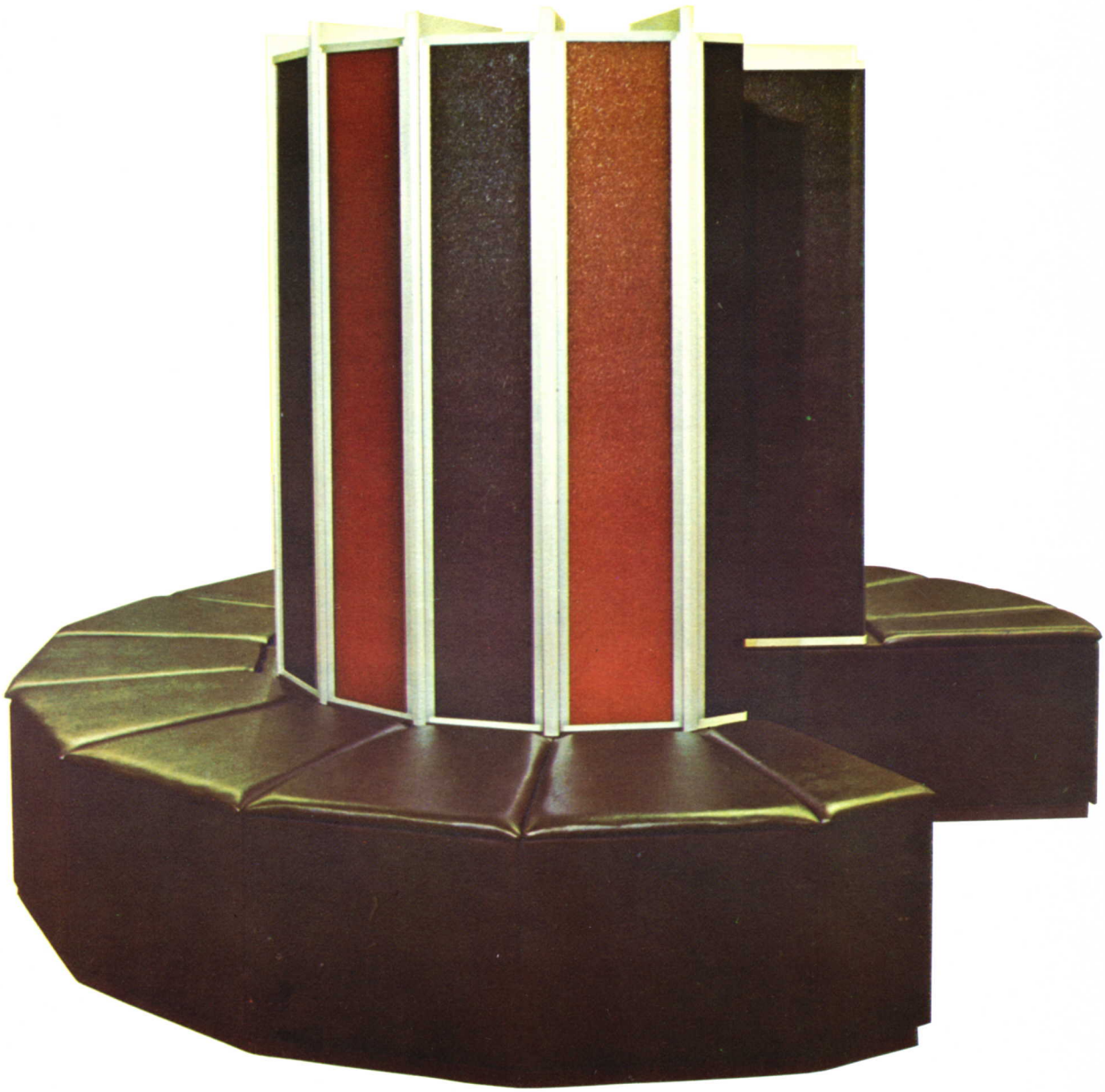
CRAY RESEARCH, INC.

CRAY-1®
COMPUTER SYSTEM

CRAY-OS VERSION 1.0
REFERENCE MANUAL
2240011



CONTROL STATEMENT SUMMARY

Page (part 2) or pub. no.	Control statement
2-4	*. *user-defined comments*
4-3	ACCESS,DN=*dn*,PDN=*pdn*,ID=*uid*,ED=*ed*,R=*rd*,W=*wt*,M=*mn*,UQ,ENTER††.
5-1	†ACQUIRE,DN=*dn*,PDN=*pdn*,ID=*uid*,ED=*ed*,RT=*rt*,R=*rd*,W=*wt*,M=*mn*,UQ,ENTER††,TEXT=*text*,MF=*mf*,TID=*tid*,DF={CD / CB / BD / BB / TR}.
4-4	ADJUST,DN=*dn*.
3-1	ASSIGN,DN=*dn*,S=*size*,A=*adn*,BS=*blk*,DV=*ldv*,RDM,U,LM=*lm*,DC={IN / ST / SC / PR / PU / PT / MT}.
7-3	AUDIT.
11-1	BUILD,I=*ddn*,L=*ldn*,OBL=*odn*,B=*bdn*,NBL=*ndn*,IMMED†,SORT,NODIR.
2240000	CAL,I=*idn*,L=*ldn*,B=*bdn*,E=*edn*,ABORT,*options*,LIST=*name*,S=*sdn*. *options* {ON XRF DUP LIS MAC MIC MIF / OFF NXRF NDUP NLIS NMAC NMIC NMIF}
2240009	CFT,I=*idn*,L=*ldn*,B=*bdn*,C=*cdn*,ON=*string*,OFF=*string*. *string*: BCGINQSTVWX
6-5	†COMPARE,A=*adn*,B=*bdn*,L=*ldn*,DF={B / I},ME=*maxe*,CP=*cpn*,CS=*csn*,ABORT.
6-2	COPYD,I=*idn*,O=*odn*.
6-2	COPYF,I=*idn*,O=*odn*,NF=*n*.
6-1	COPYR,I=*idn*,O=*odn*,NR=*n*.
2240012	CSIM,LINES=*n*,T=*tl*,OPSYS=*osdn*,OSPAR=*pfdn*,MAXBK=*ms*.
4-6	DELETE,DN=*dn*.
5-4	DISPOSE,DN=*dn*,SDN=*sdn*,DC={IN / ST / SC / PR / PU / PT / MT},DF={CD / CB / BD / BB / TR},MF=*mf*,SF=*sf*,ID=*uid*†,TID=*tid*,ED=*ed*,RT=*rt*,R=*rd*,W=*wt*,M=*mn*,TEXT=*text*†,WAIT†.
8-2	DSDUMP,I=*idn*,O=*odn*,DF={B / U},IW=*n*,NW=*n*,IR=*n*,NR=*n*,IF=*n*,NF=*n*,IS=*n*,NS=*n*.
8-1	DUMP,I=*idn*,O=*odn*,FW=*fwa*,LW=*lwa*,JTA,NXP,V,DSP.

Page (part 2) or pub. no.	Control statement
8-1	DUMPJOB.
2-3	EXIT.
2240012	EXTRACT.
2-1	JOB,JN=*jn*,M=*fl*,T=*tl*,P=*p*,US=*us*,BP††,OLM=*lm*.
9-1	LDR,DN=*dn*,LIB=*ldn*,AB=*adn*,MAP={ON / OFF / PART / FULL},T=*tra*,NX,BP††,C,OVL=*dir*,
2-2	MODE,M={1 / 2 / 3 / 4}. CNS,NA,L=*ldn*,SET={ZERO / ONES / INDEF}.
4-5	MODIFY,DN=*dn*,PDN=*pdn*†,ID=*uid*,ED=*ed*,RT=*rt*,R=*rd*,W=*wt*,M=*mn*.
7-1	PDSDUMP,DN=*dn*,DV=*ldv*,PDS=*pds*,CW=*cw*,ID=*uid*,US=*usn*,ED=*ed*,X,C,D,I,O,S.
7-2	PDSLOAD,DN=*dn*,PDS=*pds*,CW=*cw*,ID=*uid*,US=*usn*,ED=*ed*,A,I,O,S.
3-3	RELEASE,DN=*dn*.
6-4	REWIND,DN=*dn*.
2-3	RFL,M=*fl*.
4-1	SAVE,DN=*dn*,PDN=*pdn*,ID=*uid*,ED=*ed*,RT=*rt*,R=*rd*,W=*wt*,M=*mn*,UQ.
6-4	SKIPD,DN=*dn*.
6-3	SKIPF,DN=*dn*,NF=*n*.
6-3	SKIPR,DN=*dn*,NR=*n*.
2-3	SWITCH,{1 / 2 / : / 6}={ON / OFF}.
2240012	UNB,I=*idn*,O=*odn*.
2240013	UPDATE,C=*cdn*,{F / Q},I=*idn*,L=*ldn*,LIST=*string*,N=*ndn*,P=*odn*,S=*sdn*,*=*m*,/=*c*. *string*: ACDEZ
6-5	WRITEDS,DN=*dn*,NR=*nr*,RL=*rl*.

† Deferred implementation

†† See CRI publication 2240012

Note: defaults are underscored

CRAY-1®

COMPUTER SYSTEM

CRAY-OS VERSION 1.0
REFERENCE MANUAL
2240011

RECORD OF REVISION

PUBLICATION NUMBER 2240011

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; STAGE0 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 No. 2240012); deletion of UPDATE (information now documented in CRI publication No. 2240013); 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 - Corrections 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.
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.
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 which 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 have been made to bring documentation into agreement with the released version 1.02 of the operating system.

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 each new revision level. Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower right-hand corner. All changes are noted by a change bar along the margin of the page.

