is stored in register S0; S1 and S2 provide optional arguments. The function code is enabled when the program exit instruction is executed. (Note that the contents of the registers used are not restored after the call is completed.) See Appendix C for more information on system function codes.

The permanent dataset macro instructions are divided into two categories: those that define and those that manage permanent datasets.

PERMANENT DATASET DEFINITION

The PDD macro generates a parameter table containing information about the dataset. The ACCESS, SAVE, DELETE, ADJUST, DISPOSE, and SUBMIT macros involved in permanent dataset management use the PDD table. Thus, the PDD macro must accompany the use of the permanent dataset management macros.

The LDT macro generates a table containing information required to process labels for tape datasets. The LDT macro must accompany the PDD and ACCESS macros in a program accessing a labeled tape dataset if label processing is to occur.

PDD - CREATE PERMANENT DATASET DEFINITION TABLE

The PDD macro creates a parameter table called the Permanent Dataset Definition Table (PDD). (See Appendix A for a description of the PDD table.) This macro is non-executable and must accompany the use of the ACCESS, SAVE, DELETE, ADJUST, DISPOSE, or SUBMIT macros in a program.

Format:

Location	Result	Operand

pddtag	PDD	DN=*dn*,PDN=*pdn*,SDN=*sdn*,ID=*uid*,MF=*mf*,TID=*tid*,
		DF=*df*,DC=*dc*,SF=*sf*,RT=*rt*,ED=*ed*,RD=*rd*,WT=*wt*,
		MN=*mn*,DT=*dt*,CS=*cs*,LB=*lb*,LDT=*ldt*,
		NEW=$\begin{Bmatrix}\text{ON}\\\text{OFF}\end{Bmatrix}$,MSG=$\begin{Bmatrix}\text{ON}\\\text{OFF}\end{Bmatrix}$,UQ=$\begin{Bmatrix}\text{ON}\\\text{OFF}\end{Bmatrix}$,WAIT=$\begin{Bmatrix}\text{ON}\\\text{OFF}\end{Bmatrix}$,
		DEFER=$\begin{Bmatrix}\text{ON}\\\text{OFF}\end{Bmatrix}$,NRLS=$\begin{Bmatrix}\text{ON}\\\text{OFF}\end{Bmatrix}$,EXO=$\begin{Bmatrix}\text{ON}\\\text{OFF}\end{Bmatrix}$,SID=*mf*,DID=*mf*

pddtag Symbolic address of the PDD table

Parameters are in keyword form; the only required parameter is DN.

DN=*dn* Dataset name. DN is a required parameter.

PDN=*pdn* Permanent dataset name. The default value is *dn*.

SDN=*sdn* Staged dataset name; 1-15 alphanumeric character name by which the dataset will be known at the destination mainframe. The default is the local dataset name (DN).

ID=*uid* User identification; 1-8 alphanumeric characters assigned by the dataset creator.

MF=*mf* Mainframe identifier; 2 alphanumeric character identification. This parameter identifies the front-end station to which the dataset is to be staged. If omitted, the mainframe from which the issuing job originated is used. If MF is given a value of CRAY id and DC=IN, the dataset is disposed to the CRAY input queue, after first issuing a warning message.

NOTE

If using the DISPOSE macro, see the description of the DISPOSE control statement in part 2.

TID=*tid* Terminal identifier; 1-8 alphanumeric character identifier for the destination terminal. The default is the terminal of job origin.

DF=*df* Dataset format. This parameter defines whether the destination computer is to perform character conversion. The default is CB.

df is a 2-character alpha code defined for use on the front-end computer system. CRI suggests support of the following codes:

CD Character/deblocked. The front-end system performs character conversion from 8-bit ASCII, if necessary.

CB Character/blocked. No deblocking is performed at the CRAY-1 prior to staging. The front-end performs character conversion from 8-bit ASCII, if necessary.

BD Binary/deblocked. The front-end system performs no character conversion.

BB Binary/blocked. The front-end computer performs no character conversion. No deblocking is performed at the CRAY-1 prior to staging.

TR Transparent. No blocking/deblocking or character conversion is performed.

IC Interchange tape datasets only. In interchange format each tape block of data corresponds to a single logical record in COS blocked format.

Other codes may be added by the local site. Undefined pairs of characters may be passed but will be treated as transparent mode by the CRAY-1.

DC=*dc* Disposition code; disposition to be made of the dataset. The default is PR (print).

dc is a 2-character alpha code which describes the destination of the dataset as follows:

IN Input (job) dataset. The dataset is to be queued as a job on the mainframe specified by the MF parameter.

ST Stage to mainframe. Dataset is made permanent at the mainframe designated by the MF parameter.

SC Scratch dataset. Dataset is deleted.

PR Print dataset. Dataset is printed on any printer available at the mainframe designated by the MF parameter. PR is the default value.

PU Punch dataset. Dataset is punched on any card punch available at the mainframe designated by the MF parameter.

PT Plot dataset. Dataset is plotted on any available plotter at the mainframe designated by the MF parameter.

MT Write dataset on magnetic tape at the mainframe designated by the MF parameter.

SF=*sf* Special form information to be passed to the front-end system; 1-8 alphanumeric characters. SF is defined by the needs of the front-end system. Consult on-site analyst for options.

RT=*rt* Retention period; a value between 0 and 4095 specifying the number of days a permanent dataset is to be retained by the system. The default is an installation-defined value.

ED=*ed* Edition number; a value between 1 and 4095 assigned by the dataset creator. The default is the highest edition number known to the system.

RD=*rd* Read control word; 1-8 alphanumeric characters assigned by the dataset creator. The default is no read control word.

WT=*wt* Write control word; 1-8 alphanumeric characters assigned by the dataset creator. The default is no write control word.

MN=*mn* Maintenance control word; 1-8 alphanumeric characters assigned by the dataset creator. The default is no maintenance control word.

DT=*dt* Tape dataset generic device name or synonym. This parameter is required for tape datasets; it is ignored when used for mass storage datasets.

Generic Name	Synonym	Significance
*6250	*TAPE	Device capable of 6250 bpi
*1600		Device capable of 1600 bpi

CS=*cs* Character set of tape dataset, for data only. This parameter applies only to tape datasets; it is ignored when used for mass storage datasets.

AS	ASCII; default.
EB	EBCDIC

LB=*lb* Tape dataset label processing option. This parameter applies only to tape datasets; it is ignored when used for mass storage datasets.

BLP	By-pass label processing§
SL	IBM standard-labeled tapes
NL	Unlabeled tapes; default
AL	ANSI standard-labeled tapes

LDT=*ldt*§ Label Definition Table (LDT). The name of the LDT for tape processing. This parameter applies only to tape datasets; it is ignored when used for mass storage datasets. *ltd* is identical to *ldttag* on the LDT macro.

NEW=$\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$ Tape dataset is to be created; the dataset must be written starting at the beginning of information.

ON	Tape dataset to be created
OFF	Tape dataset not to be created; default.

MSG=$\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$ Normal completion message suppression indicator. The default is OFF.

ON	Indicator is set
OFF	Indicator is cleared

UQ=$\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$ Unique access. If UQ is specified, write maintenance and/or read permission may be granted if the appropriate write or maintenance control words are specified. The default (OFF) is multiread access if the read control word is specified.

WAIT=$\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$

Job wait/nowait. If WAIT=ON is specified, the job waits for the dataset to be transferred to the front-end system. If the transfer is canceled, the job is aborted. If WAIT=OFF is specified, the job resumes immediately and does not wait for the dataset to be transferred. If the transfer is canceled, the job is not aborted. If the parameter is omitted, an installation default parameter is used.

§ Deferred implementation

DEFER=$\left\{\begin{matrix}\text{ON}\\ \text{OFF}\end{matrix}\right\}$

Deferred disposition. When DEFER is specified, disposing of the dataset is delayed until the dataset is released either by a RELEASE request or by termination.

The default is OFF, in which case the dataset is disposed immediately.

NRLS=$\left\{\begin{matrix}\text{ON}\\ \text{OFF}\end{matrix}\right\}$

No release. When NRLS=ON is specified, the dataset remains local to the job after a DISPOSE request has been processed. The default is NRLS=OFF.

NOTE

The dataset is available only for reading when NRLS=ON is specified.

EXO=$\left\{\begin{matrix}\text{ON}\\ \text{OFF}\end{matrix}\right\}$ Execute-only dataset. EXO=ON sets the execute-only status of a dataset. EXO=OFF clears the execute-only status. If omitted, the status is ignored.

SID=*mf* Default source mainframe identifier. Two alphanumeric characters. This parameter defines the source front-end station where all staging to the CRAY-1 mainframe will default.

DID=*mf* Default destination mainframe identifier. Two alphanumeric characters. This parameter defines the destination front-end station where all staging from the CRAY-1 mainframe will default.

NOTE

Use of the MF parameter with either SID or DID is not allowed.

LDT - CREATE LABEL DEFINITION TABLE

The LDT macro creates a table called the Label Definition Table (LDT). (See Appendix A for a description of the LDT table). This macro is non-executable and may accompany the PDD and ACCESS macros in a program accessing a labeled tape dataset.

Format:

Location	Result	Operand

ldttag	LDT	CT=*ct*,VOL=(vsn_1,vsn_2,...vsn_n),FSEC=*fsec*,
		FSEQ=*fseq*,GEN=*gen*,GVN=*gvn*,CDT=*yyddd*,
		XDT=*yyddd*,RF=*rf*,MBS=*mbs*,RS=*rs*

ldttag Symbolic address of the LDT table; identical to *ldt* on PDD macro.

Parameters are in keyword form.

CT=*ct*§ Tape dataset conversion type. Required if run-time record and data format conversion is to be performed. The default is no conversion. *ct* is a 3-character code describing the machine internal data representation.

IBM IBM 370 and compatible

VOL=(vol_i)

Volume identifier list; a list of 6-character alphanumeric volume identifiers, separated by commas, that comprise the tape dataset. The maximum number of volume identifiers per dataset is an installation parameter.

FSEC=*fsec* File section number; a number from 1 through 9999 specifying the volume in the dataset. The first section (or volume) of a dataset is numbered 0001. The default is 1.

§ Deferred implementation

FSEQ=*fseq*[§]
File sequence number; a number from 1 through 9999 identifying this file among the files of this set. The first file is numbered 0001. The default is 1.

GEN=*gen*[§] Generation number; a number from 1 to 9999 that distinguishes successive generations of the file. The default is 1.

GVN=*gvn*[§] Generation version number; a number from 1 to 9999 that distinguishes among successive iterations of the same generation. The default is 0.

CDT=*yyddd* Creation date. *yy* specifies the year and is a number from 0-99. *ddd* specifies the day within the year and is a number from 001 to 366. It indicates the creation date for this file.

XDT=*yyddd* Expiration date; same format as creation date. It indicates the date on which this file may be overwritten.

RF=*rf*[§] Tape dataset record format. *rf* is a 2- to 8-character code describing the record type.

IU	IBM U (undefined) format; default if CT=IBM.
IF	IBM F (fixed) format
IFB	IBM FB (fixed blocked) format
IV	IBM V (variable) format
IVB	IBM VB (variable blocked) format
IVBS	IBM VBS (variable blocked spanned) format

RS=*rs* Record size. If CT=IBM, expressed in units of 8-bit bytes.

MBS=*mbs* Maximum tape block size; that is, the number of bytes in the largest tape block to be read or written. The maximum size allowed at the installation and the default are specified as installation parameters.

[§] Deferred implementation

PERMANENT DATASET MANAGEMENT

The user can access, save, adjust, and delete permanent datasets by use of the permanent dataset management macros. All of these macros must be accompanied by the PDD macro in the job.

ACCESS - ACCESS PERMANENT DATASET

The ACCESS macro associates an existing permanent dataset with a job and assures that the user is authorized to use this dataset. ACCESS must precede any logical I/O macros for the permanent dataset.

Format:

Location	Result	Operand
	ACCESS	*pddtag*

pddtag Address of PDD macro call

SAVE - SAVE PERMANENT DATASET

The SAVE macro enters a local dataset in the Dataset Catalog, making it permanent. A permanent dataset is uniquely identified by permanent dataset name, user identification, and edition number.

SAVE has a twofold function:

- Creation of an initial edition of a permanent dataset
- Creation of an additional edition of a permanent dataset

If all of the following conditions are true for the dataset, SAVE makes a call to close the dataset and consequently to flush the buffer. This assures that all the data is disk resident.

1. The dataset is currently opened for output only.
2. The dataset has not had an end-of-data written.
3. The dataset is being written sequentially.
4. The dataset has COS blocked dataset structure.
5. The dataset's DSP is managed by COS.

SAVE does not close the dataset unless all of these conditions are true. SAVE applies to mass storage datasets; it is ignored for tape datasets.

Format:

Location	Result	Operand
	SAVE	*pddtag*

pddtag Address of PDD macro call

DELETE - DELETE PERMANENT DATASET

The DELETE macro removes a permanent dataset from the Dataset Catalog. A dataset must be accessed within a job with the maintenance permission control word and unique access before a DELETE may be issued.

Format:

Location	Result	Operand
	DELETE	*pddtag*

pddtag Address of PDD macro call

ADJUST - ADJUST PERMANENT DATASET

The ADJUST macro changes the size of a permanent dataset, that is, redefines end-of-data for the dataset. A dataset must be accessed with the write permission control word and unique access within a job before an ADJUST may be issued.

If all of the following conditions are all true for the dataset, ADJUST makes a call to close the dataset and consequently to flush the buffer. This assures that all the data is disk resident.

1. The dataset is currently opened for output only.
2. The dataset has not had an end-of-data written.
3. The dataset is being written sequentially.
4. The dataset has COS blocked dataset structure.
5. The dataset's DSP is managed by COS.

ADJUST does not close the dataset unless all of these conditions are true.

ADJUST applies to mass storage datsets only; it is ignored when used with tape datasets.

Format:

Location	Result	Operand
	ADJUST	*pddtag*

pddtag   Address of PDD macro call

CFT LINKAGE MACROS 5

The CFT linkage macros handle subroutine linkage between CFT-compiled routines and CAL-assembled routines.

These macros perform the following functions:

- Generate code for calls, entries, and returns
- Assign B and T registers
- Fetch argument addresses

These macros should be used to maintain compatibility across the various versions of CFT.

CALL EXTERNAL ROUTINES

The CALL and CALLV macros generate code to call external routines using the call-by-address or call-by-value convention.

CALL - CALL EXTERNAL ROUTINE USING CALL-BY-ADDRESS CONVENTION

The CALL macro builds a list of addresses for a call-by-address external routine. This first argument address is stored at entry point minus one, the second at entry point minus two, etc of the called routine. The total number of arguments is entered in A7.

Format:

Location	Result	Operand
	CALL	*name*,(*argument list*)

name The name of the routine being called

argument list
The list of arguments to be passed (1 or more). If more than one argument is passed, the arguments must be separated by commas. Each argument may be a literal, a word address symbol, or a register containing an address.

CALLV - CALL EXTERNAL ROUTINE USING CALL-BY-VALUE CONVENTION

The CALLV macro generates a call-by-value to an external routine. The arguments are passed in registers S1 through S6 and must all be scalar quantities. The first argument is placed in S1, the second in S2, etc.

Format:

Location	Result	Operand
	CALLV	*name*,(*argument list*)

name The name of the routine being called

argument list
The list of arguments to be passed (1-6). If more than one argument is passed, the arguments must be separated by commas. Each argument may be either a literal, a word address symbol, or a register symbol. S registers are loaded in numerical order.

ENTER AND EXIT

The ENTER and EXIT macros generate code for entries and returns using the call-by-address convention for normal CFT calls or call-by-value convention for library routines.

ENTER - FORM A CFT CALLABLE ENTRY

The ENTER macro generates code for a normal CFT call-by-address entry. It reserves space for parameter addresses, saves B and T registers, and sets up traceback linkage. As a option, it also sets up call-by-value entries for library routines.

Format:

Location	Result	Operand

name	ENTER	NB=*nb*,NT=*nt*,NP=*np*,MODE=*mode*,
		TYPE=*type*,PRELOAD=*nsv*,COPYIN=*copy*,
		SHARED=*share*,ARGSIZE=*size*

name The name of the entry

NB=*nb* The number of B registers explicitly used by the routine, not including those that must be used by the calling sequence/traceback linkage. The default is 0.

NT=*nt* The number of T registers explicitly used by the routine. The default is 0.

NP=*np* The number of parameters to the routine. The default is 0.

MODE=*mode* The kind of entry (USER or LIBRARY). The default is USER. LIBRARY mode is intended for special purpose use in $FTLIB routines. In LIBRARY mode, NB and NT must both be 0.

TYPE=*type* The kind of calling sequence used to call *name*. The options are VALUE, ADDRESS, or BOTH. The default is ADDRESS. If BOTH is specified, the call-by-address entry precedes the call-by-value entry. If either BOTH or VALUE is specified, *name* must be the call-by-address name. The ENTER macro automatically appends a % to name for the call-by-value entry. *type* can be specified only in a LIBRARY mode entry.

PRELOAD=*nsv*

The number of parameters to be loaded into the S or V registers in a call-by-address entry. The default is *np*. PRELOAD can be specified only in a LIBRARY mode entry. If the first character of *name* is %, the arguments are loaded into V registers. Otherwise, they are loaded into S registers. V register arguments have two parts -- the first is the base address of the argument, the second is the address of the memory increment between argument values.

COPYIN=*copy*

If future calling sequences pass argument addresses in a different manner, COPYIN=ON will generate code to copy the addresses into locations corresponding to the current calling sequence conventions. The default is COPYIN=OFF.

SHARED=*share*

The name of a previous ENTER macro. If this parameter is specified, the previous entry and the current entry share storage space in which the B and T registers are saved. SHARED is intended for routines such as SIN and COS, which also share code sequences.

ARGSIZE=*size*

The size of the arguments. This parameter is only used in LIBRARY mode. If ARGSIZE=ONEWORD is specified, each argument is loaded into consecutive registers; that is, argument 1 is loaded in S1, argument 2 is loaded in S2, etc. If ARGSIZE=TWOWORD is specified, each argument is loaded into consecutive pairs of registers; that is, the first and second words of argument 1 are loaded into S1 and S2, the first and second words of argument 2 are loaded into S3 and S4, etc. The default is ARGSIZE=ONEWORD. Note that this parameter has meaning only when used with TYPE=ADDRESS or TYPE=BOTH.

EXIT - RETURN FROM A ROUTINE

The EXIT macro generates code to return program control from a routine to its caller. It restores those B or T registers used.

Format:

Location	Result	Operand
	EXIT	NB=*nb*,NT=*nt*,NAME=*name*,MODE=*mode*,KEEP=$\begin{Bmatrix}\text{ON}\\\text{OFF}\end{Bmatrix}$

Normally EXIT without any parameters is sufficient. The values from the most recently assembled (not necessarily executed) ENTER are used.

NB=*nb* The number of B registers to restore; not including those used for call linkage

NT=*nt* The number of T registers to restore

NAME=*name* The name of the entry corresponding to this exit

MODE=*mode* The same mode as that of the corresponding entry. The default is USER. If MODE=LIBRARY, EXIT assumes that (B0) has not been changed since the entry and that (B1) may be restored from (A1).

KEEP=$\left\{\begin{array}{l}\text{ON}\\\text{OFF}\end{array}\right\}$ Requests that EXIT save and restore any scratch registers used. The default is KEEP=OFF. If KEEP=ON (or KEEP=YES) is specified, any scratch registers used by EXIT are restored to their original values. If KEEP=OFF (or KEEP=NO) is specified, the original values are not restored.

REGISTER ASSIGNMENT

The BREG and TREG macros assign values to symbols for use as B and T register names.

BREG - ASSIGN SYMBOLS FOR B REGISTER NAMES

The BREG macro assigns numerical values to symbols for use as B register names. It also checks that no more registers are used than are declared on the ENTER macro. The register names are assigned after any registers used in the call linkage. The first B register assigned is B2.

Format:

Location	Result	Operand
br	BREG	

br A symbolic name used to designate a B register as in B.BR. BREG assigns *br* a numerical value in sequence.