Requests for copies of CRAY RESEARCH, INC. publications should be directed to:

CRAY RESEARCH, INC.
7850 Metro Parkway
Suite 213
Bloomington, MN 55420

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 JCL control statement is given, along with an explanation of the function of each. Examples are provided at the end of each section.

PART 3 MACRO INSTRUCTIONS

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

Part 1

SYSTEM DESCRIPTION

CONTENTS
PART 1 SYSTEM DESCRIPTION

INTRODUCTION 1

The CRAY-1 Operating System (COS) is a multiprogramming operating system for the CRAY-1 Computer System. 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 that resides in CRAY-1 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*. 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.

The FORTRAN compiler, the CAL assembler, and UPDATE are described in separate publications.

HARDWARE REQUIREMENTS

The CRAY-1 Operating System executes on the basic configuration of the CRAY-1 Computer System, which consists of the CRAY-1 central processor unit (CPU), a maintenance control unit (MCU), and a mass storage subsystem. The CRAY-1 CPU holds the computation, memory, and I/O sections of the computer system. COS operates with any of three memory size options: one million, one-half million, and one-quarter million words. The mass storage subsystem may have one or more disk units.

Figure 1-1 illustrates a basic system configuration. For more information about CRAY-1 hardware characteristics, refer to CRI publication 2240004, The CRAY-1 Hardware Reference Manual.

SYSTEM INITIALIZATION

COS is loaded into memory and activated through a system startup procedure performed at the MCU. At startup, permanent datasets are reloaded or 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.)

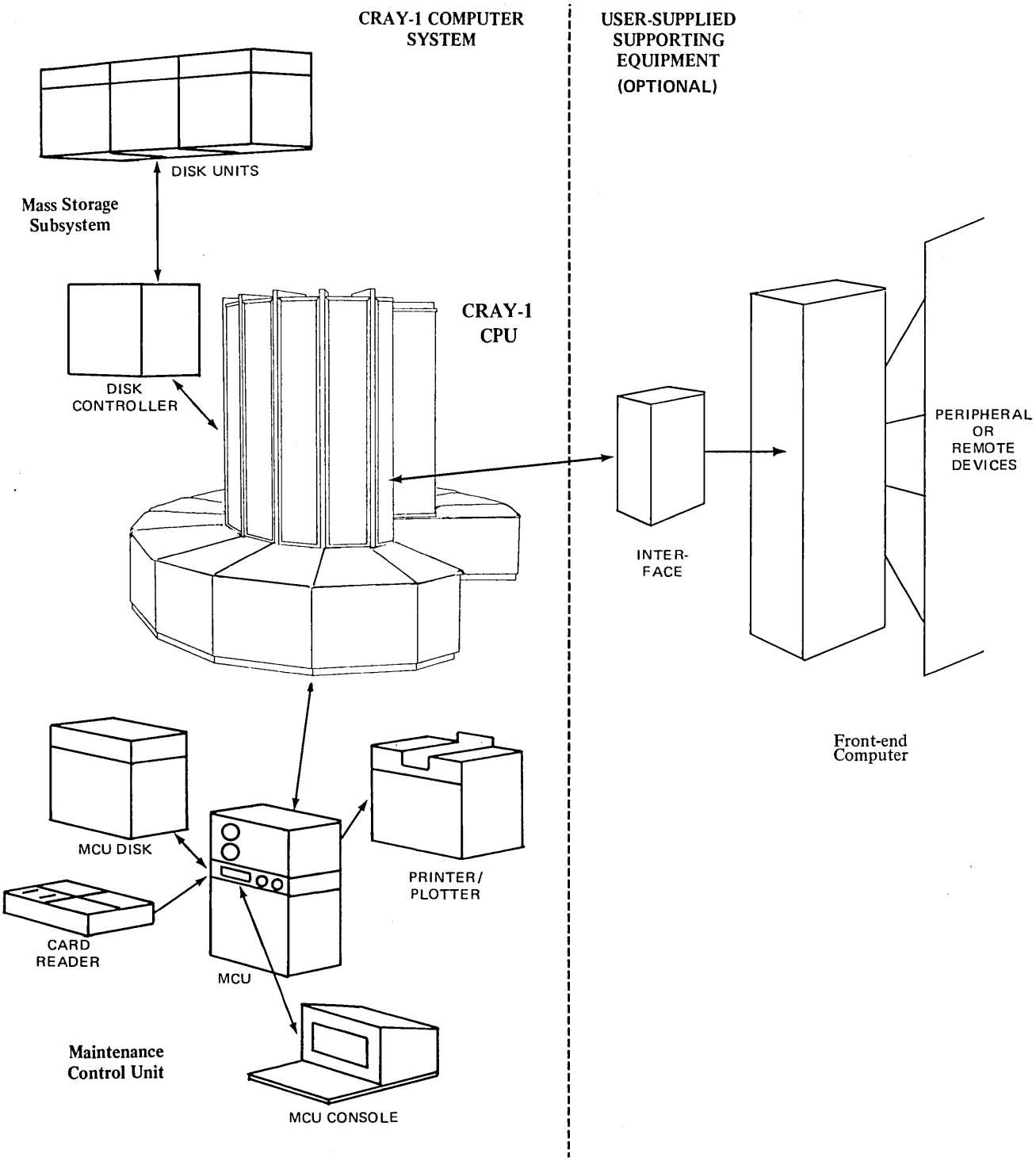


Figure 1-1. CRAY-1 system configuration

MEMORY ASSIGNMENT

Jobs running on the CRAY-1 and datasets associated with these jobs share CRAY-1 memory with each other and with COS. 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.

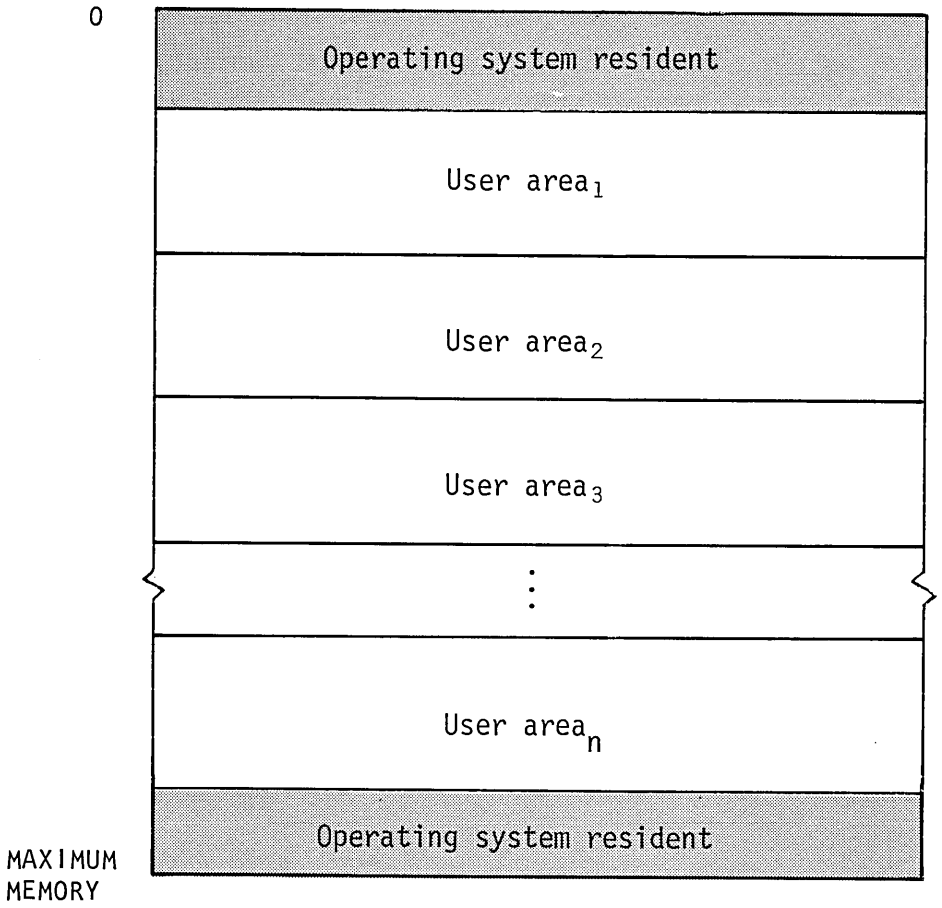


Figure 1-2. Memory assignment

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). In extreme upper memory, the operating system resident contains station I/O buffers and memory for the system log. For a detailed description of the COS components and design, refer to The System Programmer's Handbook, CRI publication 2240012.

USER AREA OF MEMORY

Every job is assigned a *user area* in memory. The user area consists of a Job Table Area 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 contiguous block of memory that immediately follows the job's JTA. The Base Address (BA) and the upper Limit Address (LA) are set by the operating system. The Limit Address is specified by a parameter on the JOB control statement (refer to Part 2) or by default. A user is able to request changes in field size during the course of a job.

Compilers, assemblers, 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. During the processing of a job step, various disk-resident portions of the operating system, such as the loader and copy utilities, may also execute in the user field.

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 intercommunications 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 buffers required for input/output operations. The user field limit is equal to (LA)-1.

Program addresses for instructions and operands are relative to BA. The CRAY-1 hardware adds the contents of BA to a relative address when it executes the instruction containing the address. A user cannot access memory outside of the user field as defined by the BA and LA register contents; LA-1 is the user limit.

MASS STORAGE USAGE

Mass storage for the CRAY-1 consists of one to forty-four DD-19 Disk Storage Units (DSUs). These devices are physically nonremovable. In general, the user is not 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. The Disk Queue Manager and the Disk Driver in the operating system translate the user's logical requests for input and output into disk controller functions automatically.

For purposes of orientation, however, the user may wish to be aware of some of the characteristics of mass storage. Operational characteristics of DD-19 Disk Storage Units are summarized in Table 1-1.

Table 1-1. Characteristics of DD-19 Disk Storage Unit

Bit capacity per drive	2.424×10^9
Tracks per surface	411
Sectors per track	18
Bits per sector	32,768
Number of head groups	10
Recording surfaces per drive	40
Latency	16.7 msec
Access time	15 - 80 msec
Data transfer rate (average bits per second)	35.4×10^6
Total bits that can be streamed to a unit (disk cylinder capacity)	5.9×10^6

All information maintained on mass storage by the CRAY-1 Operating System is organized into quantities of information known as datasets.

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

To the user, permanent datasets are those datasets that appear always to be present and available on mass storage. This "permanence" is achieved through techniques that permit the datasets noted in the Dataset Catalog to be recovered or re-established in the event of system failures. Portions of the operating system, such as the loader, utility programs, the compiler, the assembler, and library maintenance and generation routines, reside in permanent datasets and can be accessed by user jobs at any time. Datasets containing job input decks and output from jobs that have 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. This permanence of these system-defined datasets allows them to be recovered along with other permanent datasets.

Any user job can create a permanent dataset that can be subsequently accessed modified, or deleted, by any job that can produce the correct permission codes when attempting to associate it with a job. These permission codes are defined at the time the dataset is designated as permanent (i.e., saved).

A permanent dataset ceases to exist when a user with the correct permission code "deletes" it. This deletion notifies the system that the space previously occupied by the dataset can be reassigned to other datasets.

In addition to the various permanent datasets, mass storage is used for any number of datasets local to the jobs being processed. A local dataset is created by the job using it unless designated as permanent or disposed to a front-end by the job. The dataset can be accessed only by the job to which it is local. A local dataset that is neither saved as permanent nor disposed is termed a "scratch" dataset and ceases to exist when the job terminates.

The operating system allocates space to datasets by sectors on an "as-needed basis". Storage assigned to a single dataset can be non-contiguous 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.

DATASETS 2

All information maintained on mass storage 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 may be *local* to a job or *permanent* and available to the system and other jobs.

DATASET TYPES

Datasets are of two types: local and permanent.

LOCAL DATASETS

A local dataset is a dataset available only to the job that created it. Local 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 a write request or an OPEN macro call (refer to part 3, section 2).

A local dataset is empty until written on. Rewind or backspace of the dataset is necessary before it can be read. A local dataset may 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.

PERMANENT DATASETS

A permanent dataset is available to the system and to other jobs and is maintained across system deadstarts. Permanent datasets are of two types: those that are created by SAVE requests made by the user or front-end system (user permanent datasets), and those that represent input or output 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. 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 may be disposed while the job is running or may be made permanent when the job terminates and then deleted from the CRAY-1 after they have been sent to the front-end system for processing.

DATASET NAMING CONVENTIONS

The user assigns a symbolic name to each user dataset. This name is 1-7 characters, the first of which can be A-Z, $, @, or %; remaining characters can also be numeric. Certain language processors may place further restrictions on dataset names.

By CRAY-1 Operating System convention, all datasets defined by the 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

Two dataset formats are supported for the CRAY-1: blocked and unblocked.

BLOCKED FORMAT

The 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 may be composed of one or more files which are in turn composed of one or more records.

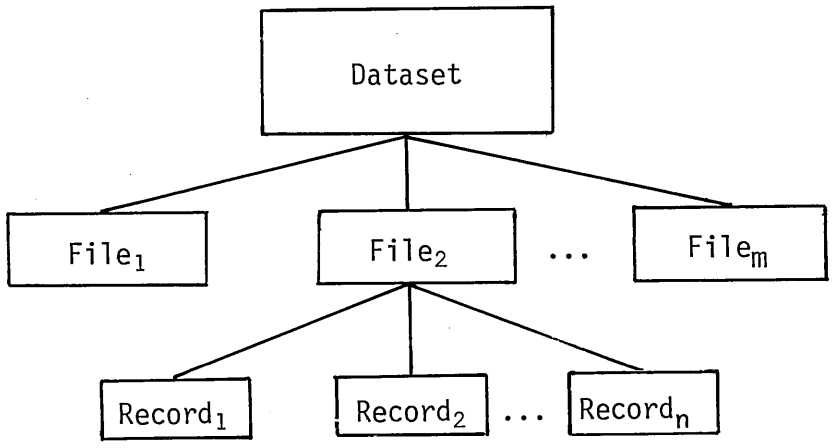


Figure 2-1. Data hierarchy within a dataset

The data in a blocked dataset may 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 (BCW)

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

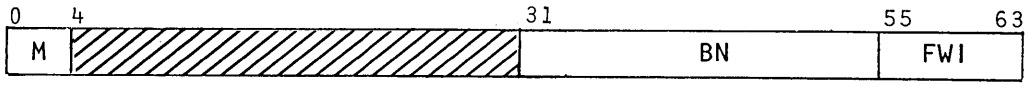


Figure 2-2. Format of a block control word

Field	Bits	Description
M	0-3	Mode indicator (for block control word, M=0)
BN	31-54	Block number; designates the number of the current data block. The first block in a dataset is block zero.
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 (RCW)

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

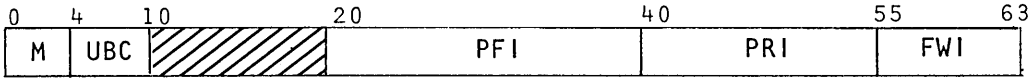


Figure 2-3. Format of a record control word

Field	Bits	Description
M	0-3	Mode: 10_8 End of record - <*eor*> 16_8 End of file - <*eof*> 17_8 End of data - <*eod*> Disregarding block control words occurring at 512-word intervals in a dataset, RCWs have the following logical relationship in a dataset. An <*eor*> RCW immediately follows the data for the record it terminates. If the record is null, i.e., contains no data, an <*eor*> RCW may immediately follow an <*eor*> or <*eof*> RCW or may be the first word of the dataset. An <*eof*> RCW immediately follows the <*eor*> RCW for the final record in a file. If the file is null, i.e., it contains no records, the <*eof*> RCW may immediately follow an <*eof*> RCW or may be the first word on the dataset. An <*eod*> RCW immediately follows the <*eof*> RCW for the final file in the dataset. If the dataset is null, the <*eod*> RCW may be the first word on the dataset.
UBC	4-9	Unused bit count. For <*eor*>, UBC designates the number of unused low order bits in the last data word in the record terminated by the <*eof*>. For <*eof*> and <*eod*> RCWs, this field is 0.
PFI	20-39	Prevous file index. This field points to the block containing the beginning of the file. The pointer is relative to the current block such that if the beginning of the file is in the same block as this RCW, PFI is 0.
PRI	40-54	Previous RCW index. This field points to the block containing the beginning of the record. The pointer 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 (i.e., if the next word is an RCW or BCW, FWI is 0).

The typical dataset has many <*eor*> RCWs per block. An example of dataset control words is illustrated in figure 2-4.

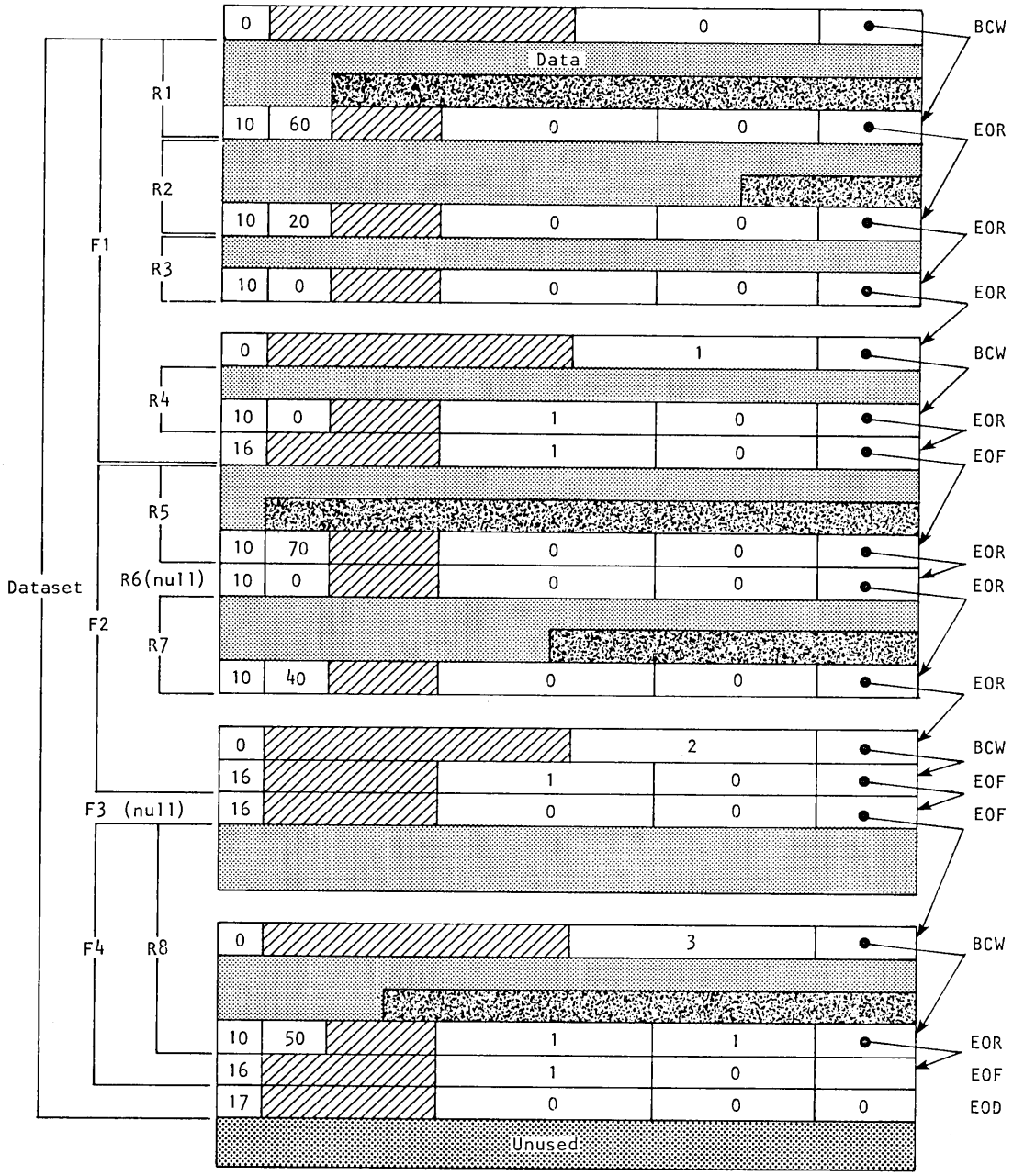


Figure 2-4. Example of dataset control words
(octal values shown)