TREG - ASSIGN SYMBOLS FOR T REGISTER NAMES

The TREG macro assigns numerical values to symbols for use as T register names. It also checks to see that no more registers are used than are declared on the ENTER macro. The register names are assigned after any registers used in the call linkage. The first T register assigned is T0.

Format:

Location	Result	Operand
tr	TREG	

tr A symbolic name used to designate a T register, as in T.TR. TREG assigns *tr* a numerical value in sequence.

FETCH ARGUMENT ADDRESS

ARGADD - FETCH ARGUMENT ADDRESS

The ARGADD macro fetches an argument address (not a value) and places it in an A register. It is used only for call-by-address routines.

NOTE

The EXIT, BREG, TREG, and ARGADD macros can be used only in conjunction with an ENTER macro.

Format:

Location	Result	Operand
	ARGADD	*result*,*n*,USE=*use*

result The result register (A or S) to be loaded with the n^{th} argument address

n Argument number

USE=*use* The intermediate register used if result register is an S register

APPENDIX SECTION

CONTENTS
APPENDIX SECTION

FIGURES

TABLES

JOB USER AREA

The user area of memory is assigned to one or more jobs. Figure A-1 illustrates the user area of one job. The shaded area is not accessible to the user.

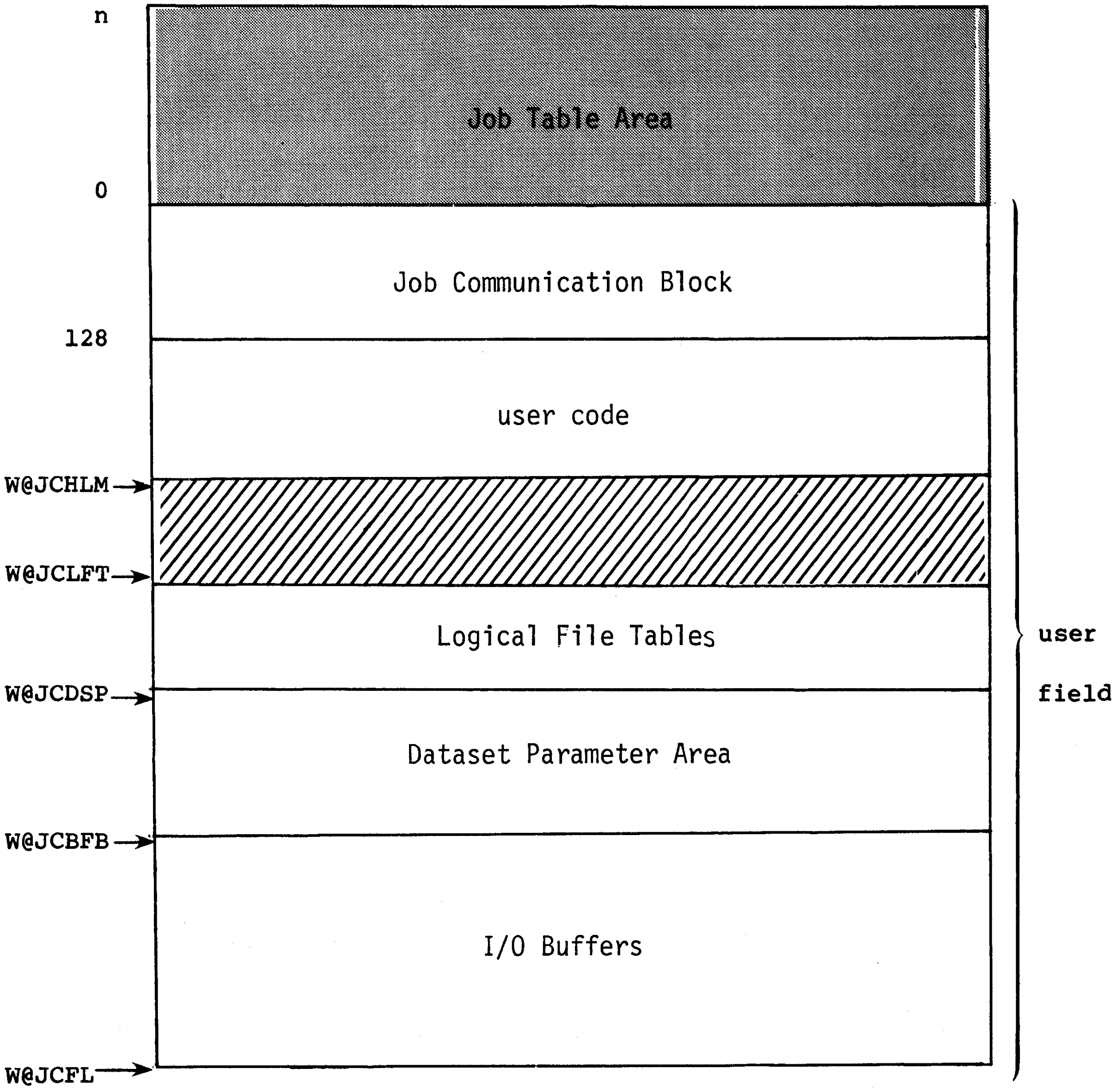


Figure A-1. User area of memory for a job

JOB TABLE AREA - JTA

Each job has an area referred to as the Job Table Area (JTA) preceding the field defined for the user. A JTA is accessible to the operating system but not to the user. The format of a JTA is described in the COS Table Descriptions Internal Reference Manual, CRI publication SM-0045. The Job Table Area contains job-related information such as accounting data; a JXT pointer; sense switches; an area for saving B, T, and V register contents, control statement and logfile DSPs; a logfile buffer; a copy of the user's LFTs; and a Dataset Name Table (DNT) for each dataset used by the job.

JOB COMMUNICATION BLOCK - JCB

Following the JTA is a 128-word block referred to as the Job Communication Block (JCB). The user accessible JCB contains a copy of the current control statement for the job and other job-related information.

Figure A-2 illustrates an expansion of the JCB.

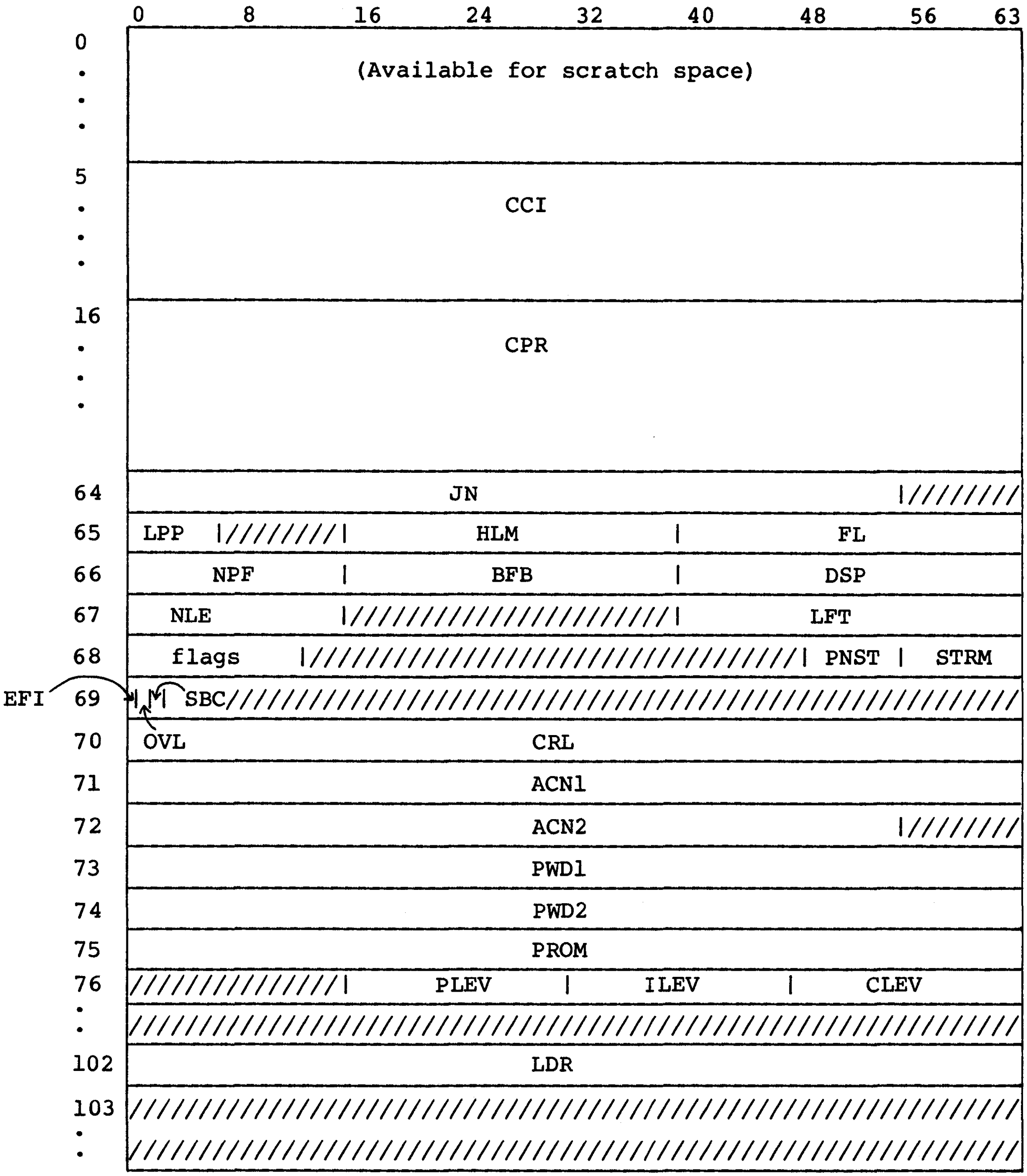


Figure A-2. Job Communication Block (JCB)

	0 8 16 24 32 40 48 56 63
118	BDAT
119	BTIM
120 • • • 127	DIG

Figure A-2. Job Communication Block (JCB) (continued)

Field	Word	Bits	Description
JCCCI	5-15	0-63	Control statement image packed 8 characters per word
JCCPR	16-63	0-63	Control statement parameters, expanded to 2 words per parameter
JCJN	64	0-55	Job name; bits 56-63 must be 0
JCLPP	65	0-7	Lines per page
JCHLM	65	16-39	High limit of user code
JCFL	65	40-63	Current field length
JCNPF	66	0-15	Number of physical buffers and datasets
JCBFB	66	16-39	Base address of I/O buffers
JCDSP	66	40-63	Base address of DSP area
JCNLE	67	0-15	Number of entries in LFT
JCLFT	67	40-63	Base of LFT
Flags:	68	0-12	
JCSIM		0	Simulator flag
JCCSDB		1	CSP debug flag
JCBP		2	JOB statement breakpoint (BP) flag
JCMRF		3	Memory request flag. If set, dynamic field management by CAL, LDR, etc. is not allowed.
JCIOAC		4	I/O area current status flag: 0 User's I/O area is unlocked 1 User's I/O area is locked

Field	Word	Bits	Description
Flags (continued):			
JCIOAP	68	5	I/O area previous status flag: 0 User's I/O area is unlocked 1 User's I/O area is locked
JCIA		6	Interactive flag
JCCHG		7	Execute CHARGES utility for trailer message
JCJBS		8	JOB statement flag (if set, JOB statement just processed)
JCCSIM		9	Flag is set when CRAY-1 simulator is running.
JCDLIT		10	Display literal delimiters in control statement crack.
JCRPRN		11	Retain level 1 parentheses
JCVSEP		12	Last character was valid separator.
JCPNST	68	48-55	Parentheses nesting level for current control statement
JCSTRM	68	56-63	Statement termination for current control statement
JCEFI	69	0	Enable floating interrupt flag; used by $FTLIB math routines to reset floating-point interrupt flag
JCOVL	69	1	Overlay flag
JCSBC	69	2	SBCA flag
JCCRL	70	0-63	COS revision level
JCCRLS	70	32-63	COS revision number
JCACN	71-72	0-63	1 through 15 character account number
JCACN1	71	0-63	Characters 1 through 8 of account number
JCACN2	72	0-55	Characters 9 through 15 of account number
JCPWD	73-74	0-63	1 through 15 character password
JCPWD1	73	0-63	Characters 1 through 8 of password
JCPWD2	74	0-55	Characters 9 through 15 of password

Field	Word	Bits	Description
JCPROM	75	0-63	Current user job interactive prompt, 1-8 ASCII characters, left-justified, zero-filled. 64 bits of binary zeroes disables user job prompt. Set to system default at beginning of each job step.
JCPLEV	76	16-31	Current procedure nesting level
JCILEV	76	32-47	Current iterative nesting level
JCCLEV	76	48-63	Current conditional nesting level
JCLDR	102	0-63	Unsatisfied externals
JCBDAT	118	0-63	Date of absolute load module generation
JCBTIM	119	0-63	Time of absolute load module generation
JCDIG	120-127	0-63	Reserved for diagnostics

LOGICAL FILE TABLE - LFT

The Logical File Table contains a 2-word entry for each dataset name and each alias for a dataset. Each entry points to the DSP for a dataset. Figure A-3 illustrates an LFT for a dataset.

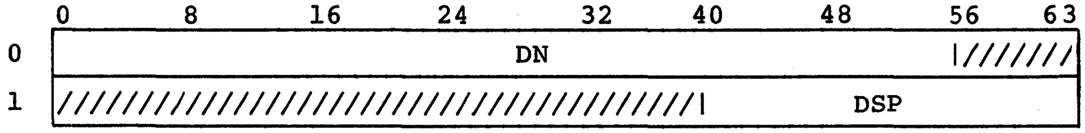


Figure A-3. Logical File Table (LFT) entry

Field	Word	Bits	Description
LFDN	0	0-55	Dataset name or alias
LFDSP	1	40-63	DSP address

DATASET PARAMETER AREA - DSP

Information concerning the status of a particular dataset and location of the I/O buffer for the dataset is maintained in the Dataset Parameter Area (DSP) of the user field. The DSP is illustrated in figure A-4.

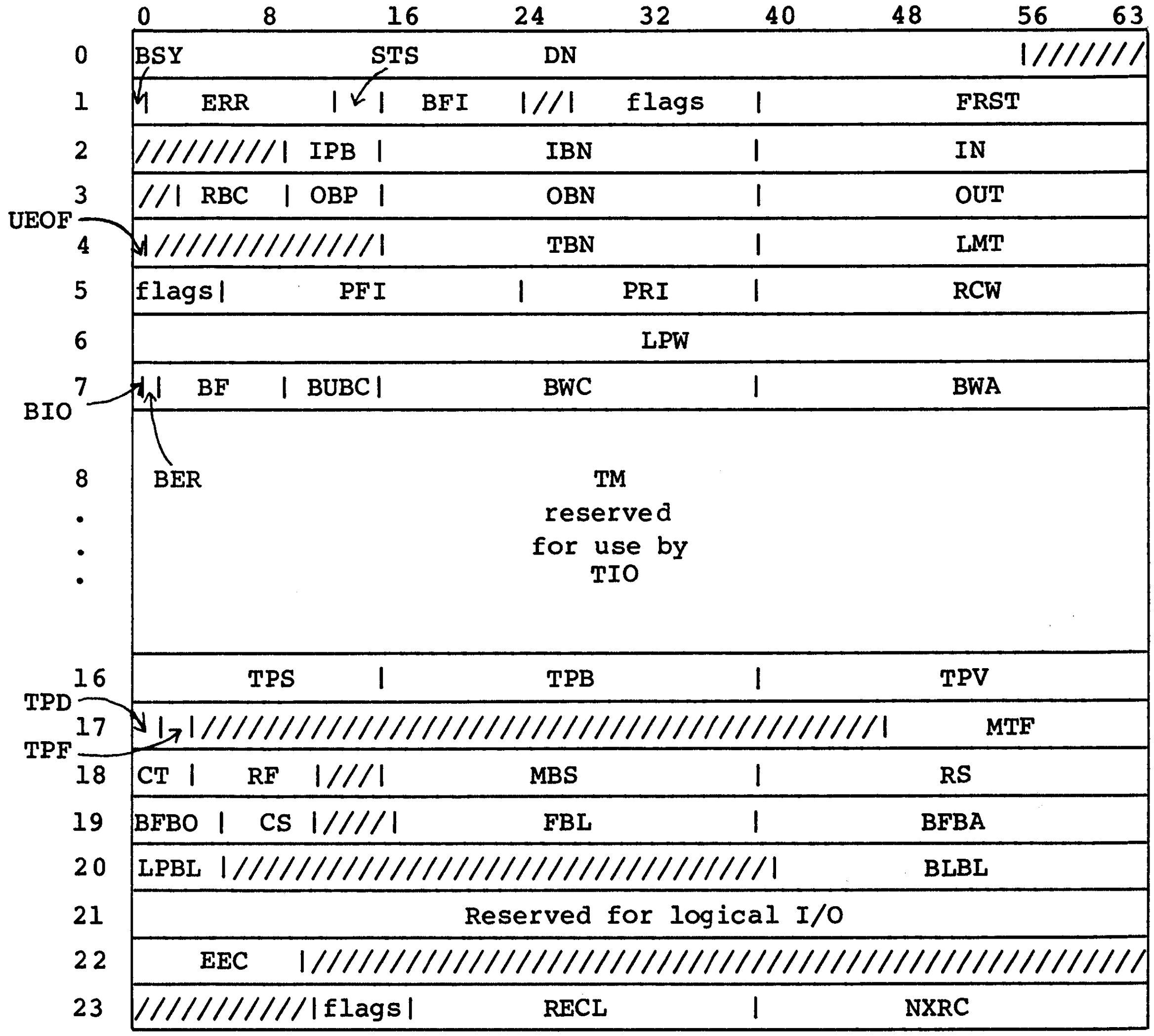


Figure A-4. Dataset Parameter Area (DSP)

Field	Word	Bits	Description
DPDN	0	0-55	Dataset name
DPBSY	1	0	Busy flag, circular I/O: 0 Not busy 1 Busy

Field	Word	Bits	Description
DPERR	1	1-12	Error flags:
DPEOI	1	1	End of data on read; write past allocated disk space on write
DPENX	1	2	Dataset does not exist
DPEOP	1	3	Dataset not open
DPEPD	1	4	Invalid processing direction
DPEBN	1	5	Block number error
DPEDE	1	6	Unrecovered data error
DPEHE	1	7	Unrecovered hardware error
DPERW	1	8	Attempted read after write or past EOD
DPEPT	1	9	Dataset prematurely terminated
DPELE	1	10	Unrecovered logical data error
	1	11	Reserved
DPEEP	1	12	Extended error (see DPEEC)
DPSTS	1	14-15	Status:
			00 Closed
			01 Open for input (I)
			10 Open for output (O)
			11 Open for I/O
DPBFI	1	16-24	Blank compression character in ASCII (BFI=777_8 implies no compression)
Flags:	1	28-39	
DPABD		28	Accept bad data flag
DPBDF		29	Bad data flag
DPTCS		30-31	Tape dataset character set
DPTP		32-33	Tape dataset (online/staged)
DPTRAN		34	Transparent mode for interactive dataset
DPIA		35	Dataset is interactive
DPMEM		36	Dataset is memory resident
DPRDM		37	Random dataset flag: 0 Sequential dataset 1 Random dataset
DPUDS		38	Undefined dataset structure: 0 COS blocked dataset structure 1 Undefined dataset structure
DPEND		39	Write end-of-data flag
DPFRST	1	40-63	Address of first word of buffer
DPIPB	2	10-15	Bit position in current input word (character I/O only)

Field	Word	Bits	Description
DPIBN	2	16-39	Block number, read request. System reads from block number until buffer is filled. DPIBN is then set to the next block number.
DPIN	2	40-63	Address of current input word
DPRBC	3	3-9	Remaining blank count
DPOBP	3	10-15	Bit position in current output word (character I/O only)
DPOBN	3	16-39	Block number, write request. System writes from block number until buffer is empty. The next block number is then in DPOBN.
DPOUT	3	40-63	Address of current output word
DPUEOF	4	0	Uncleared end-of-file (EOF)
DPTBN	4	16-39	Temporary block number; used by random I/O for last block read.
DPLMT	4	40-63	Address of last word+1 of buffer. LMT minus FRST defines buffer size.
Flags:	5	0-4	
DPEOR		0	EOR flag
DPEOF		2	EOF flag
DPEOD		3	EOD flag
DPRW		4	Previous operation read/write flag: 0 Read 1 Write
DPPFI	5	5-24	Previous file index; backward index to block containing previous EOF.
DPPRI	5	25-39	Previous record index; backward index to block containing previous EOR.
DPRCW	5	40-63	Control word address: Previous RCW address if in write mode Next RCW if in read mode