Blank compression

Blank fields are compressed for blocked files. Blank field compression is accomplished by using a blank field initiator code followed by a count. The blank field initiator code is 33_8, the ASCII ESC code. The count is biased by 36_8. A blank field of 3 through 96 characters may be compressed to a two-character field. The actual count character is limited to $41_8 \leq$ *character count* $\leq 176_8$ (the ASCII graphics).

UNBLOCKED FORMAT

Dataset I/O may also be performed using unblocked datasets. Any dataset that does not have COS blocked format is considered unblocked. The data stream for unblocked datasets does not contain COS RCWs or BCWs.

The unblocked format is used internally by COS for some operating system generated datasets.

USER LOGICAL I/O INTERFACES

Figure 2-5 illustrates the relationship of different levels of user logical I/O interfaces and routines. It summarizes the request levels and routine calls without going into details on the movement of data between the system buffers and user program areas. All user I/O is logical; the user is never directly concerned with the actual transfer of data between the disks and the system buffers.

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 (or F$) requests.

User blocked I/O can take place at the highest level; user unblocked I/O can only be done at the lowest level.

FORTRAN statements fall into two categories: formatted/unformatted and buffered. The formatted/unformatted statements result in calls to library routines $RFI through $WUF. These routines contain calls to Logical Record I/O routines $RWDR through $BKSP or contain macros that format these calls. The logical record I/O routines perform blocking and deblocking. The Logical Record I/O routines communicate with the system through the Exchange Processor F$RDC and F$WDC requests.

Note that since all blocking and deblocking for blocked I/O is performed at the logical record level, the user desiring to perform unblocked logical I/O can do so by directly formatting the F$RDC and F$WDC Exchange Processor calls. No user routines are provided for this purpose.

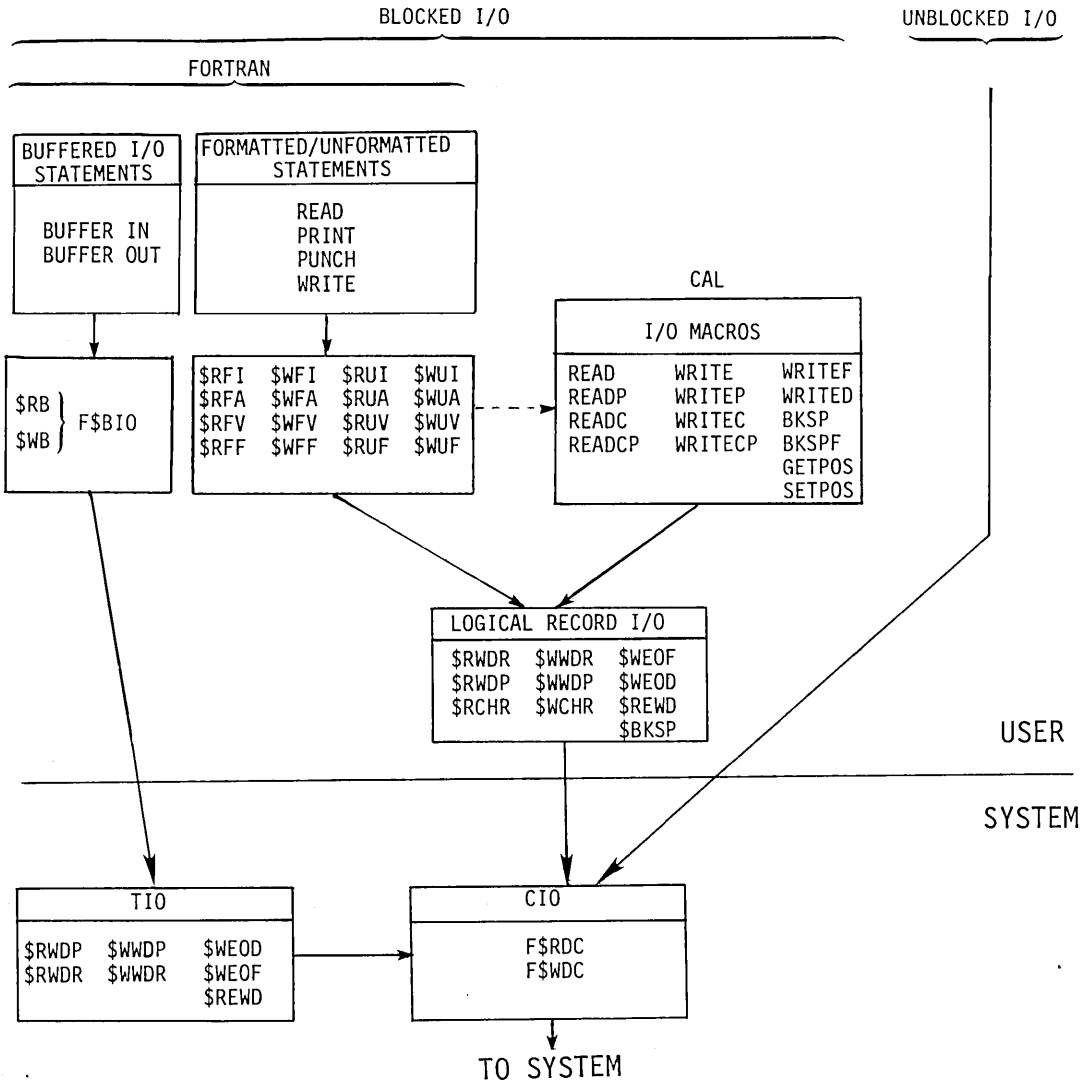


Figure 2-5. Relationship of levels of user I/O

The 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 the Circular I/O routines in COS although the mechanism for making these requests is different. CIO is the focal point for all logical I/O in the system. CIO communicates its needs for physical I/O to the Disk Queue Manager (not shown). The Disk Queue Manager coordinates all physical I/O activity in the system.

CAL language 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 CRI publication 2240009 for a description of FORTRAN statements. For descriptions of TIO and CIO, refer to CRI publication 2240012.

DATASET DISPOSITION CODES

Each dataset is assigned a *disposition code* that tells 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 two-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 will be automatically routed back to the designated mainframe to be printed on any available printer.

DECK STRUCTURE & JOB PROCESSING 3

A *job* is a unit of work submitted to a computing system. A job consists of one or more files of 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. Cards in the job deck dataset are organized into one or more files. Figure 3-1 illustrates a typical job deck consisting of a JCL control statement file, a source file, and a data file. (The physical card form for *<eof>* and *<eod>* are defined by the front-end system.)

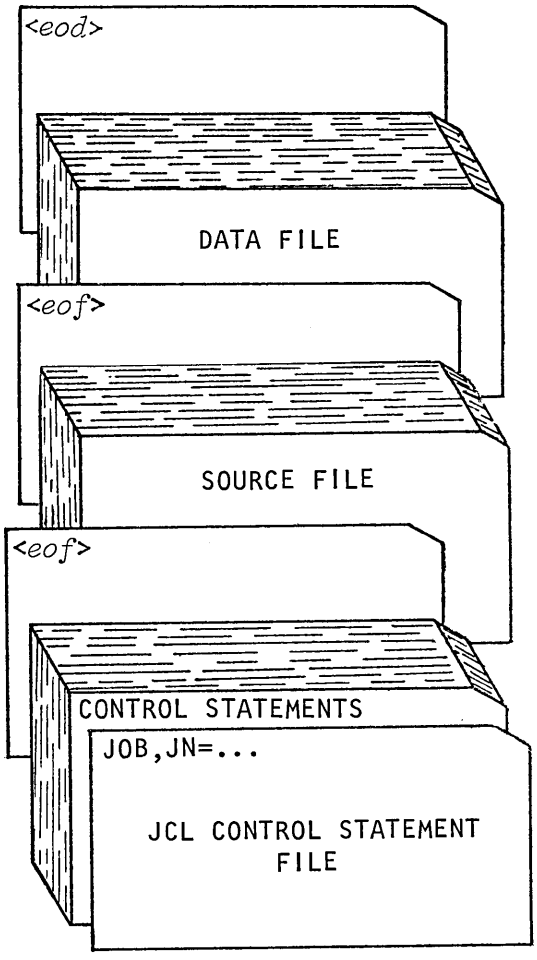


Figure 3-1. Basic job deck

The first (or only) file of the job deck must contain the Job Control Language control statements that specify the job processing requirements. (Refer to part 1, section 4 for information on control statements.) Each job begins with a JOB statement, which identifies the job to the system. All other control statements directly follow the JOB statement in the control statement file. The end of the control statement file is designated by an *<eof>* (or an *<eod>* 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 *<eod>*.

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 enters 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 the job is scheduled to begin processing. An entry is made in the system tables for the job; this makes the job input dataset permanent until it is deleted at the completion of the job.

The job input dataset is identified at the front-end computer system by a dataset name given to it by the user. The job itself can access the input dataset by its local name, $IN, or as FORTRAN unit 5 (or any other unit number *nn*, where FT*nn* is assigned as an alias for $IN; refer to part 2, section 3). $IN is initially positioned after the control statement file.

JOB INITIATION

The operating system examines the parameters on the JOB 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-related system area, 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 schedules the job for processing, it creates three datasets: $CS, $OUT, and $LOG.

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

- $OUT is the job output dataset. The job can access this dataset by name or as FORTRAN unit 6 (or as any other unit number if FT*nn* is aliased to $OUT). The disposition code for $OUT is PR (print).

- $LOG is the job's logfile and 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 $LOG with the MESSAGE system action request macro (see part 3).

JOB ADVANCEMENT

The system begins processing the job by examining each of the control statements that accompany the job deck. Control statements inform the operating system of the tasks to be performed by the job. The control statements are read, interpreted, and acted on sequentially.

The CRAY-1 Operating System multiprograms jobs so that a large number of jobs can be in some stage of processing concurrently. The CPU is shared among jobs in memory based on an installation-chosen scheduling algorithm internal to the system.

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 dataset itself is deleted from CRAY-1 mass storage.

The front-end computer will process $OUT as specified by the dataset disposition code. Output will normally be printed on the printer available at the designated front-end computer.

If, for any reason, $OUT is destroyed by the job, $LOG will be the only output given at job termination.

JOB LOGFILE AND ACCOUNTING INFORMATION

The logfile for a job consists of a list of comments at the end of the job output. The logfile presents an abbreviated history of the progress of the job through the system. Each control statement is listed in sequence, 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 five items illustrated in figure 3-2. Item 6 illustrates the accounting information given to the user.

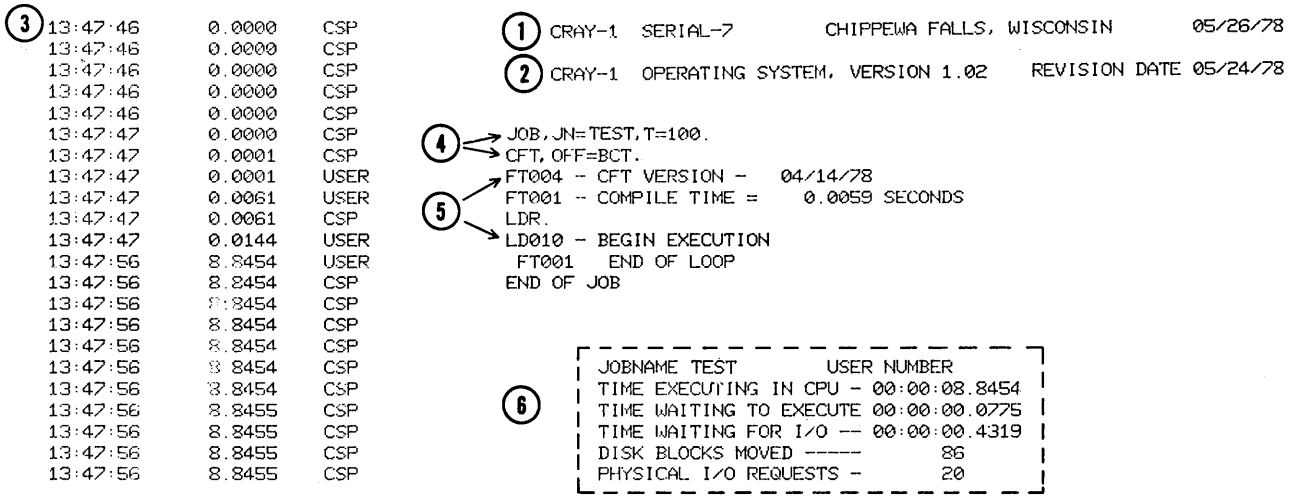


Figure 3-2. Example of a job logfile

1 First header line: identifies site and the date the job was run.

2 Second header line: identifies operating system, its current revision level, and the date of last revision.

3 Columns: 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 the system module that used the CPU time, or if execution is in the user field, notes USER. All times are in decimal. Entries commonly noted include the following:

CSP	Control Statement Processor
PDM	Permanent Dataset Manager
EXP	Exchange Package Processor
ABORT	Abort message

4 Control statements: Control statements are listed sequentially. The last control statement listed is the last one processed by the job; therefore, if the job abnormally terminates, not all of the control statements may be listed.

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 spent waiting to execute
- Time waiting for I/O
- Number of disk blocks (sectors) moved
- Number of physical I/O requests made by the job

CONTROL STATEMENTS 4

INTRODUCTION

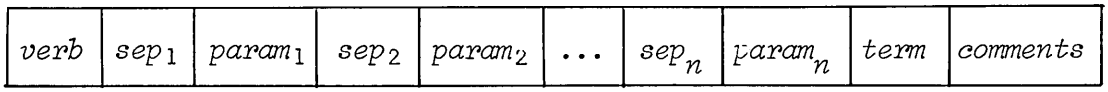
The CRAY-1 Operating System (COS) Job Control Language allows the user to present a job to the CRAY-1, define and control execution of programs within the job, and manipulate datasets associated with a job.

JCL is composed of *control statements*. Each control statement is contained on one card image and contains information for a job step. COS sequentially reads these cards from the *JCL control statement file*, $CS, created by COS.

CONTROL STATEMENT SYNTAX

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

The syntax of a control statement is:



Every control statement consists of a *verb* and a *terminator* (*term*) as a minimum. Additionally, most control statements require *parameters* and *separators* between the verb and the terminator.

The maximum number of parameters (zero, one, or more) depends on the verb. A verb and a terminator are always required; the comment is optional.

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. All other syntax errors invoke the *job step abort* procedure, which causes the system to search for an EXIT control statement in the JCL. A successful search resumes JCL processing with the job step after EXIT. If no such job step exists or if an EXIT statement is not found, the job is terminated.

VERB

A control statement verb is 1 to 7 alphanumeric characters beginning with an alphabetic character.† A verb may be preceded by blanks but must be the first field of a control statement. A verb is either a *system verb* that directs COS to take an action, or the name of a dataset containing the absolute binary of a program to be executed. Every control statement must have a verb.

If the verb is a system verb it causes the system to perform the requested action.

If the verb is not a system verb, it is assumed to be the name of one of the two types of datasets known to the system. The first type of dataset searched for is the user dataset. If the user job has a local dataset with this name, the system loads the program in absolute form from the dataset and executes it. If the verb is not the name of a user dataset, the System Directory Table (SDR) is searched. The SDR is a list of common language processors and utilities which are known to the system and made available to users at Startup. The name of the program (e.g., CAL, CFT, or DUMP) is also the name of the dataset that contains the absolute binary of the program. If a match is found in the SDR, the program is loaded and executed; otherwise, a control statement error is issued.

SEPARATOR

A separator separates the verb from the first parameter, separates parameters from one another, delimits subparameters, terminates verbs and parameters, and separates keyword from value in parameters having keyword form.

A separator is represented in a control statement by one of several ASCII characters, as illustrated in Table 4-1.

---

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

Table 4-1. Control statement separators

Graphic Symbol	Internal Code (octal)	Character	Application
:	3	Colon	Concatenation separator
=	7	Equal Sign	Equivalence separator
,	17	Comma	Parameter separator
(	17	Left parenthesis	Optional initial separator
.	37	Period	Terminator
)	37	Right parenthesis	Optional terminator

Parameter separator

A parameter separator separates the control statement verb from the first parameter and subsequent adjacent parameters from each other. The parameter separator character is the comma. Optionally, a left parenthesis may be used to indicate the beginning of a parameter string. Adjacent parameter separators indicate a null parameter, which is ignored unless it is a positional parameter.

Equivalence separator

An equivalence separator separates a parameter keyword from the first parameter value for that keyword. The equivalence separator character is the equal sign. Adjacent equivalence separators are illegal.

Concatenation separator

A concatenation separator separates multiple parameter values from each other. The concatenation separator character is the colon.

Terminator

A terminator signifies the end of control statement information. Any characters following ther terminator on the card image represent comments. Every control statement requires a terminator. The control statement terminator is the period or, optionally, the right parenthesis.

LITERAL DELIMITER†

The literal delimiter character, the apostrophe ('), brackets a character string that is to be taken literally as the value of a parameter. Any ASCII graphic character (character codes 040_8 through 176_8) can be included in a literal character string. The character string can be of any length, with possible restrictions depending on the verb. Two successive apostrophes within the literal string represent a single apostrophe. The literal character has the octal internal code 01.

Characters normally interpreted as separators or terminators are not interpreted as such within a literal string. The first occurrence of a literal delimiter denotes the beginning of the literal string; the second occurrence signifies the end of the literal string (continued literals represent an exception). Two adjacent literal delimiters represent a null literal string.

Examples of valid literal strings:

'ABC'	String consists of the characters A, B, and C.
'ABC=DEF'	The equal sign is not interpreted as a separator.
''	The literal string is null.
'DON''T'	The literal string is interpreted as DON'T.
'A VERY LONG LITERAL STRING WITH ,.=:()'	

PARAMETER

A *parameter* is a control statement argument, the exact requirements of which are defined by the verb. Parameters are either *positional* or have a *keyword* form.