Field	Word	Bits	Description
DPLPW	6	0-63	Last partial word; used for character mode I/O.
DPBIO	7	0	Buffered I/O busy: 0 Buffered I/O operation complete 1 Buffered I/O operation incomplete
DPBER	7	1	Buffered I/O error flag
DPBF	7	2-9	Function code: 000 Read partial 010 Read record 040 Write partial 050 Write record 052 Write end-of-file 056 Write end-of-data
DPBPD	7	4	Processing direction: 0 Read 1 Write
DPBEO	7	6-9	Termination condition: 00 Partial 10 Record 12 File, write only 16 Dataset, write only
DPBUBC	7	10-15	Unused bit count; must be specified on a write record request. Value returned on a read request.
DPBWC	7	16-39	Word count; number of words at DPBWA to read or write. Field contains actual number of words read when request is completed.
DPBWA	7	40-63	Word address of user data area
DPTM	8-15	0-63	Used by TIO as follows:
	8	0-63	Saved word W@DPPRI
	9	0-63	Saved A2 in WB30
	10	16-39	$RWDP/$WWDP return address
	10	40-63	$RWDP/$WWDP first word address (FWA)

Field	Word	Bits	Description
DPTM (continued)			
	11	16-39	WB30/$WEOF return address
	11	40-63	$WEOD return address
	12	0-7	JTA length/1000_8 when registers are saved
	12	8-15	Bits 0-7 of RBLK/WBLK A5
	12	16-39	(B.ZE)
	12	40-63	RBLK/WBLK B0
	13	16-39	DNT address
	13	40-63	(A7) JXT address
	13	0-63	RBLK/WBLK S5 during task recall
	14	0-15	Bits 8-23 of RBLK/WBLK A5
	14	16-39	RBLK/WBLK A2
	14	40-63	RBLK/WBLK A3
	15	0-63	RBLK/WBLK S6
DPTPS	16	0-15	Online tape status
DPTPB	16	16-39	Tape maximum block size in bytes
DPTPV	16	40-63	Tape pointer to label definition table
DPTPD	17	0-1	Tape density
DPTPF	17	2-3	Tape format
DPMFT	17	48-63	Maintenance test field (used by DQM)

DPCT	18	0-3	Conversion type; nonzero if run-time data and record format conversion selected.
			DPCTNONE=0 No conversion
			DPCTIBM=1 IBM format data

DPRF	18	4-11	Record format (if DPCT nonzero)
			DPRFNONE=0 None
			DPRFIU=1 IBM undefined format
			DPRFIF=2 IBM fixed format
			DPRFIFB=3 IBM fixed blocked format
			DPRFIV=4 IBM variable format
			DPRFIVB=5 IBM variable blocked format
			DPRFIVBS=6 IBM variable block span format

DPMBS	18	16-39	Maximum block size
DPRS	18	40-63	Record length

Field	Word	Bits	Description
DPBFBO	19	0-5	User data area current bit offset

DPCS	19	6-11	Character set (if DPCT nonzero):
			DPCSA=0 ASCII, 8 bits/character
			DPCSE=1 EBCDIC, 8 bits/character

DPBFBL	19	16-39	User data area current bit length
DPBFBA	19	40-63	User data area current address
DPLPBL	20	0-5	Last partial word bit length
DPBLBL	20	40-63	Current tape block bit length
Reserved	21	0-63	Reserved for logical I/O
DPEEC	22	0-11	Error code if DPEEP is set; correspond to EXP abort codes.
Flags:	23	12-15	
DPDEL		12	FORTRAN file status: 0 Keep 1 Delete
DPBLNK		13	FORTRAN numeric input blank conversion: 0 Null 1 Zero
DPDIR		14	FORTRAN direct access flag
DPUFMT		15	FORTRAN unformatted I/O flag
DPRECL	23	16-39	FORTRAN direct access record length (in number of characters)
DPNXRC	23	40-63	FORTRAN direct access next record number

PERMANENT DATASET DEFINITION TABLE - PDD

The PDD is a parameter list that gives input to the Permanent Dataset Manager. The contents of PDD are illustrated in figure A-5.

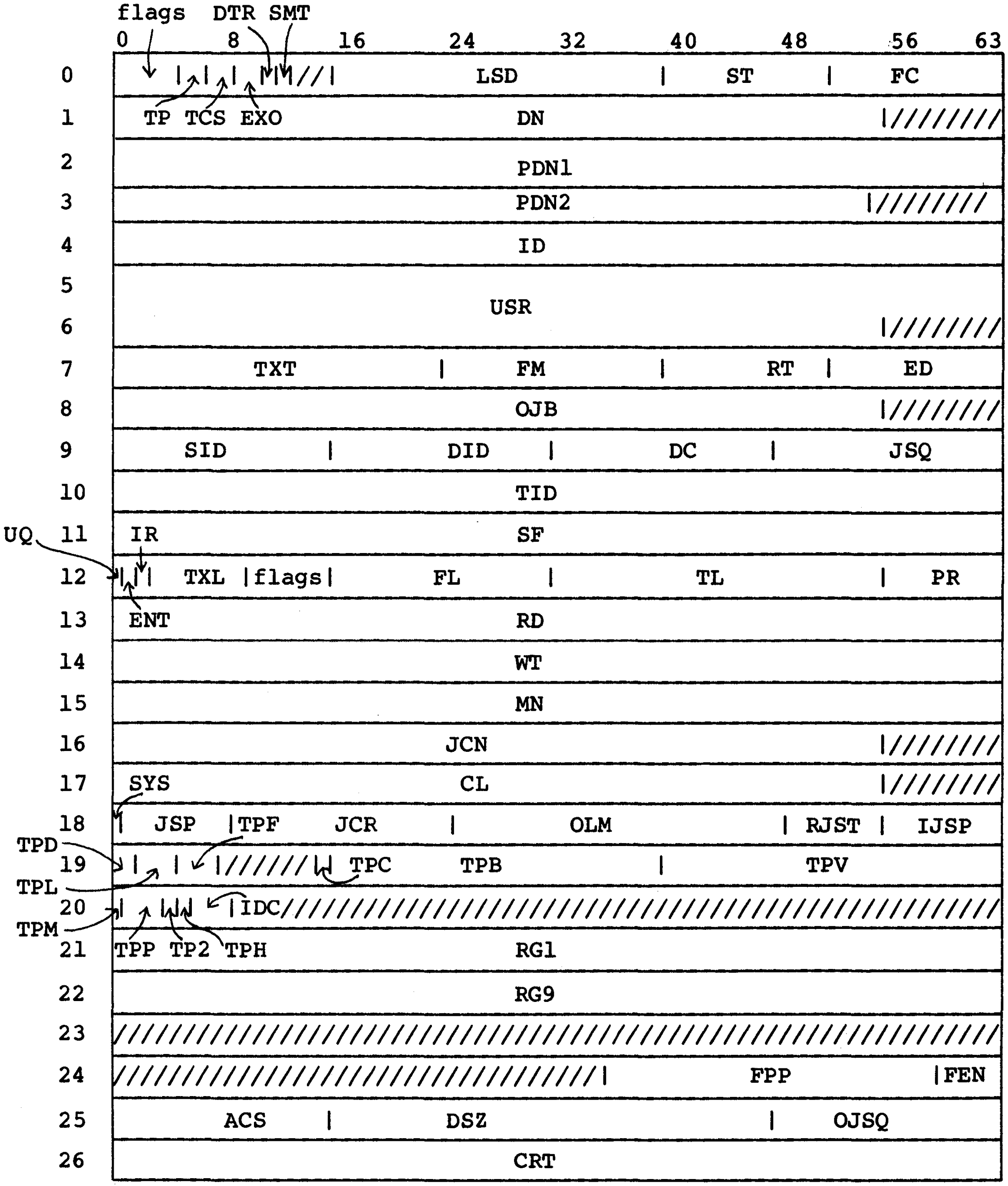


Figure A-5. Permanent Dataset Definition Table (PDD)

<table>
<tr><th></th><th>0</th><th>8</th><th>16</th><th>24</th><th>32</th><th>40</th><th>48</th><th>56 63</th></tr>
<tr><td>27</td><td colspan="8">ACT</td></tr>
<tr><td>28</td><td colspan="8">TDM</td></tr>
<tr><td>29</td><td colspan="8">MOD</td></tr>
<tr><td>30</td><td>SSC</td><td>TXC</td><td colspan="2">MML</td><td colspan="4">///////////////////////////////////</td></tr>
</table>

Figure A-5. Permanent Dataset Definition Table (PDD) (continued)

Field	Word	Bits	Description
Flags:	0	0-4	
PMSG		0	Normal completion message suppression indicator
PMERR		1	Error message suppression indicator
PMWAIT		2	WAIT flag for a disposed dataset
PMNRLS		3	No release of dataset on DISPOSE
PMAQR		4	Acquire flag for accounting
PMTP	0	5-6	Tape dataset (online/staged)
PMTCS	0	7-8	Tape dataset character set
PMEXO	0	9-10	Execute only
PMDTR	0	11	Update dump-time on PDSDUMP access
PMSMT	0	12	Submit flag
PMLSD	0	16-39	Temporary SDT address for load input/output
PMST	0	40-51	Return status; the codes are defined in Appendix F.
PMFC	0	52-63	Function code
PMDN	1	0-55	Local dataset name
PMPDN	2-3	0-63	Permanent dataset name
PMPDN1	2	0-63	Characters 1-8
PMPDN2	3	0-55	Characters 9-15
PMID	4	0-63	User identification

Field	Word	Bits	Description
PMUSR	5-6	0-63	User number
PMUSR1	5	0-63	Characters 1-8
PMUSR2	6	0-55	Characters 9-15
PMTXT	7	0-23	Address of optional text field
PMFM	7	24-39	Format designator (two characters):
			FMCD=CD Character/deblocked
			FMCB=CB Character/blocked
			FMBD=BD Binary/deblocked
			FMBB=BB Binary/blocked
PMRT	7	40-51	Retention period; 0-4095 days
PMED	7	52-63	Edition number (0-4095)
PMOJB	8	0-55	Originating job name
PMSID	9	0-15	Source ID; 2 characters.
PMDID	9	16-31	Destination ID; 2 characters.
PMDC	9	32-47	Disposition code; 2 characters.
			DCIN=IN Job dataset
			DCST=ST Dataset to be staged
			DCSC=SC Scratch dataset
			DCPR=PR Print dataset
			DCPU=PU Punch dataset
			DCPT=PT Plot dataset
			DCMT=MT Magnetic tape dataset
PMJSQ	9	48-63	Job sequence number
PMTID	10	0-63	Terminal ID; 1-8 characters.
PMSF	11	0-63	Special forms
PMUQ	12	0	Unique access required
PMENT	12	1	Enter in System Directory
PMIR	12	2	Immediate reply requested
PMTXL	12	3-10	Number of words of text

Field	Word	Bits	Description
PMNRR	12	11	Job rerun flag; set if job cannot be rerun (input entries only).
PMINIT	12	12	Job initiate flag; set if job has been initiated.
PMIA	12	13	Interactive flag
PMDFR	12	14	Deferred disposition indicator
PMNA	12	15	No abort flag. If set, processing continues even if an error is encountered.
PMFL	12	16-31	Field length/512 (input datasets only)
PMTL	12	32-55	Time limit (input datasets)
PMPR	12	56-63	Priority (input datasets)
PMRD	13	0-63	Read permission control word
PMWT	14	0-63	Write permission control word
PMMN	15	0-63	Maintenance permission control word
PMJCN	16	0-55	Job class name
PMCL	17	0-55	CL parameter from JOB statement
PMSYS	18	0	System job
PMJSP	18	1-8	JOB statement priority
PMJCR	18	9-24	Job class rank
PMOLM	18	25-48	Size of $OUT in 512-word block
PMRJST	18	49-55	Job status flag
PMIJSP	18	56-63	Original job card priority
PMTPD	19	0-1	Tape density
PMTPL	19	2-4	Tape label type
PMTPF	19	5-6	Tape format
PMTPC	19	15	Tape cataloged dataset

Field	Word	Bits	Description
PMTPB	19	16-39	Tape maximum block size in bytes
PMTPV	19	40-63	Tape pointer to label definition table
PMTPM	20	0	Tape online maintenance access
PMTPP	20	1-3	Tape parallel device count
PMTP2	20	4	Tape second device assignment
PMTPH	20	5	Tape hold assigned device
PMIDC	20	6-8	Tape initial desposition code
PMRG1	21	0-63	First word of resource generic names
PMRG9	22	0-63	Second word of resource generic names
PMFPE	24	36-63	First DSC page/entry for dataset
PMFPP	24	36-59	First DSC page for dataset
PMFEN	24	60-63	First entry for dataset
PMACS	25	0-15	Number of accesses (load saved datasets only)
PMDSZ	25	16-47	Size of dataset as reflected by DSC DAT bodies (used only when a pseudo access is performed during the recovery of rolled jobs)
PMOJSQ	25	48-63	Originating job sequence number
PMCRT	26	0-63	Creation time in cycles (load request only)
PMACT	27	0-63	Time of last access in cycles (load request only)
PMTDM	28	0-63	Time of last dump in cycles (load request only)
PMMOD	29	0-63	Time of last modification in cycles (load request only)
PMSSC	30	0-7	Station slot word length
PMTXC	30	8-15	Text field word length
PMMML	30	16-27	Interactive maximum message length

BEGIN CODE EXECUTION TABLE - BGN

The BGN table specifies necessary parameters to begin execution of code loaded into the user area by the Control Statement Processor. Figure A-6 illustrates the BGN.

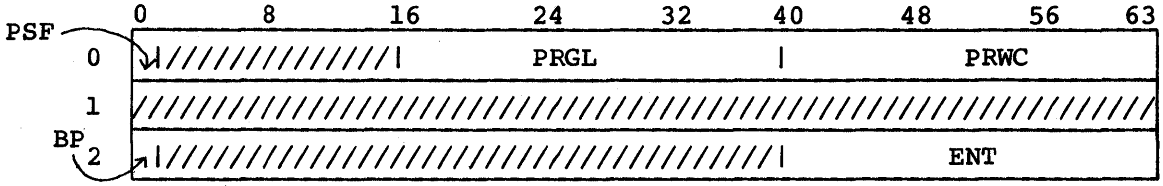


Figure A-6. Begin Code Execution Table (BGN)

Field	Word	Bits	Description
BGPSF	0	0	Preset value flag
BGPRGL	0	16-39	Total program length including blank common
BGPRWC	0	40-63	Program word count
BGBP	2	0	Breakpoint flag
BGENT	2	40-63	Program entry point P-address

DATASET DEFINITION LIST - DDL

A Dataset Definition List in the user field must accompany any create DNT (F$DNT) request. The DDL is illustrated in figure A-7.

```
     0     8     16     24     32     40     48     56   63
   ----------------------------------------------------------
0  |                         DN                  |////////|
   ----------------------------------------------------------
1  |                         LDV                            |
   ----------------------------------------------------------
2  | flags|  BFI  |//////////////////////////|      SZ      |
   ----------------------------------------------------------
3  |///////////////|      DNT      |/////////|  BFZ (BSZ)  |
   ----------------------------------------------------------
4  |//////////////////////////////////////////////|   DC    |
   ----------------------------------------------------------
5  |//////////////////////////////////////////|     LM      |
   ----------------------------------------------------------
```

Figure A-7. Dataset Definition List (DDL)

Field	Word	Bits	Description
DDDN	0	0-55	Dataset name
DDLDV	1	0-63	Logical device name
Flags:	2	0-6	
DDRDM		0	Random dataset flag: 0 Sequential 1 Random
DDUDS		1	Undefined dataset structure: 0 COS blocked dataset structure 1 Undefined structure
DDNFE		2	Return error if dataset does not exist. Register S0 returned nonzero if DNT does not exist; no DNT is created.
DDSTAT		3	Request dataset statistics; ignored unless DDNFE=1 (see DDDNT)
DDMR		4	Dataset is to be memory resident
DDIA		5	Interactive type dataset
DDTRAN		6	Transparent mode for interactive dataset
DDBFI	2	7-15	Blank field indicator for character I/O 000_8 BFI=I@BFI $< 400_8$ BFI=user specified ASCII character $= 400_8$ BFI=000 $> 400_8$ Blank compression disabled

Field	Word	Bits	Description
DDSZ	2	40-63	Dataset size in 512-word blocks
DDDNT	3	16-39	Address of DNT image returned by F$DNT when DDNFE=1 and DDSTAT=1
DDBFZ	3	49-63	Buffer size in 512-word blocks
DDBSZ	3	49-63	Alternate name for DDBFZ to match $SYSTXT name

DDDC	4	48-63	Disposition code (two characters):
			DCIN=IN Job dataset
			DCST=ST Staged permanent dataset
			DCSC=SC Scratch dataset
			DCPR=PR Print dataset
			DCPT=PT Plot dataset
			DCPU=PU Punch dataset
			DCMT=MT Magnetic tape dataset

DDLM	5	40-63	Dataset size limit in 512-word blocks

OPEN DATASET NAME TABLE - ODN

A 2-word Open Dataset Name Table (ODN) is generated in the user field the first time an OPEN of the specified dataset is encountered. Figure A-8 illustrates the ODN.

```
       0      8       16      24      32      40      48      56   63
      ---------------------------------------------------------------
  0  | flags                   DN                             |//////// |
      ---------------------------------------------------------------
  1  |/| ↓ |//|            LDT            |//////|        DSP          |
      ---------------------------------------------------------------
```

Figure A-8. Open Dataset Name Table (ODN)

Field	Word	Bits	Description
ODDN	0	0-55	Dataset name
Flags:	1	1-4	
ODV§		1	Close volume
ODM§		2	Open for 'MOD'
ODS§		3	Close/open with saved position
ODH§		4	Hold resources
ODLDT§	1	8-32	LDT address
ODDSP	1	40-63	DSP pointer: Negative Negative offset from beginning of DSPs Positive Offset from user base address

§ Deferred implementation

OPTION TABLE - OPT

The Option Table (OPT) is used for F$OPT calls. Figure A-9 illustrates the OPT.

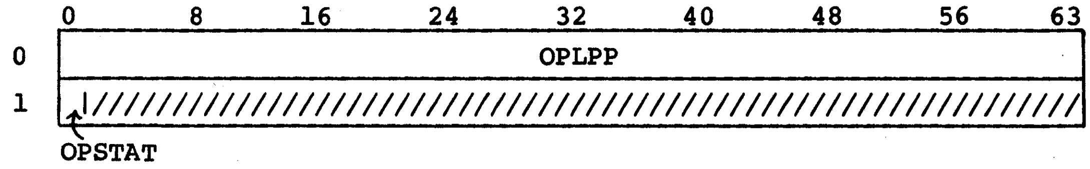


Figure A-9. Option Table (OPT)

Field	Word	Bits	Description
OPLPP	0	0-63	Page length
OPSTAT	1	0	DSP address

JCL BLOCK INFORMATION TABLE - JBI

The 1-word JCL Block Information Table (JBI) is generated in the user field and has two formats: one for conditional information (figure A-10) and the other for interative information (figure A-11).

Conditional block information:

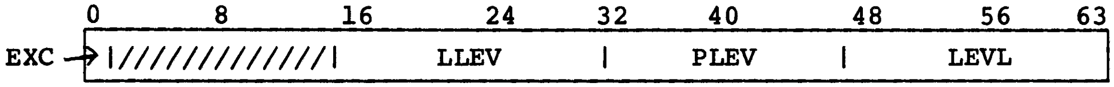


Figure A-10. JCL conditional block information

Field	Bits	Description
JBEXC	0	Conditional sequence is in execution
JBLLEV	16-31	Conditional is contained in this iterative nesting level
JBPLEV	32-47	Conditional is contained in this procedure level
JBLEVL	48-63	Current conditional nesting level

Iterative block information:

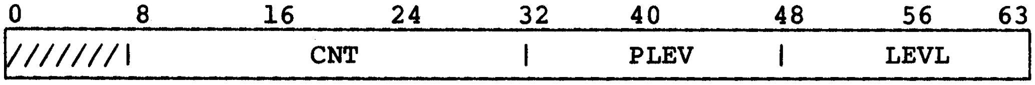


Figure A-11. JCL iterative block information

Field	Bits	Description
JBCNT	8-31	Iteration count
JBPLEV	32-47	Iterative is contained in this procedure level
JBLEVL	48-63	Current iterative nesting level

JCL SYMBOL TABLE - JST

The 4-word JCL Symbol Table (JST) is generated in the user field and contains information about system and user symbols. See figure A-12.

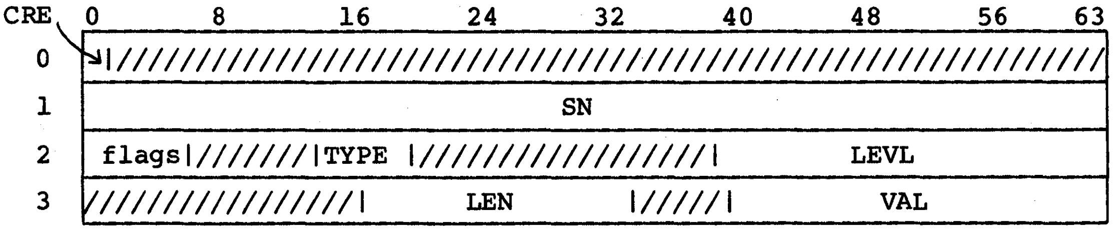


Figure A-12. JCL Symbol Table (JST)

Field	Word	Bits	Description
JSCRE	0	0	Create if not found. Available only for system use.
JSSN	1	0-63	Symbol name
Flags:	2	0-4	
JSLOC		0	Local or global. If set, symbol is procedure local.
JSCON		1	Constant or variable. If set, symbol is constant.
JSSRS		2	System reserved. If set, the symbol name is reserved by the system.
JSUSR		3	User settable. If set, symbol may be modified by the job.
JSSYS		4	System settable. If set, the symbol may be modified by COS.
JSTYPE	2	10-15	One of the following symbol types: SYMTUDF 00 Undefined - no type SYMTBOO 01 Boolean - logical SYMTINT 02 Decimal integer SYMTLIT 03 ASCII literal; 1-8 characters.
JSLEVL	2	40-63	Procedure definition level
JSLEN	3	12-35	Length of value
JSVAL	3	40-63	Base of value buffer

LABEL DEFINITION TABLE - LDT

The following conditions must be met for constructing a Label Definition Table (LDT):

- The table order must include an LDT header, volume entry, header 1 entry, and header 2 entry.
- The length value for either header 1 or header 2 must be at least the defined length of the respective entry.
- The length value for Volume 1 must be at least the length of the entire first VSN.

Header:

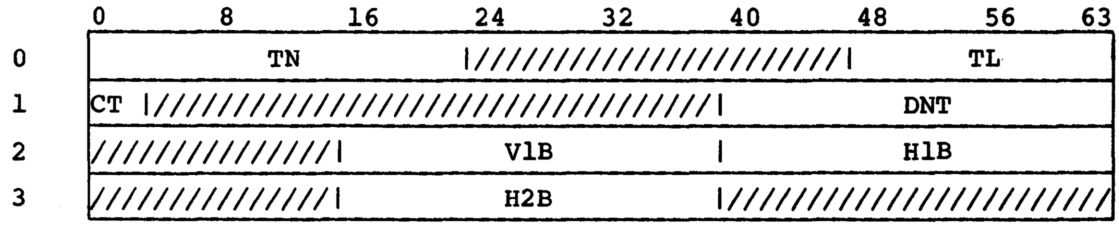


Figure A-13. LDT header

Field	Word	Bits	Description
LDTN	0	0-23	Table name
LDTL	0	48-63	Table length (variable)
LDCT	1	0-3	Conversion type
LDDNT	1	40-63	Dataset name table (DNT) pointer
LDV1B	2	16-39	Offset of volume 1 entry, relative to LDT base
LDH1B	2	40-63	Offset of header 1 entry, relative to LDT base
LDH2B	3	16-39	Offset of header 2 entry, relative to LDT base

Volume 1 entry:

	0	8	16	24	32	40	48	56 63
0	VOL1				////////	////////	VL1L	
1	VSNL		CVN		////////	////////	////////	////////
2	VSN1						////////	////////

Figure A-14. LDT volume 1 entry

Field	Word	Bits	Description
LDVOL1	0	0-31	Volume 1 label identifier
LDVL1L	0	48-63	Volume 1 length
LDVSNL	1	0-15	Number of VSNs in list
LDCVN	1	16-31	Current VSN ordinal
LDVSN1	2	0-47	Beginning VSN

Header 1 entry:

	0	8	16	24	32	40	48	56 63
0	HDR1				////////	////////	HR1L	
1	FID1							
2	FID2							
3	FID3							
4	FID4							
5	FID5							
6	FID6				////////	////////	////////	////////
7	FSEC				CSEC			
8	FSEQ				////////	////////	////////	////////
9	GEN				////////	////////	GVN	
10	CDT						////////	////////
11	XDT						DRT	
12	BLK						////////	////////
13	SET						////////	////////

Figure A-15. LDT header 1 entry

Field	Word	Bits	Description
LDHDR1	0	0-31	Header 1 label identifier
LDHR1L	0	48-63	Header 1 length
LDFID1	1	0-63	Characters 1-8 of file identifier
LDFID2	2	0-63	Characters 9-16 of file identifier
LDFID3	3	0-63	Characters 17-24 of file identifier
LDFID4	4	0-63	Characters 25-32 of file identifier
LDFID5	5	0-63	Characters 33-40 of file identifier
LDFID6	6	0-31	Characters 41-44 of file identifier
LDFSEC	7	0-31	File section number
LDCSEC	7	32-63	Current file (volume) section number
LDFSEQ	8	0-31	File sequence number
LDGEN	9	0-31	Generation number
LDGVN	9	48-63	Generation version number
LDCDT	10	0-47	Creation date
LDCSP		0-7	Space
LDCYR		8-23	Year
LDCDY		24-47	Day
LDXDT	11	0-47	Expiration date
LDXSP	11	0-7	Space
LDXYR	11	8-23	Year
LDXDY	11	24-47	Day
LDRT	11	48-63	Retention period
LDBLK	12	0-47	Block count
LDSET	13	0-47	File set identifier

Header 2 entry:

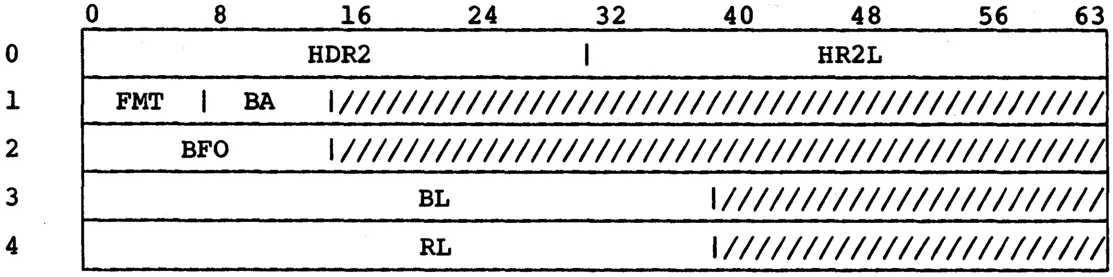


Figure A-16. LDT header 2 entry

Field	Word	Bits	Description
LDHDR2	0	0-31	Header 2 label identifier
LDHR2L	0	48-63	Header 2 length
LDFMT	1	0-7	Record format. Valid values: F, V, U IBM label types F, D, S ANSI label types
LDBA	1	8-15	Block attributes, IBM standard label: B Blocked records S Spanned records R Blocked and spanned records ' ' No blocked or spanned records
LDBFO	2	0-15	Buffer offset
LDBL	3	0-39	Block length in bytes
LDRL	4	0-39	Record length in bytes

CHARACTER SET B

This appendix describes the 128 control and graphic characters comprising the ASCII character set. Those numbers, letters, and special characters that form the CRAY-1 FORTRAN character set are identified by the appearance of the letter C in the fourth column. All other characters are members of the auxiliary character set. The letter A in the fourth column of the table indicates those characters belonging to the ANSI FORTRAN character set. Note that all control characters are grouped on the first page.

CONTROL CHARACTER	ASCII OCTAL CODE	ASCII PUNCHED-CARD CODE	FORTRAN (A=ANSI) (C=CRAY)	DESCRIPTION
NUL	000	12-0-9-8-1		Null
SOH	001	12-9-1		Start of heading (CC)
STX	002	12-9-2		Start of text (CC)
ETX	003	12-9-3		End of text (CC)
EOT	004	9-7		End of transmission (CC)
ENQ	005	0-9-8-5		Enquiry (CC)
ACK	006	0-9-8-6		Acknowledge (CC)
BEL	007	0-9-8-7		Bell (audible or attention signal)
BS	010	11-9-6		Backspace (FE)
HT	011	12-9-5		Horizontal tabulation (FE)
LF	012	0-9-5		Line feed (FE)
VT	013	12-9-8-3		Vertical tabulation (FE)
FF	014	12-9-8-4		Form feed (FE)
CR	015	12-9-8-5		Carriage return (FE)
SO	016	12-9-8-6		Shift out
SI	017	12-9-8-7		Shift in
DLE	020	12-11-9-8-1		Data link escape (CC)
DC1	021	11-9-1		Device control 1
DC2	022	11-9-2		Device control 2
DC3	023	11-9-3		Device control 3
DC4	024	9-8-4		Device control 4 (stop)
NAK	025	9-8-5		Negative acknowledge (CC)
SYN	026	9-2		Synchronous idle (CC)
ETB	027	0-9-6		End of transmission block (CC)
CAN	030	11-9-8		Cancel
EM	031	11-9-8-1		End of medium
SUB	032	9-8-7		Substitute
ESC	033	0-9-7		Escape
FS	034	11-9-8-4		File separator (IS)
GS	035	11-9-8-5		Group separator (IS)
RS	036	11-9-8-6		Record separator (IS)
US	037	11-9-8-7		Unit separator (IS)
DEL	177	12-9-7		Delete

Legend: CC - Communication control
FE - Format effector
IS - Information separator

GRAPHIC HARACTER	ASCII OCTAL CODE	ASCII PUNCHED-CARD CODE	FORTRAN (A=ANSI) (C=CRAY)	DESCRIPTION
(Space)	040	(None)	A,C	Space (blank)
!	041	12-8-7		Exclamation point
"	042	8-7	C	Quotation marks (diaeresis)
#	043	8-3		Number sign
$	044	11-8-3	A,C	Dollar sign (currency symbol)
%	045	0-8-4		Percent
&	046	12		Ampersand
'	047	8-5	C	Apostrophe (closing single quotation mark)
(	050	12-8-5	A,C	Opening (left) parenthesis
)	051	11-8-5	A,C	Closing (right) parenthesis
*	052	11-8-4	A,C	Asterisk
+	053	12-8-6	A,C	Plus
,	054	0-8-3	A,C	Comma (cedilla)
-	055	11	A,C	Minus (hyphen)
.	056	12-8-3	A,C	Period (decimal point)
/	057	0-1	A,C	Slant (slash, virgule)
0	060	0	A,C	Zero
1	061	1	A,C	One
2	062	2	A,C	Two
3	063	3	A,C	Three
4	064	4	A,C	Four
5	065	5	A,C	Five
6	066	6	A,C	Six
7	067	7	A,C	Seven
8	070	8	A,C	Eight
9	071	9	A,C	Nine
:	072	8-2	C	Colon
;	073	11-8-6		Semicolon
<	074	12-8-4		Less than
=	075	8-6	A,C	Equal
>	076	0-8-6		Greater than
?	077	0-8-7		Question mark

GRAPHIC CHARACTER	ASCII OCTAL CODE	ASCII PUNCHED-CARD CODE	FORTRAN (A=ANSI) (C=CRAY)	DESCRIPTION
@	100	8-4		Commercial at
A	101	12-1	A,C	Upper-case letters
B	102	12-2	A,C	
C	103	12-3	A,C	
D	104	12-4	A,C	
E	105	12-5	A,C	
F	106	12-6	A,C	
G	107	12-7	A,C	
H	110	12-8	A,C	
I	111	12-9	A,C	
J	112	11-1	A,C	
K	113	11-2	A,C	
L	114	11-3	A,C	
M	115	11-4	A,C	
N	116	11-5	A,C	
O	117	11-6	A,C	
P	120	11-7	A,C	
Q	121	11-8	A,C	
R	122	11-9	A,C	
S	123	0-2	A,C	
T	124	0-3	A,C	
U	125	0-4	A,C	
V	126	0-5	A,C	
W	127	0-6	A,C	
X	130	0-7	A,C	
Y	131	0-8	A,C	
Z	132	0-9	A,C	
[	133	12-8-2		Opening (left) bracket
\	134	0-8-2		Reverse slant (backslash)
]	135	11-8-2		Closing (right) bracket
^	136	11-8-7		Circumflex
_	137	0-8-5		Underline

GRAPHIC CHARACTER	ASCII OCTAL CODE	ASCII PUNCHED-CARD CODE	FORTRAN (A=ANSI) (C=CRAY)	DESCRIPTION
`	140	8-1		Grave accent (opening single quotation mark)
a	141	12-0-1	C	Lower-case letters
b	142	12-0-2	C	
c	143	12-0-3	C	
d	144	12-0-4	C	
e	145	12-0-5	C	
f	146	12-0-6	C	
g	147	12-0-7	C	
h	150	12-0-8	C	
i	151	12-0-9	C	
j	152	12-11-1	C	
k	153	12-11-2	C	
l	154	12-11-3	C	
m	155	12-11-4	C	
n	156	12-11-5	C	
o	157	12-11-6	C	
p	160	12-11-7	C	
q	161	12-11-8	C	
r	162	12-11-9	C	
s	163	11-0-2	C	
t	164	11-0-3	C	
u	165	11-0-4	C	
v	166	11-0-5	C	
w	167	11-0-6	C	
x	170	11-0-7	C	
y	171	11-0-8	C	
z	172	11-0-9	C	
{	173	12-0		Opening (left) brace
¦	174	12-11		Vertical line
}	175	11-0		Closing (right) brace
~	176	11-0-1		Overline (tilde, general accent)

FUNCTION CODES C

The system function codes and permanent dataset function codes are listed in the COS EXEC/STP/CSP Internal Reference Manual, publication SM-0040.

LOGICAL I/O ROUTINES D

LOGICAL RECORD I/O ROUTINES

The logical record I/O routines are divided into three basic groups: read routines, write routines, and positioning routines.

READ ROUTINES

The read routines transfer partial or full records of data from the I/O buffer to the user data area. The data is placed in the user data area one character per word or in full words depending on the read request issued. Figure D-1 provides an overview of the logical read operation.

$RWDP - Read words, partial mode

Words are transmitted from the I/O buffer defined by the Dataset Parameter Area (DSP) to the area beginning at first word address (FWA) until either the word count in A3 is satisfied or an end-of-record is encountered.

Unrecovered data errors do not abort the job; control is returned to the caller instead. The caller can use the good data read, (A2) through (A4)-1, and then abort. The user can also skip or accept the bad data.[§] If the caller does nothing, the job aborts when the next read request occurs. See the Library Reference Manual, CRI publication SR-0014, for detailed descriptions of SKIPBAD and ACPTBAD.

SUBROUTINE NAME: $RWDP

ENTRY CONDITIONS:

(A1) Address of DSP or negative DSP offset relative to DSP base (JCDSP), that is, contents of second word of Open Dataset Name Table (ODN)

(A2) FWA of user data area

(A3) Word count. If count is 0, no data is transferred

[§] Deferred implementation

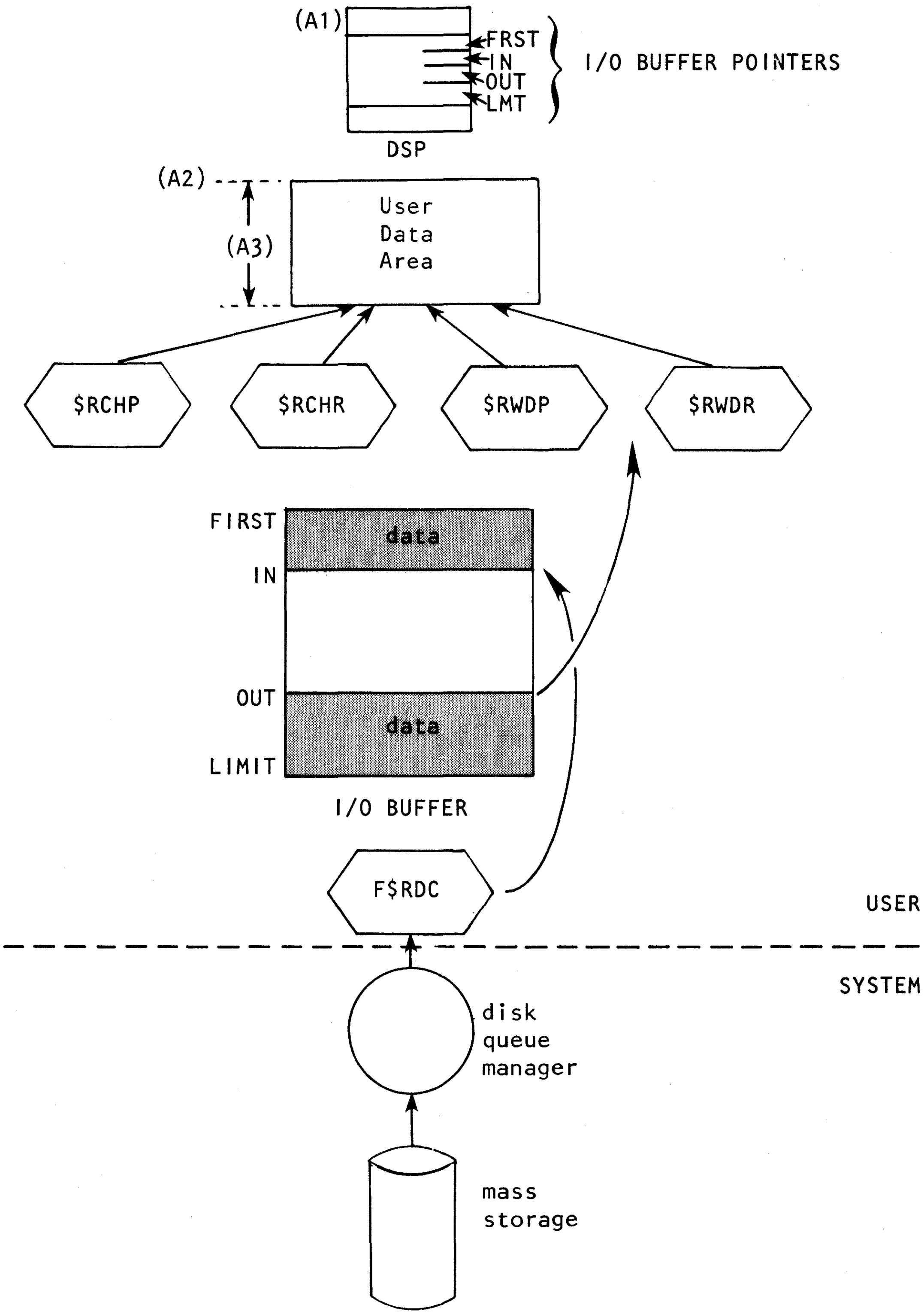


Figure D-1. Logical read

RETURN CONDITIONS:

(A1) Address of DSP

(A2) FWA of user data area

(A3) Word count

(A4) Actual LWA+1 (equals FWA if null record)

(S0) Termination mode

< 0	Read terminated by end-of-record
= 0	Null record, end-of-file, end-of-data, or unrecovered data error encountered§
> 0	Read terminated by count. If count is exhausted simultaneously with reaching end-of-record, the end-of-record takes precedence.