Positional parameters

A positional parameter has a precise position relative to separators in the control statement. Even when a positional parameter is null, the separator that delimits it from the verb or other parameters cannot be omitted.

---

† Deferred implementation

The format for a positional parameter is:

value

or *value*$_1$:*value*$_2$:...*value*$_{n-1}$:*value*$_n$

where each *value*$_i$ is a string of 1-7 alphanumeric characters or a null string. All positional parameters are required to be represented by at least one *value*, although the value may be null.

Examples of positional parameters:

...,ABCDE,...	Parameter value is ABCDE.
...,,...	The adjacent parameter separators indicate a null positional parameter.
...,P1:P2:P3,...	Parameter values are concatenated.

NOTE

Although the control statement syntax supports positional parameters, no COS *system verb* or Cray Research product uses them.

Keyword parameters

A keyword parameter is identified by its form rather than its position. The keyword is a string of 1 to 7 alphanumeric characters uniquely identifying the parameter. Parameters of this type can occur in any order or can be omitted.

The format of a keyword parameter is:

keyword

or *keyword = value*

or *keyword = value*$_1$*:value*$_2$*:value*$_3$*:...value*$_{n-1}$*:value*$_n$

where *keyword* is an alphanumeric string that depends on the requirements of the verb and *value*$_i$ is a string of 0 or more alphanumeric characters or a null string. A keyword parameter may occur anywhere in the control statement. Whether or not a keyword parameter is required depends on the control statement verb. If the keyword is not included in the control statement, a default value may be assigned.

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 concatenated values.
...,DN=,...	Null parameter value.
...,DN=A:::B,...	A,B, and two null parameter values concatenated.

COMMENTS

A *comment* is an arbitrary ASCII graphic character string that allows the user to annotate a control statement. All comments appear in the logfile. A comment follows a control statement terminator. The control statement interpreter ignores comments.

BLANKS

All blanks are ignored.

INTERNAL REPRESENTATION OF CONTROL STATEMENTS

COS interprets control statements one card image at a time and stores the elements comprising the statement in the Job Communication Block (JCB) starting at word 20_8. A 0- to 15-character string delimited by two separators is stored in two words. The two-word area contains, left-adjusted, the 0 to 15 characters of the string, followed by trailing zero fill (for strings < 15 characters), with the internal code for a separator in the rightmost character of the second word. (See Table 4-1.) For example, if the string is eight characters or fewer, it is left-adjusted with zero fill in the first word and the separator is in the second word.

A copy of the original control statement (uncracked) is stored eight characters per word starting at word 5 of the JCB.

Example:

The following control statement is shown stored in the JCB:

ASSIGN,DN=MYFILE,BS=25.

0								
5	A	S	S	I	G	N	,	D
	N	=	M	Y	F	I	L	E
	,	B	S	=	2	5	.	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	⋮							
17_8	0	0	0	0	0	0	0	0
20_8	A	S	S	I	G	N	0	0
	0	0	0	0	0	0	0	17_8
	D	N	0	0	0	0	0	0
	0	0	0	0	0	0	0	07_8
	M	Y	F	I	L	E	0	0
	0	0	0	0	0	0	0	17_8
	B	S	0	0	0	0	0	0
	0	0	0	0	0	0	0	07_8
	2	5	0	0	0	0	0	0
	0	0	0	0	0	0	0	37_8

Part 2

JOB CONTROL LANGUAGE

CONTENTS
PART 2 JOB CONTROL LANGUAGE

INTRODUCTION 1

The first file of a job dataset contains control statements that are read and processed sequentially.

Control statements identify the job to the system, define operating characteristics for the job, manipulate datasets, call for the loading and execution of user programs, and call programs that perform a number of utility functions for the user.

Information on the general syntax rules and conventions for control statements is presented in Part 1, Section 4 of this publication. In the sections following, COS control statements are described individually and examples are given. The control statements have been divided into the following categories:

- Job Definition and Control - JOB, MODE, EXIT, RFL, SWITCH, and *
- Dataset Management - ASSIGN and RELEASE
- Permanent Dataset Management - SAVE, ACCESS, ADJUST, MODIFY, and DELETE
- Dataset Staging Control - ACQUIRE† and DISPOSE
- Dataset Utilities - COPYR, COPYF, COPYD, SKIPR, SKIPF, SKIPD, REWIND, WRITEDS, and COMPARE†
- Permanent Dataset Utilities - PDSDUMP, PDSLOAD, and AUDIT
- Analytical Aids - DUMPJOB, DUMP, and DSDUMP

Also available to the user and described in this part of the manual are the Relocatable and Overlay Loader and the Build utility.

† Deferred implementation

JOB DEFINITION AND CONTROL 2

A number of control statements allow the user to specify job processing requirements. JCL 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 such characteristics as size, time limit, and priority levels.
- MODE allows the user to define the settings for the mode (M) register in the exchange package for the job.
- EXIT indicates the point in the control statement file at which processing of control statements will resume 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 a user to turn on or turn off pseudo sense switches.
- * allows the user to annotate JCL statements with comments.

JOB - JOB IDENTIFICATION

The JOB control statement must be the first statement in a control statement file. It defines the job to the operating system. JOB is a system verb.

Format:

```
JOB,JN=jn,M=fl,T=tl,P=p,US=us,OLM=lm.
```

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

JN=*jn* Job name. 1-7 alphanumeric characters, the first of which is A-Z; remaining characters may also be numeric. 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 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 (also an installation parameter).

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 is allowed.

P=*p* Priority level at which the job enters the system. This parameter may assume the values of 0-15 decimal. If omitted, a value specified by the installation is assumed.

US=*us* User number. 1-15 alphanumeric characters. The default is 0. This parameter identifies the user to whom this job belongs. The capability of having user numbers is provided for installation accounting; their specific application is defined by the installation.

OLM=*lm* Size of $OUT. *lm* specifies a decimal count of 512-word blocks. A block holds about 45 print lines (approximately one full page of output). The default and maximum values for *lm* are defined by the installation.

MODE - SET OPERATING MODE

The MODE control statement allows the user to define the settings for the mode (M) register in the exchange package for the job. The only mode register flag that can be set by this statement is the floating point error mode flag. 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.
```

† The *fl* parameter on the JOB statement does not include the job's JTA; space for the JTA is added by the system. The installation parameter, however, does include the JTA.

Parameters:

M=*mode*	A single octal digit having one of the following values: 1 or 2 No interrupt on floating point errors 3 or 4 Interrupt on floating point errors

EXIT - EXIT PROCESSING

An EXIT control statement indicates the point in the control statement file at which processing of control statements will resume following a job step abort from a program. If no job step abort occurs, it 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; octal number of 1000_8-word blocks of memory to be assigned to the job; this number does not include the JTA (see footnote on the M parameter of the JOB statement)

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 through 6) to be turned on or off

x	Switch position	
	ON	Switch *n* is turned on
	OFF	Switch *n* is turned off

* - JCL COMMENT STATEMENT

The comment control statement allows the user to annotate JCL statements with comments. * is a system verb.

Format:

```
*.      user-defined comments
```

Parameters: none

DATASET DEFINITION AND CONTROL 3

Datasets may be defined and managed by use of dataset control statements. Control statements available:

- ASSIGN allows the user to create a dataset and assign dataset characteristics or to define an alias for an existing dataset.
- RELEASE relinquishes access to the named dataset for the job.

ASSIGN - ASSIGN DATASET CHARACTERISTICS

The ASSIGN control statement is used to create a dataset and assign dataset characteristics or to define an alias for an existing dataset. 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. ASSIGN statements are not required for defining the alias FT05 for the $IN dataset or for defining the alias FT06 for the $OUT dataset; aliases for these two datasets are defined automatically during job initialization. ASSIGN is a system verb.

Format:

ASSIGN,DN=*dn*,S=*size*,A=*adn*,BS=*blk*,DV=*ldv*,RDM,U,LM=*lm*,DC=*dc*.

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.

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.

A=*adn* Alias; alternate dataset name. Although any name can be used, in practice, common usage is limited to FORTRAN. By FORTRAN convention, an alias is specified in the form FT*nn* where *nn* is a FORTRAN unit number from 00-99. An alias assigned to one dataset can be reassigned to a second dataset; the new assignment takes precedence over the previous assignment. A dataset may have multiple aliases

and can be assigned a new alias at any time. The new alias is added to the list of current aliases.

BS=*blk* Buffer size. Octal number of 1000_8-word blocks to be reserved for user buffer. The default number of blocks is set by an installation parameter.

DV=*ldv* Logical device on which dataset is to begin. If a logical device name is not given, one is chosen by the system.

RDM Random dataset. If the RDM parameter is present, the dataset is to be accessed randomly. If the RDM parameter is absent, the dataset is to be accessed sequentially.

U Undefined 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.

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.

DC=*dc* Disposition code; disposition to be made of the dataset at job termination. The default is SC when the DC parameter is omitted.

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

- IN Input to mainframe. Dataset is placed in the job input queue for the mainframe of job origin.
- 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 any printer available 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 Write dataset on magnetic tape at the mainframe of job origin.

NOTE

Currently, the system does not automatically flush the memory buffers during job termination. Therefore, the user must explicitly do a rewind or release to flush the memory buffers.

RELEASE - RELEASE DATASET

The RELEASE control statement relinquishes access to the named dataset for the job. If the 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 has a disposition code other than SC, then the dataset is entered in the output queue for staging to the mainframe of job origin. RELEASE is a system verb.

Format:

```
RELEASE,DN=dn.
```

Parameters:

DN=*dn* Name of dataset to be released.

PERMANENT DATASET MANAGEMENT 4

Permanent dataset management provides methods for creating, protecting, and accessing datasets that are assigned permanently to mass storage. 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 accessable by the user. (See part 1, section 2 for a description of the types of datasets.)

The user manages user permanent datasets by communicating with the Permanent Dataset Manager through the following control statements:

- SAVE enters a dataset's identification and location in a system-maintained Dataset Catalog (DSC). Datasets recorded in the DSC are user permanent datasets and are recoverable at deadstart.
- ACCESS causes a saved dataset to be assigned to a job. The usage (reading or writing, for example) of a dataset is determined by permissions granted when the dataset is accessed.
- ADJUST changes the size of a user permanent dataset in the Dataset Catalog (DSC).
- MODIFY changes established information for an existing user permanent dataset in the DSC.
- DELETE causes a saved dataset to be removed from the DSC.

SAVE - SAVE PERMANENT DATASET

The SAVE control statement makes a local dataset permanent. Saving a dataset consists of making an entry in the Dataset Catalog. A permanent dataset is uniquely identified by permanent dataset name, user identification, and edition number. SAVE is a system verb.

SAVE has a two-fold function:

1. Creation of an initial edition of a permanent dataset, or
2. Creation of an additional edition of a permanent dataset.

The 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. The read and write control words specified on the oldest existing edition of a permanent dataset apply to all subsequent editions of that dataset.

NOTE

The assurance that a dataset is written to disk prior to being saved is the responsibility of the user. This may be accomplished by writing an end of data on the dataset. Issuing a REWIND is a convenient way of writing an end of data on the dataset. If no data at all has been written to disk, the SAVE request will abort. If some data has been written, but the last buffer has never been written, the SAVE will succeed, but a subsequent job attempting to use the dataset may abort.

Format:

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

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

DN=*dn* Name of a dataset that is local to the job.

PDN=*pdn* Permanent dataset name; 1-15 alphanumeric 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 through 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; a value specifying the number of days a permanent dataset is to be retained by the system. The parameter may assume the value of 0 through 4095. 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 UQ is specified, only this job may access the permanent dataset at the completion of the SAVE function. Otherwise, multi-user access to the permanent dataset is granted.

ACCESS - ACCESS PERMANENT DATASET

The ACCESS control statement makes an existing permanent dataset local to a job. Following the ACCESS, 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.

Permanent datasets entered into the System Directory (SDR) do not require the use of the ACCESS control statement prior to being accessed. 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 the 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.
```

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

DN=*dn* Dataset name by which the permanent dataset is to be known.

PDN=*pdn* Name of a permanent dataset being accessed and which must already exist in the system. The default value is *dn*.

ID=*uid* 1-8 character user identification. If *uid* was specified at SAVE time, the ID parameter must be specified on the ACCESS control statement. The default is no user ID.

ED=*ed* Edition number of permanent dataset being accessed; a value from 1 to 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).

R=*rd* Read control word as specified at SAVE time. 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.

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.

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, e.g., maintenance permission is required to delete a dataset.

UQ Unique access. If the UQ parameter is specified and the appropriate write or maintenance control words are specified, write, maintenance, and/or read permission may be granted. If UQ is not specified, multi-user read access is granted by default (if at a minimum the read control word is specified).

ADJUST - ADJUST PERMANENT DATASET

The ADJUST control statement changes the size of a permanent dataset, that is, it redefines dataset size for the dataset. When a permanent dataset is overwritten and the dataset size changes, issuing an ADJUST 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.

Format:

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

Parameters:

DN=*dn* Local dataset name of a permanent dataset that has been accessed with write permission.

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 and all permissions before a MODIFY of a permanent dataset may 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 single edition dataset.

Format:

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

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.

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.

† Deferred implementation

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=*mn* 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.

DELETE - DELETE PERMANENT DATASET

The DELETE control statement removes a 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. The dataset remains a local dataset after DELETE. DELETE is a system verb.

Format:

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

Parameters:

DN=*dn* Local dataset name of a permanent dataset that has been accessed with maintenance permission.

DATASET STAGING CONTROL 5

Two control statements support staging of datasets between the CRAY-1 and a front-end system: ACQUIRE† and DISPOSE.

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

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 - ACQUIRE PERMANENT DATASET†

A dataset may be made permanent and accessible to the job making the request by use of the ACQUIRE control statement.

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

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. ACQUIRE is a system verb.

† Deferred implementation

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. This is a required parameter.

PDN=*pdn* Name of CRAY-1 permanent dataset to be accessed or staged from a front-end system, saved, and accessed; 1-15 alphanumeric characters assigned by the dataset creator. This is the name that is saved by the system if the dataset is staged. 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 to 4095 assigned by the dataset creator. The default value is:

- One, if there is no permanent dataset currently in existence with the same PDN and ID, 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; a value specifying the number of days that a permanent dataset is to be retained by the system. The parameter may assume a value from 0 to 4095. The default value is an installation-defined parameter.

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

W=*wt* Write control word; 1-8 alphanumeric characters assigned by the dataset creator. 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 (i.e., maintenance permission is required to delete a dataset).

UQ Unique access. If specified, the job is granted unique access to the permanent dataset; otherwise, multi-access 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 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 CRAY-OS record control words are extracted from the text string before it is passed to the front-end. *text* cannot exceed 120 characters.

MF=*mf* Mainframe identifier for the front-end computer; two alphanumeric characters. Default is mainframe of job origin.

TID=*tid* Terminal identifier. 1-8 alphanumeric characters; for identifying destination terminal. 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 when the DF parameter is omitted.

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 two-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 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 performs no character conversion. BD is the same as TR for ACQUIRE.

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.

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.

DISPOSE - DISPOSE DATASET

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

The dataset being disposed is placed in the output queue immediately. Disposing a dataset does not delay job processing unless the WAIT† parameter is specified. After the dataset is staged, it is released from CRAY-1 mass storage. 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.
```

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

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

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

† Deferred implementation

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 to mainframe. Dataset is placed in the job input queue for the mainframe designated by the MF parameter. If no MF parameter is specified, the job is disposed to the CRAY-1 input queue.
- 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.
- 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 when the DF parameter is omitted.

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 two-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 computer 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 computer 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. 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; 2 alphanumeric characters. The default is the mainframe of job origin.

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 for identifying destination terminal. The default is terminal of job origin, where applicable.

ED=*ed* Edition number meaningful only if DC=ST; a value from 1 to 4095, assigned by the user. The default value depends on the destination mainframe.

RT=*rt* Retention period meaningful only if DC=ST; a value from 0 to 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.

† Deferred implementation

TEXT=*text*† Text to be passed to the front-end system requesting transfer of a dataset; meaningful only if DC=ST. 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 120 characters. If no TEXT field is specified, the default value is SDN if specified, otherwise the default is DN.

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.

† Deferred implementation

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; i.e., 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.
- COMPARE† compares two datasets and makes a list of all discrepancies found between the two.

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.

Format:

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

Parameters (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 for *n* is 1. If the dataset contains fewer than *n* records, the copy prematurely terminates on the next <*eof*>. <*eof*> or <*eod*> is not written. If the keyword NR is specified without a value, the copy terminates at the next <*eof*>. If the input dataset is positioned midrecord, the partial record is counted as one record.

† Deferred implementation

Following the copy, the datasets are positioned after the <*eor*> for the last record copied.

COPYF - COPY FILES

The COPYF statement copies a specified number of files from one dataset to another starting at the current dataset position.

Format:

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

Parameters (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. If the dataset contains fewer than *n* files, the copy prematurely terminates on <*eod*>. <*eod*> is not written. The default for *n* is 1. If the keyword NF is specified without a value, the copy terminates at the <*eod*>. If the input dataset is positioned midfile, the partial file counts as one file.

Following the copy, the datasets are positioned after the <*eof*> for the last file copied.

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 <*eof*> of the last file copied. The <*eod*> is not written to the output dataset.

Format:

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

Parameters (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 (in keyword form):

DN=*dn* Dataset name. 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, then the system positions *dn* after the last <*eor*> of the current file. If *n* is negative, SKIPR skips backward on *dn*.

SKIPR does not bypass an <*eof*>. If an <*eof*> is encountered before *n* records have been bypassed when skipping backward, the dataset is positioned after the <*eof*>; when skipping forward, the dataset is positioned after the last <*eor*> of the current file.

SKIPR does not bypass an <*eof*> or <*bod*>. If an <*eof*> or <*bod*> is encountered before *n* records have been bypassed when skipping backward, the dataset is positioned after the <*eof*> or <*bod*>; when skipping forward, the dataset is positioned after the last <*eor*> of the current file.

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 (in keyword form):

DN=*dn* Dataset name. 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, then the system positions *dn* after the last <*eof*> 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 <*eod*> or <*bod*>. If <*bod*> is encountered before *n* files have been bypassed when skipping backward, the dataset is positioned after the <*bod*>; when skipping forward, the dataset is positioned before the <*eod*> of the current file.

EXAMPLE: If *dn* is positioned just after an <*eof*>, the following control statement will position *dn* after the previous <*eof*>. 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 <*eod*>, that is, after the last <*eof*> of the dataset. It has the same effect as the following statement:

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

Format:

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

Parameters (in keyword form):

DN=*dn* Dataset name. The default is $IN.

If the specified dataset is empty or already at <*eod*>, the statement has no effect.

REWIND - REWIND DATASET

The REWIND control statement positions a dataset at the beginning of data <*bod*>, i.e., prior to the first block control word of the dataset. If the dataset is not open, REWIND opens it. REWIND is a system verb.

Format:

```
REWIND,DN=dn.
```

Parameters (in keyword form):

DN=*dn* Name of dataset to be rewound.

WRITEDS - WRITE RANDOM OR SEQUENTIAL DATASET

The WRITEDS control statement is intended for initializing a random dataset. It writes a dataset containing a single file which consists of a specified number of records of a specified length. This utility is useful for random datasets because of the requirement that a record written on a random dataset must end on a pre-existing record boundary or must begin at *<eod>*.

WRITEDS may 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. This parameter is required.
NR=*nr*	Decimal number of records to be written. This parameter is required.
RL=*rl*	Decimal record length, i.e., 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 0.

COMPARE - COMPARE DATASETS†

The COMPARE control statement compares two datasets and makes a list of all discrepancies found between the two. Output consists of a listing of the location of each discrepancy and the contents of the differing portions of the datasets. If no discrepancies are found, a message in the user logfile informs the user that the datasets agree.

Keyword parameters allow the user to specify the maximum number of errors and the amount of context to be listed as output.

† Deferred implementation

If only parts of two datasets are to be compared, the user must make copies of the parts 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,ABORT.
```

Parameters are in keyword form; one of A and B must be specified.