(S1) Error Status§

= 0	No errors encountered
= 1	Unrecovered data error encountered

(S6) Contains RCW if (S0) $\leq$ 0 and (S1)=0

REGISTERS MODIFIED:

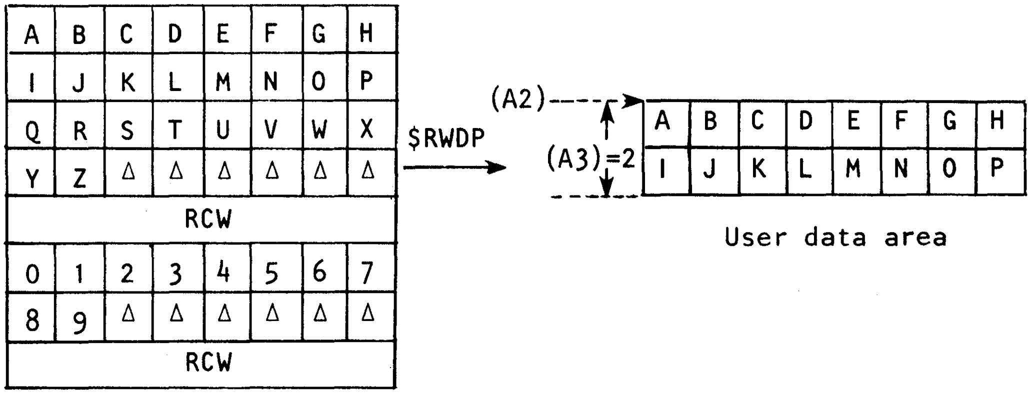
A0, A1, A4, A5, A6

B.ZA, B.ZB (within $B70_8...B77_8$)

S0, S1, S2, S3, S4, S5, S6

T.ZA (within $T70_8...T77_8$), V0, V1

Example:



User data area

Data in I/O buffer

§ Deferred implementation

$RWDR - Read words, record mode

This routine resembles $RWDP. However, following the read, the dataset is positioned after the end-of-record that terminated the current record.

Unrecovered data errors cause control to return to the caller. The caller can use the good data read, (A2) through (A4)-1, and then abort. The user can also skip or accept the bad data.§ If the caller does nothing, the job aborts when the next read request occurs. See the Library Reference Manual, CRI publication SR-0014, for detailed descriptions of SKIPBAD and ACPTBAD.

SUBROUTINE NAME:	$RWDR
ENTRY CONDITIONS:	Same as $RWDP
RETURN CONDITIONS:	Same as $RWDP
REGISTERS MODIFIED:	Same as $RWDP

$RCHP - Read characters, partial mode

The $RCHP routine unpacks characters from the I/O buffer defined by the Dataset Parameter Area (DSP) and inserts them into the user data area beginning at the first word address (FWA) specified by (A2) until either the count is satisfied or an end-of-record is encountered. If an end-of-record is encountered first, the remainder of the field specified by the character count is filled with blanks.

Unrecovered data errors cause control to be returned to the caller. The caller can use the good data read, (A2) through (A4)-1; and then abort. The user can also skip or accept the bad data.§ If the caller does nothing, the job aborts when the next read request occurs. See the Library Reference Manual, CRI publication SR-0014, for detailed descriptions of SKIPBAD and ACPTBAD.

§ Deferred implementation

SUBROUTINE NAME: $RCHP

ENTRY CONDITIONS:

(A1) Address of DSP or negative DSP offset relative to DSP base (JCDSP), that is, contents of second word of Open Dataset Name Table (ODN)

(A2) FWA of user data area

(A3) Character count. If count is 0, no data is transferred.

RETURN CONDITIONS:

(A1) Address of DSP

(A2) FWA of user data area

(A3) Character count

(A4) Actual LWA+1 (equals FWA if null record)

(S0) Termination mode

< 0 Read terminated by end-of-record
= 0 Null record, end-of-file, end-of-data, or unrecovered data error encountered[§]
> 0 Read terminated by count. If count is exhausted simultaneously with reaching end-of-record, the end-of-record takes precedence

(S1) Error status[§]

= 0 No error encountered
= 1 Unrecovered data error encountered

(S6) Contains RCW if (S0) $\leq$ 0 and (S1)=0

REGISTERS MODIFIED:

A0, A1, A4, A5, A6

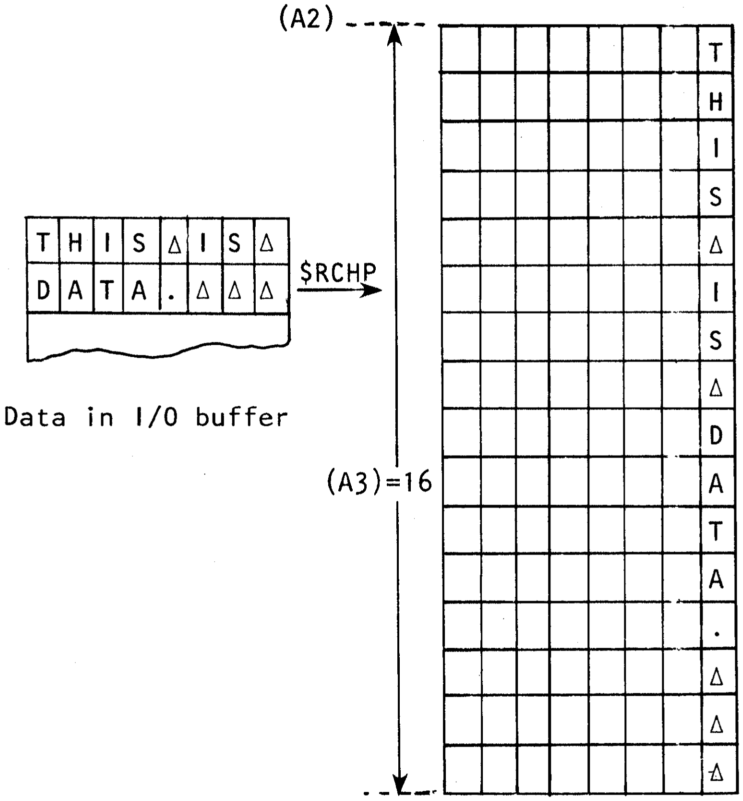
B.ZA, B.ZB (within $B70_8$...$B77_8$)

S0, S1, S2, S3, S4, S5, S6

T.ZA (within $T70_8$...$T77_8$)

[§] Deferred implementation

Example:



User data area

$RCHR - Read characters, record mode

This routine resembles $RCHP. However, following the read, the dataset is positioned after the end-of-record that terminates the current record.

Unrecovered data errors cause control to be returned to the caller. The caller can use the good data read, (A2) through (A4)-1, and then abort, skip the bad data,[§] or accept the bad data.[§] If the caller does nothing, the job aborts when the next read request occurs. See the Library Reference Manual, SR-0014, for detailed descriptions of SKIPBAD and ACPTBAD.

SUBROUTINE NAME:	$RCHR
ENTRY CONDITIONS:	Same as for $RCHP
RETURN CONDITIONS:	Same as for $RCHP
REGISTERS MODIFIED:	Same as for $RCHP

[§] Deferred implementation

WRITE ROUTINES

The write routines transfer partial or full records of data from the user data area to the I/O buffer. The data is taken from the user data area one character per word and packed eight per word or is transferred in full words depending on the write operation requested. Figure D-2 provides an overview of the logical write operation.

$WWDP - Write words, partial mode

The number of words specified by the count is transmitted from the area beginning at first word address (FWA) and is written in the I/O buffer defined by the Dataset Parameter Area (DSP).

SUBROUTINE NAME: $WWDP

ENTRY CONDITIONS:

(A1) Address of DSP or negative DSP offset relative to DSP base (JCDSP), that is, contents of second word of Open Dataset Name Table (ODN)

(A2) FWA of user data area

(A3) Word count. If count is 0, no data is transferred.

RETURN CONDITIONS:

(A1) Address of DSP

(A2) FWA of user data area

(A3) Word count

(A4) LWA+1

REGISTERS MODIFIED:

A0, A1, A4, A5, A6

B.ZA, B.ZB (within B70$_8$...B77$_8$)

S0, S1, S2, S3, S4, S5, S6

T.ZA (within T70$_8$...T77$_8$)

V0, V1

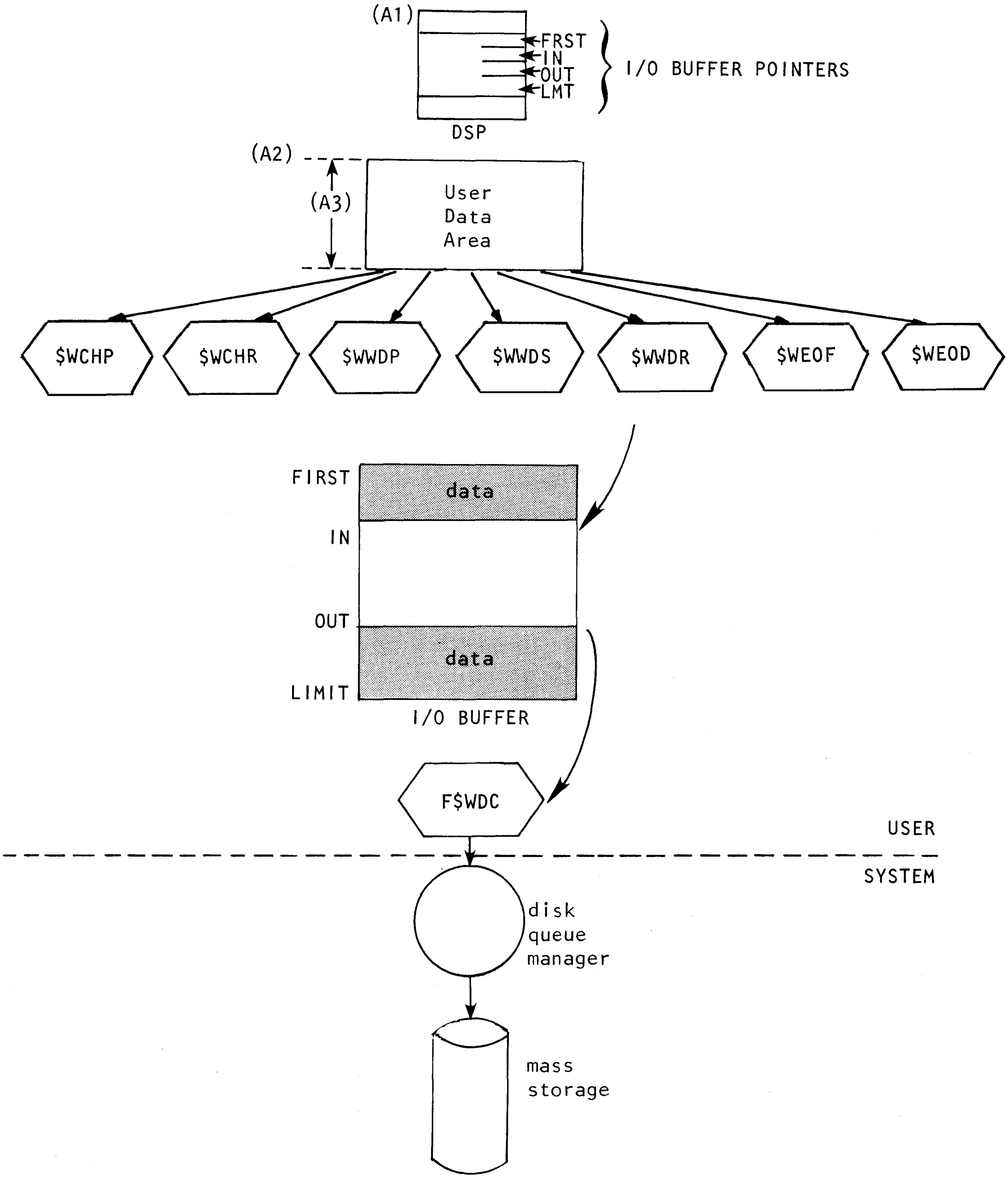
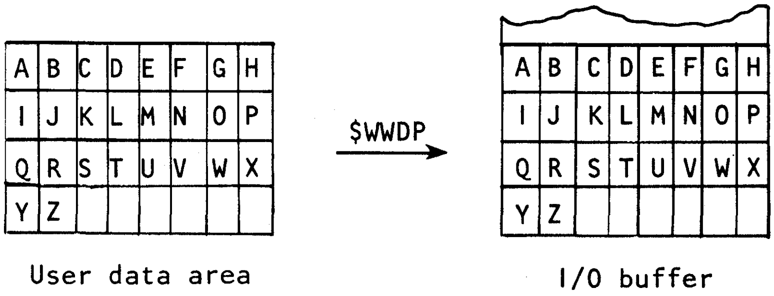


Figure D-2. Logical write

Example:



$WWDR - Write words, record mode

The $WWDR routine resembles $WWDP. However, an end-of-record RCW terminating the record is inserted in the I/O buffer in the next word following the data. To write simply an end-of-record, the user issues a $WWDR with (A3)=0.

SUBROUTINE NAME: $WWDR

ENTRY CONDITIONS: Same as $WWDP

RETURN CONDITIONS: Same as $WWDP

REGISTERS MODIFIED: Same as $WWDP

$WWDS - Write words, record mode with unused bit count

The $WWDS routine resembles $WWDR. However, the user may specify the unused bit count in the last word of the record as an entry condition.

SUBROUTINE NAME: $WWDS

ENTRY CONDITIONS: Same as $WWDP with the addition of the following:

(A4) Unused bit count in the last word of the record; a value from 0 through 63.

RETURN CONDITIONS: Same as $WWDP

REGISTERS MODIFIED: Same as $WWDP

$WCHP - Write characters, partial mode

The $WCHP routine packs the number of characters specified by the count from the user area defined at first word address (FWA) to the I/O buffer for the dataset defined by the Dataset Parameter Area (DSP). The number of characters specified by the count is packed from the area beginning at FWA to the dataset defined by DSP.

SUBROUTINE NAME: $WCHP

ENTRY CONDITIONS:

(A1) Address of DSP or negative DSP offset relative to DSP base (JCDSP), that is, contents of second word of ODN

(A2) FWA of user data area

(A3) Character count. If count is 0, no data is transferred.

RETURN CONDITIONS:

(A1) Address of DSP

(A2) FWA of user data area

(A3) Character count

(A4) LWA+1

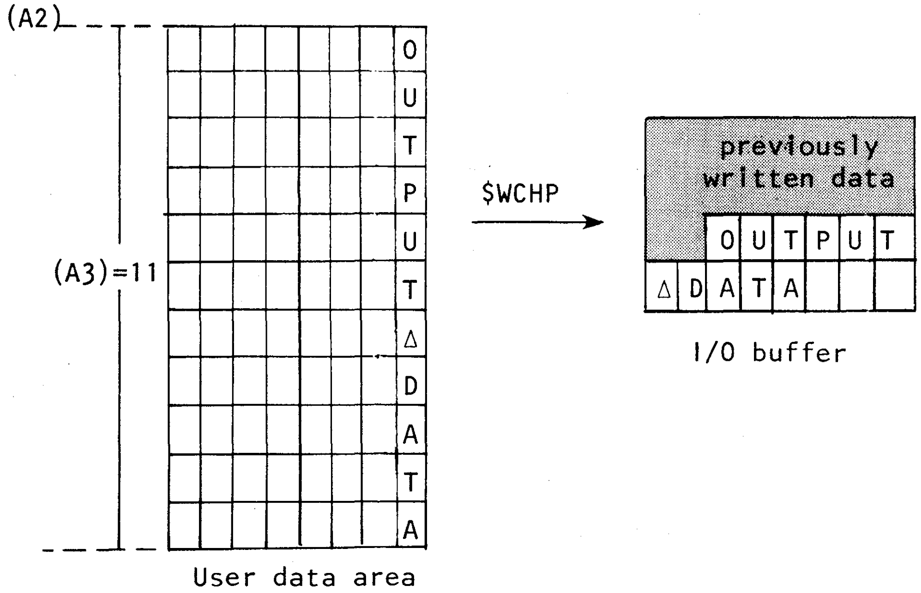
REGISTERS MODIFIED:

A0, A1, A4, A5, A6

B.ZA, B.ZB (within $B70_8...B77_8$)

S0, S1, S2, S3, S4, S5, S6

Example:



$WCHR - Write characters record mode

The $WCHR routine resem les $WCHP. However, an end-of-record RCW terminating the record is inserted in the I/O buffer in the next full word following the data. The unused bit count in the RCW specifies the end of data in the previous word. To write an end-of-record, the user issues a $WCHR with (A3)=0. The RCW is written in the next full word.

SUBROUTINE NAME:	$WCHR
ENTRY CONDITIONS:	Same as $WCHP
RETURN CONDITIONS:	Same as $WCHP
REGISTERS MODIFIED:	Same as $WCHP

$WEOF - Write end of file

This routine writes an end-of-file RCW preceded by an end-of-record RCW if necessary as the next words in the I/O buffer. If the previous operation was a call to $WCHP, then a call to $WCHR with (A3)=0 is necessary to set the UBC since $WEOF does not check for a partial characte- write in progress.

SUBROUTINE NAME:	$WEOF	
ENTRY CONDITIONS:	(A1)	Address of Dataset Parameter Area (DSP) or negative DSP offset relative to DSP base (JCDSP), that is, contents of second word of Open Dataset Name Table (ODN)
RETURN CONDITIONS:	(A1)	Address of DSP
REGISTERS MODIFIED:	A0, A1, A2, A3, A4, A5, A6	
	B.ZC (within B70_8...B77_8)	
	S0, S1, S2, S3, S4, S5, S6	
	T.ZB (within T70_8...T77_8)	

$WEOD - Write end of data

This routine writes an end-of-data RCW preceded by an end-of-file and an end-of-record if necessary as the next words in the I/O block. If the previous operation was a call to $WCHP, then a call to $WCHR with (A3)=0 is necessary to set the UBC since $WEOD does not check for a partial character write in progress.

The $WEOD forces the final block of data to be written on the disk; that is, it flushes the I/O buffer. The dataset is left positioned before the end-of-data.

SUBROUTINE NAME: $WEOD

ENTRY CONDITIONS: (A1) Address of Dataset Parameter Area (DSP) or negative DSP offset relative to DSP base (JCDSP), that is, contents of second word of Open Dataset Name Table (ODN)

RETURN CONDITIONS: (A1) Address of DSP

REGISTERS MODIFIED: A0, A1, A2, A3, A4, A5, A6

B.ZD (within B70$_8$...B77$_8$)

S0, S1, S2, S3, S4, S5, S6

T.ZB (within T70$_8$...T77$_8$)

POSITIONING ROUTINES

The positioning routines, except for $GPOS, set the current processing direction to input (reading). If the processing direction was previously output (writing), on a sequential dataset $WEOD is called to write an end-of-data and the buffer is flushed. On a random dataset, the buffer is flushed.

$REWD - Rewind dataset

The $REWD routine positions the dataset at beginning-of-data. It functions as a no-op if the dataset is already positioned there.

SUBROUTINE NAME: $REWD

ENTRY CONDITIONS: (A1) Address of Dataset Parameter Area (DSP) or negative DSP offset relative to DSP base (JCDSP), that is, contents of second word of Open Dataset Parameter Area (ODN)

RETURN CONDITIONS: (A1) Address of DSP

REGISTERS MODIFIED: A0, A1, A2, A3, A4, A5, A6

S0, S1, S2, S3, S4, S5, S6

$BKSP - Backspace one record

The $BKSP routine positions the dataset after the previous end-of-record RCW. If the dataset is positioned just after an end-of-record RCW, $BKSP positions it just before the end-of-file RCW, that is, $BKSP treats an end-of-file RCW as if it were a normal record.

SUBROUTINE NAME: $BKSP

ENTRY CONDITIONS: (A1) Address of Dataset Parameter Area (DSP) or negative DSP offset relative to DSP base (JCDSP), that is, contents of second word of Open Dataset Name Table (ODN)

RETURN CONDITIONS: (A1) Address of DSP

(S6) RCW locationg after which dataset is positioned; equals 0 if at beginning-of-data

REGISTERS MODIFIED: A0, A1, A2, A3, A4, A5, A6

S0, S1, S2, S3, S4, S5, S6

$BKSPF - Backspace one file

The $BKSPF routine positions a dataset after the previous end-of-file RCW or at beginning-of-data if there is no previous end-of-file. The function is a no-op if the dataset is at beginning-of-data.

SUBROUTINE NAME: $BKSPF

ENTRY CONDITIONS: Same as $BKSP

RETURN CONDITIONS: Same as $BKSP

REGISTERS MODIFIED: Same as $BKSP

$GPOS - Get current dataset position

The $GPOS routine returns the current dataset position, including the current word address and flags that indicate whether the dataset is positioned at a record, a file, or a dataset boundary.

This routine does not alter the dataset position.

SUBROUTINE NAME: $GPOS

ENTRY CONDITIONS: (A1) Address of Dataset Parameter Area (DSP) or negative DSP offset relative to DSP base (JCDSP), that is, contents of second word of Open Dataset Name Table (ODN)

RETURN CONDITIONS: (A1) DSP address

(S1) Dataset position

Flags - the upper 4 bits of S1 indicate record, file, and dataset boundaries:

<u>bits</u> <u>significance</u>

0 End-of-record flag. 1 indicates the dataset is positioned at a record boundary, that is, following a record control word. 0 indicates the dataset is either at beginning of data or in the middle of a record.

1 Unused

2 End-of-file flag. 1 indicates the dataset is at a file boundary, that is, following the end of file RCW.

3-30 Unused

31-63 Word address. This is the current physical word address within the dataset, including record control words.

Note: The entire word in S1 is 0 at beginning-of-data.

REGISTERS MODIFIED: A0, A1, A2, A3

S0, S1, S2, S3, S4

$SPOS - Set current dataset position

The $SPOS routine positions the dataset at the position specified. The position must be at a record boundary, that is, at beginning-of-data or following an end-of-record or end-of-file, or before an end-of-data. A dataset cannot be positioned beyond the current end-of-data.

SUBROUTINE NAME: $SPOS

ENTRY CONDITIONS: (A1) Address of Dataset Parameter Area (DSP) or negative DSP offset relative to DSP base (JCDSP), that is, contents of second word of Open Dataset Name Table (ODN)

(S1) Dataset position

bits	significance
0-30	Unused
31-63	Word address. The desired physical word address within the dataset, including record control words

Special cases: (S1) = -1 Denotes end-of-data. The dataset is positioned at end-of-data, that is, before the end-of-data record control word.

(S1) = 0 Denotes beginning-of-data

RETURN CONDITIONS: (A1) DSP address

(S1) Dataset position

(S6) Contains RCW after which the dataset is positioned; (S6)=0 if at beginning-of-data

REGISTERS MODIFIED: A0, A1, A2, A3, A4, A5, A6

S0, S1, S2, S3, S4, S5, S6

FORTRAN LEVEL I/O

FORTRAN I/O consists of formatted and unformatted I/O routines, buffered I/O routines, and positioning and control I/O routines.

Although they do not perform I/O in the strict sense, the encode/decode routines are also described in this section.

FORMATTED AND UNFORMATTED I/O ROUTINES

These routines are divided into six basic groups: read formatted, write formatted, read unformatted, write unformatted, encode, and decode.

Routines in the four read and write groups transfer data between user locations and the system I/O buffer area allocated to a dataset and associated with a particular I/O unit. Routines in the encode and decode groups transfer data to or from user locations and a user-supplied buffer. The buffer contains eight characters per word and has no I/O unit association. All dataset processing by these routines is sequential.

Each of the six groups is accessed through a minimum of two calls: the first to an initiation routine and the last to a termination routine. Optionally, one or more calls may be made to either of two transfer routines between initiation and termination routine calls. The initiation routine name is identified by an I uffix, the termination routine name by an F suffix.

Transfer routines are of two types: call by address and call by value. Routine names are suffixed by an A if a call-by-address routine or by a V if a call-by-value routine. Both types of routines can be called within the same sequence.

These routines are named and their functions summarized in the chart below:

OPERATION SEQUENCE	READ FORMATTED	WRITE FORMATTED	READ UNFORMATTED	WRITE UNFORMATTED	DECODE	ENCODE
INITIATION ROUTINES	$RFI	$WFI	$RUI	$WUI	$DFI	$EFI
TRANSFER ROUTINES CALL BY ADDRESS	$RFA	$WFA	$RUA	$WUA	$DFA	$EFA
TRANSFER ROUTINES CALL BY VALUE	$RFV	$WFV	$RUV	$WUV	$DFV	$EFV
TERMINATION ROUTINES	$RFF	$WFF	$RUF	$WUF	$DFF	$EFF

Type-checking entry points - Each transfer routine has six different entry points. Each entry point corresponds to a particular type of data to be processed and is specified as the name of the routine (*xnam*) plus a (Parcel) increment value. These entry points and the FORTRAN data types they accommodate are:

Entry Point	Type of data
xnam or *xnam* + 0	Typeless (Boolean)
xnam + 3	Integer
xnam + 6	Real
xnam + 9	Double precision
xnam + 12	Complex
xnam + 15	Logical

The increment entry point names are used by the FORTRAN run-time system to verify the correspondence between variable types and format specifications.

In transfer routines that process formatted data, double-precision values must be specified by using the *xnam* + 9 entry. All other types of values may use the appropriately incremented entry or the *xnam* entry. If the *xnam* entry is used, typing is determined from format specification edit descriptors. If the *xnam* + offset entry is used, the format specification must be compatible with the type implied by the entry offset. Transfer routines processing unformatted data must be entered at *xnam* + 9 and *xnam* + 12 for double-precision and complex values. Values of all other types may be processed by using the appropriately incremented entry or the *xnam* entry.

Format specifications identified by initiation routines and used by transfer routines are described in the CRAY-1 (CFT) FORTRAN Reference Manual.

If an end-of-file record is read, zeros or blanks are supplied in place of valid values or characters. An optional end-of-file exit address may be supplied to the read-initiation routine to suppress this action. Acknowledgement of an end-of-file's record having been read must occur before initiating another read operation at the same unit. This is done by:

- Providing an end-of-file exit address to the read initiation routine,

- Writing, rewinding, or backspacing the dataset, or

- Calling the utility procedure IEOF.

Routine	$DFI
Function	Decode formatted initialize. Provides arguments for subsequent $DFA and $DFV calls.
Type of call	By address
Entry	(EP-1) = address of record length in characters (EP-2) = address of FORMAT specification (EP-3) = address of character string
Exit	No arguments returned

Routine	$DFA (type-checking entry points)
Function	Decode formatted, call by address. Decodes items in a packed character string, placing results into an array.
Type of call	By address
Entry	($DFA-1) = address of array ($DFA-2) = address of item count ($DFA-3) = address of item increment
Exit	Items are at user item addresses

Routine	$DFV (type-checking entry points)
Function	Decode formatted, call by value. Decodes a single item in a character string.
Type of call	By value
Entry	No arguments required
Exit	S1 contains the decoded item S2 contains the second word of the decoded item, if required

Routine $DFF

Function Decode formatted final. Terminates a decoding sequence.

Type of call No arguments required

Routine $EFI

Function Encode formatted initialize. Provides arguments for subsequent $EFA and $EFV calls.

Type of call By address

Entry (EP-1) = address of record length in characters

(EP-2) = address of FORMAT specification

(EP-3) = address of character string buffer

Exit Content of character string buffer

Routine $EFA (type-checking entry points)

Function Encode formatted, call by address. Encodes items in an array, placing results in the packed character string buffer.

Type of call By address

Entry ($EFA-1) = address of array

($EFA-2) = address of item count

($EFA-3) = address of item increment

Exit Content of character string buffer

Routine $EFV (type-checking entry points)

Function Encode formatted, call by value. Encodes a value and places the result in the character string buffer.

Type of call By value

Entry S1 contains the value to be encoded

S2 contains the second word of the value to be encoded, if required

Exit Content of character string buffer

Routine $EFF

Function Encode formatted final. Terminates an encoding sequence.

Type of call No arguments required

Routine $RFI

Function Read formatted initialize. Provides arguments for subsequent $RFA and $RFV calls

Type of call By address

Entry (EP-1) = address of unit name or number

(EP-2) = address of FORMAT specification

(EP-3) = address of error exit address (optional)

(EP-4) = address of end-of-file exit address (optional)

(EP-5) = address of status specifier (optional)

(EP-6) = address of record number for this transfer; present if and only if unit is connected for direct device

Exit Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

Routine $RFA (type-checking entry points)

Function Read formatted, call by address. Decodes and moves the number of items specified by (EP-2) to locations beginning at (EP-1) as incremented by (EP-3).

Type of call By address

Entry ($RFA-1) = Address of array

($RFA-2) = address of item count

($RFA-3) = address of array address increment

Exit Decoded items are at user item addresses

Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

Routine $RFV (type-checking entry points)

Function Read formatted, call by value. Decodes a single item.

Type of call By value

Entry No arguments required

Exit S1 contains the decoded item

S2 contains the second word of the decoded item, if required

Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

Routine $RFF (type-checking entry points)

Function Read formatted final. Terminates a read formatted sequence.

Type of call No arguments required

Exit Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

Routine $RUI

Function: Read unformatted initialize. Provides arguments for subsequent $RUA and $RUV calls.

Type of call: By address

Entry:

(EP-1) = address of unit name or number

(EP-3) = address of error exit address (optional)

(EP-4) = address of end-of-file exit address (optional)

(EP-5) = address of status specifier (optional)

(EP-6) = address of record number for this transfer; present if and only if unit is connected for direct device

Exit: Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

Routine $RUA (type-checking entry points)

Function: Read unformatted, call by address. Relocates the number of words specified by (EP-2) from the I/O buffer to locations beginning at (EP-1) as incremented by (EP-3).

Type of call: By address

Entry:

($RUA-1) = address of array

($RUA-2) = address of word count

($RUA-3) = address of array address increment

Exit: Requested words are in the array

Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

Routine $RUV (type-checking entry points)

Function Read unformatted, call by value. Moves a single value from the I/O buffer.

Type of call By value

Entry No arguments required

Exit S1 contains the requested word

S2 contains a second requested word, if required (for two-word values)

Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

Routine $RUF

Function Read unformatted final. Terminates a read unformatted sequence.

Type of call No arguments required

Exit Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

Routine $WFI

Function Write formatted initialize. Provides arguments for subsequent $WFA and $WFV calls.

Type of call By address

Entry (EP-1) = address of unit name or number

(EP-2) = address of FORMAT specification

(EP-3) = address of error exit address (optional)

(EP-4) = address of end-of-file exit address (optional)

(EP-5) = address of status specifier (optional)

(EP-6) = address of record number for this transfer; present if and only if unit is connected for direct device

Routine	$WFA (type-checking entry points)
Function	Write formatted, call by address. Encodes and moves to the I/O buffer the number of items specified by (EP-2) from locations beginning at (EP-1) as incremented by (EP-3).
Type of call	By address
Entry	($WFA-1) = address of array ($WFA-2) = address of item count ($WFA-3) = address of array address increment
Exit	Encoded items are in the I/O buffer Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

Routine	$WFV (type-checking entry points)
Function	Write formatted, call by value. Encodes and moves the word(s) provided into the I/O buffer.
Type of call	By value
Entry	S1 contains the word to be encoded and moved S2 contains a second word to be encoded and moved, if required
Exit	Encoded item is in the I/O buffer Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

Routine	$WFF
Function	Write formatted final. Terminates a write formatted sequence.
Type of call	No arguments required.
Exit	Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

Routine $WUI

Function
Write unformatted initialize. Provides arguments for subsequent $WUA and $WUV calls.

Type of call
By address

Entry
(EP-1) = address of unit name or number

(EP-3) = address of error exit address (optional)

(EP-4) = address of end-of-file exit address (optional)

(EP-5) = address of status specifier (optional)

(EP-6) = address of record number for this transfer; present if and only if this unit is connected for direct device

Exit
Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

Routine $WUA (type-checking entry points)

Function
Write unformatted, call by address. Transfers the number of words specified by (EP-2) from the locations beginning at (EP-1) as incremented by (EP-3).

Type of call
By address

Entry
($WUA-1) = address of array

($WUA-2) = address of word count

($WUA-3) = address of array increment

Exit
Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

Routine — $WUV (type-checking entry points)

Function — Write unformatted, call by value. Transfers the word(s) provided into the I/O buffer.

Type of call — By value

Entry — S1 contains the word to be transferred

S2 contains a second word to be transferred, if required (for two-word values)

Exit — Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

Routine — $WUF

Function — Write unformatted final. Terminates a write unformatted sequence.

Type of call — No arguments returned

Exit — Status specifier provided in initialization call; zero if no error, nonzero if error during initialization, transfer, or finalization

BUFFERED I/O ROUTINES

Buffered I/O routines perform operations on logical records. Control may be returned to the calling program before the I/O transfer is complete.

Routine $RB

Function Read buffered. Reads (EP-4) - (EP-3) + 1 words or until an end-of-record RCW is encountered, whichever is first, from the I/O buffer to the specified array locations. If (EP-2) < 0, a partial record may be read with a subsequent read capable of transferring all or part of the remaining words in the record. If (EP-2) ≥ 0, a subsequent read transfers words from the next record.

Type of call By address

Entry (EP-1) = address of unit name or number

(EP-2) = address of mode specifier

(EP-3) = address of first word of array

(EP-4) = address of last word of array

Exit No arguments returned

Routine	$WB
Function	Write buffered. Writes (EP-4) + (EP-3) + 1 words to the I/O buffer from locations (EP-3) through (EP-4) of the array. If (EP-2) < 0, a partial record may be written with a subsequent write capable of transferring all or part of the remaining words to the same record. If (EP-2) ≥ 0, a subsequent write transfers words to a new record. If (EP-4) is set to (EP-3) - 1, the partial record being written is terminated. Any attempt to write past the end of the allocated area or after encountering an end-of-data results in job abortion.
Type of call	By address
Entry	(EP-1) = address of unit name or number (EP-2) = address of mode specifier (EP-3) = address of first word of array (EP-4) = address of last word of array
Exit	No arguments returned

POSITIONING AND CONTROL I/O ROUTINES

The FORTRAN I/O routines descibed below perform dataset positioning and control operations:

Routine	$EOFW
Function	Write end-of-file. This function writes an end-of-file record on the specified dataset.
Type of call	By address
Entry	(EP-1) = address of unit name or number
Exit	No arguments returned

Routine $BACK

Function Backspace record. Positions the dataset to the start of the preceding record.

Type of call By address

Entry (EP-1) = address of unit name or number

Exit No arguments returned

Routine $REWF

Function Rewind function. Rewinds the specified dataset to the beginning-of-data point.

Type of call By address

Entry (EP-1) = address of unit name or number

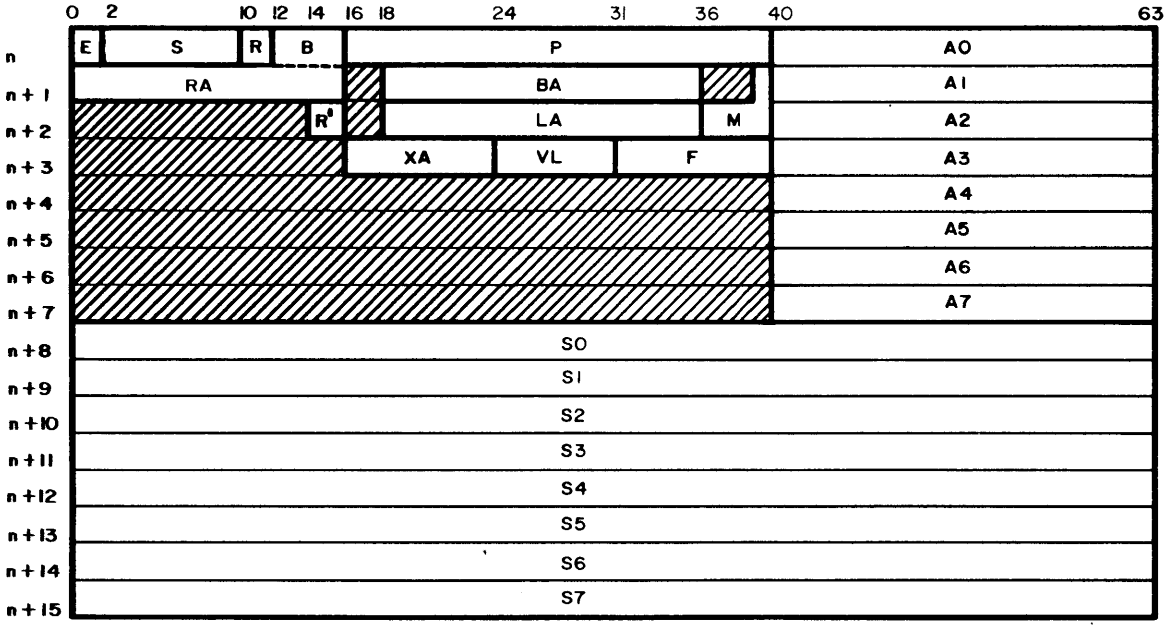
Exit No arguments returned

Routine $TRBK

Function Abort function. Makes the $FTLIB error procedure available to user programs. Returns to the error entrance to COS, not to the calling program.

Type of call No arguments required

EXCHANGE PACKAGE E



Registers

S	Syndrome bits
R'RAB	Read address for error (where B is bank)
P	Program address
BA	Base address
LA	Limit address
XA	Exchange address
VL	Vector length

E - Error type (bits 0,1 of n)

10	Uncorrectable memory
01	Correctable memory

R - Read mode (bits 10,11 of n)

00	Scalar
01	I/O
10	Vector
11	Fetch

M - Modes

n+1	39	Interrupt monitor mode†
n+2	36	Interrupt on correctable memory error
n+2	37	Interrupt on floating point error
n+2	38	Interrupt on uncorrectable memory error
n+2	39	Monitor mode

F - Flags

n+3	31	Programmable clock interrupt††
n+3	32	MCU interrupt
n+3	33	Floating point error
n+3	34	Operand range error
n+3	35	Program range error
n+3	36	Memory error
n+3	37	I/O interrupt
n+3	38	Error exit
n+3	39	Normal exit

† Supports Monitor Mode Interrupt option on CRAY-1A and CRAY-1B.

†† Supports Programmable Clock (optional on CRAY-1A and CRAY-1B; standard on CRAY-1 S Series computers)

ERROR AND STATUS CODES F

SYSTEM ERROR CODES

Table F-1 describes the system error codes as released. Installation differences can change data in this table. Consult the on-site analyst for details. The CRAY-OS Message Manual, publication SR-0039, also contains additional descripions of the abort codes and their corresponding messages.

Table F-1. Error codes for reprieve processing

System Error Code	Fatal/ Non-fatal	Reprieve Error Class (Octal Mask Value)	Description
AB001	NF	4	End-of-file on read
AB002	NF	4	Invalid LOCK or UNLOCK indicator
AB003	F	4	Device Allocation Table exhausted
AB004	NF	4	Dataset not open
AB005	NF	4	Invalid dataset open request
AB006	NF	4	No read permission
AB007	NF	4	No write permission
AB008	NF	4	Illegal bit set in RFL request word
AB009	NF	4	Attempt to delete memory outside program area
AB010	F	400	No available disk space
AB011	F	4000	System directory is full
AB012	NF	4	Job Table Area overflow
AB013	NF	4	More memory requested than available
AB014	NF	4	More memory requested than allowed
AB015	NF	2000	Unknown acquire error
AB016	NF	2000	Subdataset $IN cannot be disposed

Table F-1. Error codes for reprieve processing (continued)

System Error Code	Fatal/ Non-fatal	Reprieve Error Class (Octal Mask Value)	Description
AB017	NF	4	Invalid dataset close request
AB018	NF	4	Dataset already opened
AB019	NOT REPRIEVABLE		Job Communication Block destroyed
AB020	NF	4	Invalid system request parameter
AB021	NF	4	Dataset not found
AB022	NF	4	Invalid program load dataset
AB023	F	200	Job time limit exceeded
AB024	F	10	Operator dropped user job
AB025	NF	2	User program requested abort
AB026	NF	4	Invalid (undefined) user request
AB027	NF	4	Call not between user BA and LA
AB028[§]	NF		XP errors (no message)
AB029	NF	4	Logical device name not found
AB030	NF	4	Block number error
AB031	NF	4	Unrecoverable data error
AB032	NF	4	Unrecoverable hardware error
AB033	NF	4	Read after write or after EOD
AB034	NF	4	Unknown error
AB035	NF	4	Invalid processing direction
AB036	NF	4	Dataset prematurely terminated
AB037	NF	4	Dataset Parameter Table invalid
AB038	NOT REPRIEVABLE		Operator killed user job
AB039	NF	20	Operator reran the job
AB040	NF	4	Invalid disposition code
AB041	F	4000	"Enter" allowed on access only

[§] The AB028 error code is set during abort processing when any Exchange Package error flag is set. It does not represent a single reprievable condition. One of the Exchange Package error codes (AB053 through AB058) will be set later to indicate the appropriate error.

Table F-1. Error codes for reprieve processing (continued)

System Error Code	Fatal/ Non-fatal	Reprieve Error Class (Octal Mask Value)	Description
AB043	F	400	Allowable user log size exceeded
AB044	NF	4	Invalid dataset name
AB045	NF	400	Specified LM is too big
AB046	NF	400	Dataset size limit exceeded
AB047	NF	2000	Dataset not available from station
AB048	NF	2000	Dataset cannot be saved on a front end
AB049	NF	4	Invalid LFTs in user area
AB051	F	4	Invalid pointer to first JTA LFT
AB052	NF	4	No user LFT DN matches JTA LFT
AB053	NF	100	Floating-point error
AB054	NF	4	Operand range error
AB055	NF	4	Program range error
AB056	NF	40	Uncorrected memory error
AB057	NOT REPRIEVABLE		Interactive ABORT
AB058	F	4	Error exit
AB061	NF	4	No invoke request provided
AB062	NF	4	Invoke request abort pending
AB063	NF	4	Invoke length not multiple of 512
AB064	NF	4	Invoke length greater than maximum
AB066	NF	4	Dataset has related disposes active
AB067	NF	4	Invalid procedure dataset
AB068	NF	4	Procedure nest level exceeded
AB070	NF	10000	An ATTENTION request command was entered at an interactive terminal.
AB071	NF	4	Bad class structure
AB072	NF	4	DSP destroyed by user
AB073	NF	4	Undefined function code in F$INS

Table F-1. Error codes for reprieve processing (continued)

System Error Code	Fatal/ Non-fatal	Reprieve Error Class (Octal Mask Value)	Description
AB074	NF	4000	DUMPJOB processing has been inhibited
AB075	NF	4000	No permissions granted while dataset is execute-only
AB076	NF	4	Dataset is already accessed by the job
AB077	NOT REPRIEVABLE		CSP internal error
AB078	NF	4000	Privileged system request
AB079	NF	4	Unassigned JCL symbol
AB080	NF	4	Receive buffer too small
AB081	NF	4	Undefined JCL symbol
AB082	NF	4	JCL symbol cannot be modified
AB083	NF	4	Invalid message class

PERMANENT DATASET STATUS CODES

The permanent dataset status octal codes are flagged in the PMST field of the Permanent Dataset Definition Table (PDD) which is presented in Appendix A.

PMST	Status
1	Complete; no error
11	A DNT cannot be found for the specified dataset.
21	Maintenance permission not granted
31	Edition already exists
41	DSC full
51	Function code out of range

PMST	Status
61	The job has a dataset of the local name (DN) specified.
71	No permission granted
101	Delay and try again
111	DSC does not contain the requested dataset.
121	Edition does not exist
131	PDS full
141	Dataset not permanent
151	PDS entry not found
161	Continuation error
171	DAT full
201	DNT full
211	End of DSC
221	PDN already accessed by this job
231	Request to read zero pages
241	Invalid page number requested
251	No data has been written to disk
261	SDT does not exist
271	SDT not on input or output queue
301	Unable to queue SDT
311	Dataset name in PDD is 0
331	Multiple editions of the dataset exist, prohibiting changes to the permission control words.
341	Unique access is not acceptable because the dataset is part of the System Directory.
351	The PDD contains a text length without a text address, or a text address without a length specified.
361	The text length specified exceeds the allowable maximum.
371	The device on which all or part of the dataset resides is down
421	Access is denied because crossed allocation unit exists.
441	The DSC entry was flagged by Startup as containing a fatal error. Access is denied.

PMST	Status
461	No available QDT entries exist to coordinate the dispose.
471	The dataset has outstanding disposes; do not deallocate disk space.
501	Allocation of multitype dataset inconsistent with related datasets
511	Multitype dataset has non-existent QDT entry.
521	Maximum edition reached

GLOSSARY

GLOSSARY

A

Abort - To terminate a program or job when a condition (hardware or software) exists from which the program or computer cannot recover.

Absolute address - (1) An address that is permanently assigned by the machine designator to a storage location. (2) A pattern of characters that identifies a unique storage location without further modification. Synonymous with machine address.

Absolute block - Loader tables consisting of the image of a program in memory. It can be saved on a dataset for subsequent reloading and execution.

Address - (1) An identification, as represented by a name, label, or number, for a register, location in storage, or any other data source or destination such as the location of a station in a communication network. (2) Any part of an instruction that specifies the location of an operand for the instruction.

Allocate - To reserve an amount of some resource in a computing system for a specific purpose (usually refers to a data storage medium).

Alphabetic - A character set including, $, %, @, as well as the 26 uppercase letters A through Z.

Alphanumeric - A character set including all alphabetic characters and the digits 0 through 9.

Arithmetic operator - Part of an expression that indicates action to be performed during evaluation of expression; can be symbolic character representing addition, unary plus, subtraction, unary minus, multiplication, or division.

Assemble - To prepare an object language program from a symbolic language program by substituting machine operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic instructions.

B

Base address - The starting absolute address of the memory field length assigned to the user's job. This address is maintained in the base address (BA) register. The base address must be a multiple of 20_8.

$BLD - A dataset on which load modules are placed by a compiler or assembler unless the user designates some other dataset.

Blank common block - A common block into which data cannot be stored at load time. The first declaration need not be the largest. The blank common block is allocated after all other blocks have been processed.

Block - (1) A tape block is a collection of characters written or read as a unit. Blocks are separated by an interblock gap and may be from 1 through 1,048,576 bytes. A tape block and a physical record are synonymous on magnetic tape. (2) In CRAY-1 blocked format, a block is a fixed number of contiguous characters preceded by a block control word as the first word of the block. The internal block size for the CRAY-1 is 512 words (one sector on disk). In CRAY-1 manuals, the terms tape block and 512-word block are consistently used to distinguish between the two uses.

Block control word - A word occurring at the beginning of each block in the CRAY-1 blocked format that identifies the sequential position of the block in the dataset and points forward to the next block control word.

BOT - Beginning of tape; the position of the beginning-of-tape reflective marker.

BOV - Beginning of volume. See BOT.

BPI - Bits per inch. COS supports the 1600 and 6250 bpi recording densities.

Buffer - A storage device used to compensate for the difference in rate of flow of data, or time of occurrence of events, when transmitting data from one device to another. It is normally a block of memory used by the system to transmit data from one place to another. Buffers are usually associated with the I/O system.

Buffer Memory - A 64-bit memory in the I/O Subsystem common to all I/O Processors.

C

Call - The transfer of control to a specified closed routine.

Card image - A one-to-one representation of the contents of a punched card, for example, a matrix in which a 1 represents a punch and a 0 represents the absence of a punch. In CRAY-1 blocked format, each card image is a record.

Catalog (noun) - A list or table of items with descriptive data, usually arranged so that a specific kind of information can be readily located.

Channel - A path along which signals can be sent.

Character - A logical unit composed of bits representing alphabetic, numeric, and special symbols. The CRAY-1 software processes 8-bit characters in the ASCII character set.

Code - (1) A system of character and rules representing information in a form understandable by a computer. (2) Translation of a problem into a computer language.

Common block - A block that can be declared by more than one program module during a load operation. More than one program module can specify data for a common block but if a conflict occurs, information from later programs is loaded over previously loaded information. A program may declare no common blocks or as many as 125 common blocks. The two types of common blocks are labeled and blank.

Conditional control statement block - Defines the conditions under which a group of control statements are to be processed. The statements which define the block and conditions are: IF, ELSE, ELSEIF, and ENDIF.

Control statement - The format, consisting of a verb and its parameters, used to control the operating system and access its products. Directives are used to control products.

Control statement input file - A dataset containing valid control statements as its first file.

$CS - A primary control statement input file.

D

Data - (1) Information manipulated by or produced by a computer program. (2) Empirical numerical values and numerical constants used in arithmetic calculation. Data is considered to be that which is transformed by a process to produce the evidence of work. Parameters, device input, and working storage are considered data.

Dataset - A quantity of information maintained on mass storage by the CRAY-1 Operating System. Each dataset is identified by a symbolic name called a dataset name. Datasets are of two types: temporary and permanent. A temporary dataset is available only to the job that created it. A permanent dataset is available to the system and to other jobs and is maintained across system deadstarts.

Dataset name verb - A verb that is the name of a dataset. See local or system dataset name verb.

Deadstart - The process by which an inactive machine is brought up to an operational condition ready to process jobs.

Debug - To detect, locate, and remove mistakes from a routine or malfunction of a computer. Synonymous with troubleshoot.

Delimiter - A character that separates items in a control statement or a directive; synonymous with separator.

Density - See tape density.

Device - A piece of equipment that mechanically contains and drives a recording medium.

Directive - A command used to control a product, such as UPDATE.

Diagnostic - (1) Pertaining to the detection and isolation of a malfunction or a mistake. (2) A message printed when an assembler or compiler detects a program error.

Disposition code - A code used in I/O processing to indicate the disposition to be made of a dataset when its corresponding job is terminated or the dataset is released.

Dump - (1) To copy the contents of all or part of a storage device, usually from internal storage, at a given instant of time. (2) The process of performing (1). (3) The document resulting from (1).

E

End-of-data delimiter - Indicates the end of a dataset. In CRAY-1 blocked format, this is a record control word with a 17_8 in the mode field.

End-of-file delimiter - Indicates the end of a file. (1) In CRAY-1 blocked format, this is a record control word with a 16_8 in the mode field. (2) On magnetic tape, this is a tapemark.

End-of-record delimiter - Indicates the end of a record. (1) In CRAY-1 blocked format, this is a record control word with a 10_8 in the mode field. (2) In an ASCII punched deck, this is indicated by the end of each card.

Entry point - A location within a block that can be referenced from program blocks that do not declare the block. Each entry point has a unique name associated with it. The loader is given a list of entry points in a loader table. A block can contain any number of entry points.

An entry point name must be 1 to 8 characters and cannot contain the characters blank, asterisk, or slash. Some language processors (i.e., FORTRAN) may produce entry point names under more restricted formats due to their own requirements.

EOD - End-of-data on tape. The definition of EOD is a function of whether the tape is labeled or nonlabeled and of the type of operation being performed (input or output). When reading a labeled tape, EOD is returned to the user when an EOF1 trailer label is encountered. When reading a nonlabeled tape, EOD is returned when a tapemark is read on the last volume in the volume list for a particular dataset. When writing a labeled or nonlabeled tape, EOD processing is initiated by a write EOD, rewind, close, or release request.

EOI - End-of-information; see EOD.

EOT - End-of-tape; a status, set only on a write operation indicating sensing of the end of the tape reflective marker.

EOV - End-of-volume. On output, EOV occurs when end-of-tape status is returned on a write operation. This status occurs when the EOT reflective marker is sensed by the tape device. For input of a labeled tape dataset, EOV occurs when an EOV1 trailer label is read; for input of a nonlabeled dataset, EOV is returned when a tapemark is encountered and the volume list is not exhausted.

Exchange package - A 16-word block of data in memory which is associated with a particular computer program or memory field. It contains the basic parameters necessary to provide continuity from one execution interval for the program to the next.

Expression (JCL parameter expression) - A series of characters grouped into operands and operators which are computed as one value during parameter evaluation; should be delimited by parentheses.

External reference - A reference in one program block to an entry point in a block not declared by that program. Throughout the loading process, externals are matched to entry points (this is also referred to as satisfying externals); that is, addresses referencing externals are supplied with the correct address.

F

File - A collection of records in a dataset. In CRAY-1 blocked format, a file is terminated by a record control word with 16_8 in the mode field.

Filemark - Refer to tapemark.

Foreign label - A special condition that can occur during the label scan at the beginning of a tape. If a NOT CAPABLE status is returned on a BOV label scan, TQM declares the tape to be foreign labeled (FRN) which protects a 7-track tape or a 9-track, 800 bpi tape from being accidently destroyed.

Formal parameter specifications - Parameters in a procedure definition which identify the character strings within the procedure body that can be substituted during the procedure's evaluation.

Front-end processor - A computer connected to a CRAY-1 channel. The front-end processor supplies data and jobs to the CRAY-1 and processes or distributes the output from the jobs. Front end systems are also referred to as stations in Cray publications.

G

Generic name - Tape resource requirements are expressed using generic names or installation-defined synonyms. A generic name corresponds to a device type. COS supports up to 16 generic names[§]. A generic name may be represented by a synonym.

H

HLM - High-level memory, the user's program and data area in memory.

[§] Deferred implementation

I

$IN - A dataset containing the job control language statements as well as the source input and data for compilers and assemblers, unless the user designates some other dataset (FT05 for example).

In-line procedure - A procedure defined in a control statement file.

Input/Output - (1) Commonly called I/O. To communicate from external equipment to the computer and vice versa. (2) The data involved in such a communication. (3) Equipment used to communicate with a computer. (4) The media carrying the data for input/output.

Integer constant - Specifies an octal value or a decimal value that can be signed as positive or negative.

Interchange format - One of the two ways in which tape datasets can be read or written. Each tape block of data corresponds to a single logical record in COS blocked format. Interchange format is selected by setting DF=IC when a tape dataset is accessed. As far as I/O routines in the CRAY-1 mainframe are concerned, interchange datasets must be in CRAY blocked format because the CRAY blocked structure (BCW's and RCW's) is used to describe each tape block read or written. This blocked structure allows the user to write or read variable-length tape blocks at high speed with data resolution to the 88-bit byte level of the tape device. The record control word (RCW) is used to define the tape block length on output and to describe the block length on input. No BCW or RCW ever appears in the data written on the tape.

Interblock gaps - The physical separation between successive tape blocks on magnetic tape.

I/O Subsystem - Part of a CRAY-1 S Series Model S/1200 through S/4400 consisting of two to four I/O processors and one-half, one, four, or eight million words of shared Buffer Memory. The optional tape subsystem is composed of at least one block multiplexer channel, one tape controller, and two tape units. The tape units supported are IBM-compatible 9-track, 200 ips, 1600/6250 bpi devices.

Iterative control statement block - Defines the repeated execution of a series of statements if a condition is satisfied

J

JCL block control statement - A statement in the control statement file that is part of a group of control statements called a block which specifies an action to be taken by COS; the three types of blocks are: procedure defintion, conditional, and iterative.

Job - (1) An arbitrarily defined parcel of work submitted to a computing system. (2) A collection of tasks submitted to the system and treated by the system as an entity. A job is presented to the system as a formatted dataset. With respect to a job, the system is parametrically controlled by the content of the job dataset.

Job Communication Block - The first 200_8 words of the job memory field. This area is used to hold the current control statement and certain job-related parameters. The area is accessible to the user, the operating system, and the loader for inter-phase job communication.

Job control statement - Any of the statements used to direct the operating system in its functioning, as compared to data, programs, or other information needed to process a job but not intended directly for the operating system, itself. A control statement may be expressed in card, card image, or user terminal keyboard entry medium.

Job deck - The physical representation of a job before processing either as a deck of cards or as a group of records. The first file of the job dataset contains the job statements and the job parameters which will be used to control the job. Following files contain the program and data which the job will require for the various job control statements. The job deck is terminated by an end-of-data delimiter.

Job input dataset - A dataset named $IN on which the card images of the job deck are maintained. This consists of programs and data referenced by various job steps. The user can manipulate the dataset like any othr dataset (excluding write operations).

Job output dataset - Any of a set of datasets recognized by the system by a special dataset name (e.g., $OUT, $PLOT, and $PUNCH), which becomes a system permanent dataset at job end and is automatically staged to a front-end computer for processing.

Job step - A unit of work within a job, such as source language compilation or object program execution.

K

Keyword parameter - A string of 1 to 8 alphanumeric characters that consists of a keyword followed by one or more values; identified by its form rather than by its position in the control statement.

L

$LOG - See logfile.

Labeled common - A common block into which data can be stored at load time.

Library - A dataset composed of sequentially organized records and files. The last file of the library contains a library directory. The rest of the files and records, known as entries, can consist of processed procedure definitions and/or relocatable modules. The directory gives a listing of entry names with their associated characteristics.

Library-defined verb - A one through eight character name of a program or procedure definition residing in a library that is a part of the current library searchlist.

Limit address - The upper address of a memory field. This address is maintained in the limit address (LA) register.

Literal - A symbol which names, describes, or defines itself and not something else that it might represent.

Literal constant - A string of one through eight characters delimited with apostrophes whose ordinal numbers are in the range 040_8 through 176_8; value of a character constant corresponds to the ASCII character codes positioned within a 64-bit word; alignment indicated can be left or right adjusted and zero-filled or left-adjusted and space-filled; apostrophes remain as part of value.

Literal string - A string delimited with apostrophes which are normally not treated as part of the value, except with JCL block control statements which treat the apostrophes as part of the string value.

Loader tables - The form in which code is presented to the loader. Loader tables are generated by compilers and assemblers according to loader requirements. The tables contain information required for loading such as type of code, names, types and lengths of storage blocks, data to be stored, etc.

Loading - The placement of instructions and data into memory so that it is ready for execution. Loader input is obtained from one or more datasets and/or libraries. Upon completion of loading, execution of the program in the job's memory field is optionally initiated. Loading may also involve the performance of load-related services such as generation 9f a loader map, presetting of unused memory to a user- specified value, and generation of overlays.

Load point - See BOT.

Local dataset - A temporary or permanent dataset accessible by the user.

Local dataset name verb - A verb that is the name of a local dataset consisting of an alphabetic character followed by one through six alphanumeric characters. Requests that COS load and execute an absolute binary program from the first record of the named dataset.

Logfile - During the processing of the job, a special dataset named $LOG is maintained. At job termination, this dataset is appended to the $OUT file for the job. The job logfile serves as a time-ordered record of the activities of the job -- all control statements processed by the job, significant information such as dataset usage, all operator interactions with a job, and errors detected during processing of the job.

Logical operator - Represents logical function performed on operands on a bit-by-bit basis, returning a 64-bit result; functions are: inclusive OR, intersection, exclusive OR, unary complement.

M

Macro instruction - An instruction in a source language that is equivalent to a specified sequence of machine instructions.

Magnetic tape - A tape with a magnetic surface on which data can be stored by selective polarization of portions of that surface.

Mainframe - The central processor of the computer system. It contains the arithmetic unit and special register groups. It does not include input, output, or peripheral units and usually does not include internal storage. Synonymous with central processing unit (CPU).

Mass storage - The storage of a large amount of data that is also readily accessible to the central processing unit of a computer.

Memory field - A portion of memory containing instructions and data usually defined for a specific job. Field limits are defined by the base address and the limit address. A program in the memory field cannot execute outside of the field nor refer to operands outside of the field.

Multiprocessing - Utilization of several computers to logically or functionally divide jobs or processors, and to execute various programs or segments asynchronously and simultaneously.

Multiprogramming - A technique for handling multiple routines or programs simultaneously by overlapping or interleaving their execution, that is, permitting more than one program to time-share machine components.

N

Nesting - Including a block of statements of one kind into a larger block of statements of the same kind, such as an iterative block within a larger iterative block.

Not Capable - A tape status indicating the reel currently mounted cannot be read by the control unit and drive. The Not Capable status would be returned if an 800 bpi tape were mounted on a device that supported only 1600 and 6250 bpi, for example. Since it is not possible to read a Not Capable tape to verify label type and contents, COS rejects (unloads) all tapes that return a Not Capable status.

O

$OUT - A dataset that contains the list output from compilers and assemblers unless the user designates some other dataset. At job end, the job logfile is added to the $OUT dataset and the dataset is sent to a front-end computer.

Operand - A character string in an expression that is operated on during evaluation; types are integer constant, literal constant, symbolic variable, and subexpresion.

Operating system - (1) The executive, monitor, utility, and any other routines necessary for the performance of a computer system. (2) A resident executive program that automates certain aspects of machine operation, particularly as they relate to initiating and controlling the processing of jobs.

Operator - A symbolic representation indicating the action to be performed in an expression; types are arithmetic, relational, and logical operators.

Overlaying - A technique for bringing routines into memory from some other form of storage during processing so that several routines will occupy the same storage locations at different times. Overlaying is used when the total memory requirements for instructions exceeds the available memory.

P

$PROC - A dataset to which in-line procedure definitions are written.

Parameter - A quantity in a control statement which may be given different values when the control statement is used for a specific purpose or process.

Parcel - A 16-bit portion of a word which is addressable for instruction execution but not for operand references. An instruction occupies one or two parcels; if it occupies two parcels, they may be in separate words.

Parenthetic string - A string delimited with parentheses instead of apostrophes; parentheses are treated as part of the string when evaluted except when preceded by an initial, parameter, equivalence, or concatenation separator character.

Permanent dataset - A dataset known to the operating system as being permanent; the dataset survives deadstart.

Positional parameter - A parameter that must appear in a precise position relative to the separators in the control statement.

Procedure - A named sequence of control statements and/or data that is saved in a library for processing at a later time when activated by a call to its name by a calling statement; provides the capability of replacing values within the procedure with other values.

Procedure defintion - The definition of a procedure that is saved in a library to be called for processing at a later time; if defined in a job control statement is called an in-line procedure definition.

Program - (1) A sequence of coded instructions that solves a problem. (2) To plan the procedures for solving a problem. This may involve analyzing the problem, preparing a flow diagram, providing details, developing and testing subroutines, allocating storage, specifying I/O formats, and incorporating a computer run into a complete data processing system.

Program block - The block within a load module usually containing executable code. It is automatically declared for each program (though it may be zero-length). It is local to the module; that is, it can be accessed from other load modules only through use of external symbols. Data placed in a program block always comes from its own load module.

Program name - Also referred to as IDENT name or deck name, the name contained in the loader PDT table at the beginning of each load module.

Program library - (PL) The base dataset used by the UPDATE utility. This dataset consists of one or more specially formatted card image *decks*, each separated by an end-of-file.

R

Record - A group of contiguous words or characters related to each other by virtue of convention. A record may be fixed or variable length. (1) In CRAY-1 blocked format, a record ends with a record control word with 10_8 in the mode field. (2) In an ASCII coded punched deck, each card is a record. (3) For a listable dataset, each line is a rcord. (4) For a binary load dataset, each module is a record.

Relational operator - An oprator that indicates the comparison to be performed between the operands in an expression (-1 for a TRUE result and 0 for a FALSE result); types are equal, not equal, less than, greater than, less than or equal, and greater than or equal.

Relative address - An address defined by its relationship to a base address (e.g., the (BA)) such that the base address has a relative address of 0.

Relocatable address - An address presented to the loader in such a form that it can be loaded anywhere in the memory field. A relocatable address is defined as being relative to the beginning address of a load module program block or common block.

Relocatable module - This is the basic program unit produced by a compiler or assembler. CAL produces a relocatable module from source statements delineated by IDENT and END. In FORTRAN, the corresponding beginning statements are PROGRAM, SUBROUTINE, BLOCK DATA, or FUNCTION. The corresponding end statement is END.

A relocatable module consists of several loader tables that define blocks, their contents, and address relocation information.

Relocate - In programming, to move a routine from one portion of internal storage to another and to adjust the necessary address references so that the routine can be executed in its new location. Instruction addresses are modified relative to a fixed point or origin. If the instruction is modified using an address below the reference point, relocation is negative. If addresses are above the reference point, relocation is positive. Generally, a program is loaded using positive relocation.

S

Sector - A physical area on disk equivalent to 512 CRAY-1 words. In CRAY-1 blocked format, a block is also 512 contiguous words with a block control word as the first word of the block. Therefore the internal block size for the CRAY-1 is equivalent to one CRAY-1 disk sector. This is the unit of data transfer between the CRAY-1 mainframe and the I/O Subsystem.

Separator - Synonym for delimiter.

String - A sequence of characters delimited by apostrophes or parentheses which is to be taken literally as a parameter value; see literal string and parenthetic string.

Subexpression - An expression that is evaluated so that its result becomes an operand.

Substitution parameters - Parameters on procedure definition prototype statement or procedure calling statement which provide replacement values to be substituted during evaluation for strings flagged within the procedure body.

Symbolic variable - A string of one to eight alphanumeric characters, beginning with an alpha character that represents values maintained by COS and/or the user.

System dataset name verb - A verb that is the name of a system-defined dataset in the System Directory Table (SDR); consists of an alphabetic character which can be followed by one through six alphanumeric characters.

System logfile - A permanent dataset named $SYSTEMLOG.

System verb - Requests that COS perform a function; consists of an alphabetic character which can be followed by one through six alphanumeric characters

T

Table - A collection of data, each item being uniquely identified either by some label or by its relative position.

Tape block - A group of contiguous characters recorded on and read from magnetic tape as a unit.

Tape control unit - A piece of equipment connected to a block multiplexer channel that provides the capability for controlling the operation of one or more tape devices. Up to four control units may be combined to drive a maximum of 16 tape devices. The control units are cross connected to all devices. Such a configuration is called a 4x16 (four by sixteen). If one control unit were to be connected to three devices, it would be referred to as a 1x3 configuration.

Tape density (bpi) - The number of bits per inch on magnetic tape. COS supports 6250 bpi and 1600 bpi.

Tape format - The way tape datasets are read or written. In *interchange format*, each tape block of data corresponds to a single logical record in COS blocked format. In *transparent format*, each tape block is a fixed multiple of 512 words based on the density of the tape.

Tape volume - A reel of magnetic tape.

Tapemark - A special hardware bit configuration recorded on magnetic tape. It indicates the boundary between combinations of datasets and labels. It is sometimes called a filemark.

TDT - Tape Device Table entry. Contains one entry for each device in the configuration. A TDT entry is used to control the activity associated with a tape device and contains the six-word packet through which requests to the I/O Subsystem are made.

Temporary dataset - A dataset which is not permanent and is available only to the job that created it.

Time slice - The maximum amount of time during which the CPU can be assigned to a job without re-evaluation as to which job should have the CPU next.

Transparent format - One of two ways tape datasets are read or written. Each tape block is a fixed multiple of 512 words. Transparent format is the default tape dataset format and is designated by setting DF=TR when accessing a tape dataset. This format produces a fixed-length block dataset (16384 bytes at 1600 bpi or 32768 bytes at 6250 bpi) that may be a CRAY blocked or unblocked dataset as far as any I/O routines are concerned. The tape subsytem merely takes four (1600 bpi) or eight (6250 bpi) sectors and processes them as one physical tape block. When a short block is read, it is considered to be EOD.

U

Unit record device - A device such as a card reader, printer, or card punch for which each unit of data to be processed is considered a record.

Unload - To remove a tape from ready status by rewinding beyond the load point. The tape is then no longer under control of the computer.

Unsatisfied external - An external reference for which the loader has not yet loaded a module containing the matching entry point.

User logfile - A dataset named $LOG created for a job when it is initiated by the Job Scheduler.

V

Verb - The first nonblank field of a control statement; specifies the action to be taken by COS during control statement evaluation.

Volume - A dismountable physical unit of storage media, for example, a reel of magnetic tape.

Volume identifier - Up to six alphanumeric characters used to identify a physical reel of tape. On labeled tapes, the volume identifier is actually recorded on tape in the volume header label. Volume identifier is synonomous with volume serial number.

VSN - Volume serial number. See volume identifier.

W

Word - A group of bits between boundaries imposed by the computer. Word size must be considered in the implementation of logical divisions such as character. The word size of the CRAY-1 is 64-bits.

INDEX

INDEX

COS
CONTROL STATEMENT
AND
MACRO SUMMARY

COS CONTROL STATEMENT AND MACRO SUMMARY

This section summarizes all of the control statements in the COS job control language and the macros included with COS.

Control statement section

A parameter shown in all UPPERCASE letters must be coded literally as shown, while a value must be substituted for an *italicized* item. For certain parameters, all possible values that can be taken are listed in braces; in these cases, the default value, if one exists, is underlined.

In the left margin is a reference to the location of additional information on each control statement. In most cases, this reference is to a page number in Part 2 of this manual. However, for those control statements that are documented in other Cray Research publications, the reference is to the corresponding publication number.

Macro instruction section

Detailed descriptions of each macro instruction provided with COS are given in part 3 of this manual. Included here is a list of these macros with references to their locations in part 3.

CONTROL STATEMENTS

pg./pub.	Control statement
1-7	* *comment text*
2-4	&DATA,*dn*.
4-4	ACCESS,DN=*dn*,PDN=*pdn*,ID=*uid*,ED=*ed*,R=*rd*,W=*wt*,M=*mn*,UQ,LE,NA,CS=*cs*,DF=*df*,DT=*dt*,FSEC=*fsec*,LB=*lb*,MBS=*mbs*,NEW, XDT=*yyddd*,RT=*rt*,VOL=vol_1:vol_2:...vol_n,CT=*ct*,RF=*rf*,RS=*rs*.
1-10	ACCOUNT,AC=*ac*,PW=*pw*.
5-1	ACQUIRE,DN=*dn*,PDN=*pdn*,ID=*uid*,ED=*ed*,RT=*rt*,R=*rd*,W=*wt*,M=*mn*,UQ,TEXT=*text*,MF=*mf*,TID=*tid*,DF=*df*.
4-9	ADJUST,DN=*dn*,NA.
SM-0036	APML,I=*idn*,L=*ldn*,B=*bdn*,E=*edn*,ABORT,DEBUG,*options*,LIST=*name*,S=*sdn*,SYM=*sym*,T=*bst*,X=*xdn*.
3-1	ASSIGN,DN=*dn*,S=*size*,BS=*blk*,DV=*ldv*,DT=*dt*,DF=*df*,RDM,U,MR,LM=*lm*,DC=*dc*,BFI=*bfi*,A=*un*.
7-6	AUDIT,L=*ldn*,B=*bdn*,PDN=*pdn*,ID=*uid*,PREFIX=*pfx*,DV=*dvn*,SZ=*dsz*,X=*mm/dd/yy*:'*hh:mm:ss*',TCR=*mm/dd/yy*:'*hh:mm:ss*', TLA=*mm/dd/yy*:'*hh:mm:ss*',TLM=*mm/dd/yy*:'*hh:mm:ss*'.
11-3	BUILD,I=*idn*,L=*ldn*,OBL=*odn*,B=*bdn*,NBL=*ndn*,SORT,NODIR,REPLACE.
SR-0000	CAL,I=*idn*,L=*ldn*,B=*bdn*,E=*edn*,ABORT,DEBUG,*options*,LIST=*name*,S=*sdn*,SYM=*sym*,T=*bst*,X=*xdn*.
1-8	CALL,DN=*dn*,CNS.
SR-0009	CFT,I=*idn*,L=*ldn*,B=*bdn*,C=*cdn*,E=*n*,ON=*string*,OFF=*string*,TRUNC=*nn*,AIDS=*aids*,OPT=*option*.
1-11	CHARGES,SR=*options*.
8-11	COMPARE,A=*adn*,B=*bdn*,L=*ldn*,DF=*df*,ME=*maxe*,CP=*cpn*,CS=*csn*,CW=cw_1[:cw_2],ABORT=*ac*.
6-3	COPYD,I=*idn*,O=*odn*.
6-2	COPYF,I=*idn*,O=*odn*,NF=*n*.
6-1	COPYR,I=*idn*,O=*odn*,NR=*n*.
SM-0050	CSIM,L=*ldn*,I=*idn*,LINES=*n*,T=*tl*,MAXBK=*ms*,VMEM=*vm*.
8-6	DEBUG,I=*idn*,O=*odn*,DUMP=*ddn*,TRACE=*n*,SYMS=*sym*,NOTSYMS=*nsym*,MAXDIM=*dim*,BLOCKS=*blk*,NOTBLKS=*nblk*,PAGES=*np*,COMMENT='*string*'.
4-11	DELETE,DN=*dn*,NA.
6-4	DISPOSE,DN=*dn*,SDN=*sdn*,DC=*dc*,DF=*df*,MF=*mf*,SF=*sf*,ID=*uid*,TID=*tid*,ED=*ed*,RT=*rt*,R=*rd*,W=*wt*,M=*mn*,TEXT=*text*, WAIT,NOWAIT,DEFER,NRLS.
8-8	DSDUMP,I=*idn*,O=*odn*,DF=*df*,IW=*n*,NW=*n*,IR=*n*,NR=*n*,IF=*n*,NF=*n*,IS=*n*,NS=*n*.
8-2	DUMP,I=*idn*,O=*odn*,FW=*fwa*,LW=*lwa*,JTA,NXP,V,DSP,FORMAT=*f*,CENTER.
8-1	DUMPJOB.
1-13	ECHO,ON=$class_1$:$class_2$:...:$class_i$,OFF=$class_1$:$class_2$:...:$class_i$.
2-16	ELSE.
2-18	ELSEIF(*expression*)
2-14	ENDIF.
2-22	ENDLOOP.
2-5	ENDPROC.
1-5	EXIT.
2-22	EXITLOOP. or EXITLOOP(*expression*)
8-14	FLODUMP.
2-14	IF(*expression*)
1-8	IOAREA $\left\{\begin{matrix}\text{,LOCK}\\ \text{UNLOCK}\end{matrix}\right\}$.
1-3	JOB,JN=*jn*,M=*fl*,T=*tl*,P=*p*,US=*us*,OLM=*olm*,CL=*jcn*,*gn=nr.
9-1	LDR,DN=*dn*,LIB=*ldn*,NOLIB=*ldn*,LLD,AB=*adn*,MAP=*op*,SID='*string*',T=*tra*,NX,DEB=*l*,C,OVL=*dir*,CNS,NA,USA,L=*ldn*,SET=*val*,E=*n*,I=*sdir*.

pg./pub.	Control statement
1-14	LIBRARY,DN=dn_1:dn_2:...:dn_{64},V.
2-21	LOOP.
1-5	MODE,M=*mode*.
4-9	MODIFY,DN=*dn*,PDN=*pdn*,ID=*uid*,ED=*ed*,RT=*rt*,R=*rd*,W=*wt*,W=*wt*,M=*mn*,NA,EXO=$\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$.
1-7	NORERUN $\left\{\begin{matrix}\text{,ENABLE}\\\text{,DISABLE}\end{matrix}\right\}$.
1-15	OPTION,LPP=*n*,STAT=$\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$.
7-1	PDSDUMP,DN=*dn*,DV=*ldv*,PDS=*pds*,CW=*cw*, ID=*uid*,US=*usn*,ED=*ed*,X,C,D,I,O,S.
7-4	PDSLOAD,DN=*dn*,PDS=*pds*,CW=*cw*, ID=*uid*,US=*usn*,ED=*ed*,DV=*dvn*,A,I,O,S,NA.
8-13	PRINT(*expression*)
2-3	PROC.
3-4	RELEASE,DN=dn_1:dn_2:...:dn_8,HOLD.§
1-7	RERUN $\left\{\begin{matrix}\text{,ENABLE}\\\text{,DISABLE}\end{matrix}\right\}$.
1-9	RETURN,ABORT.
6-5	REWIND,DN=dn_1:dn_2:...:dn_8.
1-6	RFL,M=*fl*.
1-12	ROLLJOB.
4-12	SAVE,DN=*dn*,PDN=*pdn*,ID=*uid*,ED=*ed*,RT=*rt*,R=*rd*,W=*wt*,M=*mn*,UQ,NA,EXO=$\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$.
1-12	SET(*symbol*=*expression*)
6-5	SKIPD,DN=*dn*.
6-4	SKIPF,DN=*dn*,NF=*n*.
6-3	SKIPR,DN=*dn*,NR=*n*.
SR-0033	SKOL,I=*idn*,L=*ldn*,X=*xdn*,E=*edn*,O=*odn*,M=*mdn*,VIEW,LISTOFF.
5-9	SUBMIT, DN=*dn*,SID=*mf*,DID=*mf*,TID=*tid*,DEFER,NRLS.
1-6	SWITCH,*n*=*x*.
8-16	SYSREF,X=*xdn*,L=*ldn*.
SR-0013	UPDATE,P=*pldn*,I=*idn*,C=*cpldn*,N=*npldn*,L=*ldn*,E=*edn*,S=*sdn*,*=*m*,/=*c*,DW=*dw*,DC=*dc*,[, $\overset{\text{F}}{\text{Q}}$=$dk_1$:$dk_2$:...$dk_n$],[*options*].
6-6	WRITEDS,DN=*dn*,NR=*nr*,RL=*rl*.

§ Deferred implementation

Macro instruction	Page no. Part 3	Macro instruction	Page no. Part 3
ABORT	2-6	LDT	4-7
ACCESS	4-9	LOADREGS	2-38
ADJUST	4-10	MEMORY	2-2
ARGADD	5-6	MESSAGE	2-3
BKSP	3-17	MODE	2-4.1
BKSPF	3-18	NORERUN	2-11
BREG	5-5	OPEN	2-15
BUFCHECK	3-13	OUTPUT	2-27
BUFEOD	3-12	PDD	4-1
BUFEOF	3-11	POSITION	3-21
BUFIN	3-9	READ	3-1
BUFINP	3-9	READC	3-4
BUFOUT	3-10	READCP	3-4
BUFOUTP	3-10	READP	3-1
CALL	5-1	READU	3-14
CALLV	5-2	RECALL	2-5
CLOSE	2-17	RELEASE	2-17
CONTRPV	2-9	RERUN	2-11
CSECHO	2-4	REWIND	3-16
DATE	2-19	ROLL	2-10
DELAY	2-6	SAVE	4-9
DELETE	4-10	SAVEREGS	2-37
DISPOSE	2-18	SETPOS	3-19
DSP	2-13	SETRPV	2-7
DUMP	2-22	SNAP	2-21
DUMPJOB	2-13	SUBMIT	2-18.1
ENDP	2-10	SWITCH	2-4.1
ENDRPV	2-10	SYSID	2-39
ENTER	5-2	TIME	2-19
EXIT	5-4	TREG	5-5
FREAD	2-32	UFREAD	2-35
FWRITE	2-33	UFWRITE	2-36
GETMODE	2-39	WRITE	3-5
GETPOS	3-18	WRITEC	3-6
GETSWS	2-40	WRITECP	3-6
INPUT	2-24	WRITED	3-8
INSFUN	2-40	WRITEF	3-7
IOAREA	2-12	WRITEP	3-5
JDATE	2-20	WRITEU	3-15
JTIME	2-5		

READERS COMMENT FORM

CRAY-OS Version 1 Reference Manual SR-0011 K

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME ____

JOB TITLE ____________________________________

FIRM ____

ADDRESS _____________________________________

CITY ____________________ STATE _______ ZIP ________

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY
RESEARCH, INC.

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

Attention:
PUBLICATIONS

FOLD

READERS COMMENT FORM

CRAY-OS Version 1 Reference Manual SR-0011 K

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME ______________________________

JOB TITLE ______________________________

FIRM ______________________________

ADDRESS ______________________________

CITY ______________ STATE ________ ZIP ________

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY
RESEARCH, INC.

Attention:
PUBLICATIONS

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

FOLD

STAPLE

Cray Research, Inc.
Publications Department
1440 Northland Drive
Mendota Heights, MN 55120
612-452-6650
TLX 298444