A=*adn*
B=*bdn*
: Input dataset names. The default is $IN; default may be taken for one input dataset only - at least one of A and B must be specified. If *adn*=*bdn*, an error message will be sent.

L= *ldn*
: Dataset name for list of discrepancies. The default is $OUT. L must be different from A and B.

DF=*df*
: Input dataset format. The default is T.

  df is a one-character alpha code as follows:

  B Binary. The input datasets will be compared logically to verify that they are identical. If they are not identical, the differing words will be printed in octal and as ASCII characters. The location printed will be a word count in decimal. The first word of each dataset is called word 1.

  T Text. If the input datasets are not identical, they will be 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 errors printed. The default is 100.

CP=*cpn*
: Amount of context printed. *cpn* words (if DF=B) or records (if DF=T) to either side of a discrepancy are printed. The default is 0.

CS=*csn* Amount of context scanned. *csn* words (if DF=B) or records (if DF=T) to either side of a discrepancy are scanned for a match. The default is 0.

If a match is found within the permitted range, subsequent comparisons will be made at the same interval. That is, if record 275 of dataset A is equivalent to record 277 of dataset B, the next comparison will be between record 276 of dataset A and record 278 of dataset B.

CAUTION

If identical records occur within *csn* records of each other, the pairing is ambiguous and COMPARE may match the wrong pair.

ABORT If ABORT is specified, the job ends with an error exit if any discrepancies are found between the two input datasets. Specifying ABORT will not prevent the listing of up to *maxe* errors.

PERMANENT DATASET UTILITIES 7

Three utility routines are provided for permanent datasets.

- PDSDUMP dumps all 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 permanent datasets to a dataset, which may then be saved or staged to a station as desired.

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; the only required parameter is CW. Optional parameters establish criteria for those datasets to be dumped. By default, all permanent datasets will be dumped.

DN=*dn*	Dataset to which dump is to be written. Default is $PDS.
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 to regulate the use of PDSDUMP. This is a required parameter.
ID=*uid*	Dump all datasets with user identification as specified.
US=*usn*	Dump all datasets with specified user number.
ED=*ed*	Edition number of permanent dataset to be dumped; meaningful only if PDS parameter is specified.

X — Dump expired datasets.

C — Dump datasets that have been modified or created since last dump of the dataset.

D — Delete datasets that are dumped.

I — Dump input datasets.

O — Dump output datasets.

S — Dump saved datasets.

} See note

NOTE

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

PDSLOAD - LOAD PERMANENT DATASET

PDSLOAD loads permanent datasets from a dataset created by PDSDUMP. Dataset Catalog (DSC) entries are reconstructed.

Format:

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

All parameters are in keyword form; the only required parameter is CW. Optional parameters establish criteria for those datasets to be loaded. By default, all permanent datasets will be loaded.

DN=*dn* — Dataset from which permanent dataset is to be reconstructed. The default is $PDS.

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

CW=*cw* — Installation-defined control word to regulate the use of PDSLOAD. This is a required parameter.

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.

A — Load only active datasets, i.e., do not load expired datasets.

I	Load input datasets.	See note
O	Load output datasets.	
S	Load saved datasets.	

NOTE

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

AUDIT - AUDIT PERMANENT DATASET

The AUDIT utility provides reports on the status of each permanent dataset known to the system. AUDIT does not include input or output datasets.

Format:

```
AUDIT.
```

Parameters: none

AUDIT output supplies the following information to $OUT:

- Permanent dataset name
- User identification
- Edition number
- User number
- Creation date/time
- Last access date/time
- Last modify date/time
- Expiration date
- Last dump date/time
- Dataset size
- Number of accesses

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 execution. DUMPJOB captures the information so that DUMP can later format and output selected parts of it.
- DSDUMP dumps all or part of a dataset to another dataset in one of two formats: blocked or unblocked.

DUMPJOB - CREATES $DUMP

The DUMPJOB control statement causes the local dataset $DUMP to be created. $DUMP receives an image of the memory assigned to the job (JTA and user field) at the time the DUMPJOB statement is encountered. DUMPJOB may be placed anywhere in the control statement file.

If $DUMP already exists, it will be overwritten by the issuing of a DUMPJOB control statement. If $DUMP is permanent and the job does not have write permission, DUMPJOB will abort. If $DUMP is permanent and the job has write permission, the dataset will be overwritten. An ADJUST control statement may be required to make any extension of the dataset permanent.

$DUMP is created as an unblocked dataset by DUMPJOB for use by DUMP. DUMPJOB is a system verb.

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 may be placed anywhere in the control statement file after $DUMP has been created by the DUMPJOB control statement.

Format:

```
DUMP,I=idn,O=odn,FW=fwa,LW=lwa,JTA,NXP,V,DSP.
```

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 format is acceptable.

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

FW=*fwa* First word address (in octal) of memory to dump. The default is 0.

LW=*lwa* Last word address+1 (in octal) 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, or T registers dumped. The default causes exchange package, B, and T registers to be dumped.

V Vector registers to be dumped. The default is no dump of V registers.

DSP Logical File Tables and Dataset Parameter Areas to be dumped. The default is to not dump LFTs and DSPs.

Placing the DUMPJOB and DUMP statements after an EXIT statement is conventional and provides the advantage of giving a dump regardless of which part of the job caused an error exit. The usage of DUMP and DUMPJOB, however, is not restricted to this purpose.

DUMP may 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.

DSDUMP - DUMP DATASET

The DSDUMP control statement dumps a specified portion 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 each begins with one and is relative to the current dataset position. Specifying an initial number other 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 beginning of the dataset, the dataset must be positioned properly prior to calling DSDUMP. When DSDUMP is completed, the input dataset is positioned after the last record read.

The unblocked format is used for datasets that do not follow the COS blocked format or for examining control words in a blocked dataset. A group of sectors within the dataset may be selected. The initial sector number begins with one and is always relative to the beginning of the dataset. The number of words dumped is always 512 words per sector. Following a DSDUMP in unblocked format, the input 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 requried parameter is I.

I=*idn*	Name of dataset to be dumped. This is a required parameter.
O=*odn*	Name of dataset to receive the dump. The default is $OUT.
DF=*df*	Dump format can be blocked (B) or unblocked (U); default is B.
IW=*n*	Decimal number (*n*) of initial word for each record on *idn*; the default is 1.
NW=*n*	Decimal number (*n*) of words per record to dump; the default is 1. Specifying NW without a value gives all words to the end of the record.
IR=*n*	Decimal number (*n*) of initial record for each file on *idn*; the default is 1.
NR=*n*	Decimal number (*n*) of records per file to dump; the default is 1. Specifying NR without a value gives all the records to the end of the file.
IF=*n*	Decimal number (*n*) of initial file for dataset on *idn*; the default is 1.
NF=*n*	Decimal number (*n*) of files *idn* to dump; the default is 1. Specifying NF without a value gives all the files to the end of the dataset.
IS=*n*	Decimal number (*n*) of initial sector on *idn*; the default is 1. Applicable only if DF=U.
NS=*n*	Decimal number (*n*) of sectors to dump; the default is 1. Specifying NS without a value gives all the sectors to the end of the dataset. Applicable only if DF=U.

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 it is 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.

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.

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,AB=adn,MAP=op,T=tra,NX,C,OVL=dir,CNS,NA,L=ldn,SET=val.
```

Parameters are in keyword form:

DN=*dn* Dataset containing modules to be loaded. The default is $BLD. Loading continues until an <*eof*> is reached. Modules are loaded according to block name as determined by a CAL IDENT card or a CFT PROGRAM, SUBROUTINE, 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 the rewind operation. Therefore, the statement

```
DN=LOAD3:LOAD3
```

causes the loading of all modules in the first two files of dataset LOAD3.

LIB=*ldn* The LIB parameter names the dataset from which unsatisfied externals are loaded. Datasets indicated are selectively scanned for load modules whose entry points are unsatisfied externals from the load of *ldn* datasets. The default is $FTLIB; the LIB parameter is not required for $FTLIB to be searched unless other files are named.

Multiple datasets may be named up to a limit of eight with dataset names separated by colons. Each dataset is repetitively scanned to permit all required system routines to be loaded. All datasets listed are automatically accessed; therefore, no ACCESS statement is required.

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.

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.

MAP=*op* Map control. If the MAP parameter appears, the loader produces a map of the loaded program on the specified dataset. MAP=OFF is the default. MAP=ON is equivalent

to MAP=FULL and produces a block list and an entry list including all cross references to each entry. MAP=PART is equivalent to MAP with no value and produces a block list only.

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 8 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.

NX No execution. Inclusion of this parameter inhibits execution of the loaded program.

C Compressed load. Inclusion of 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 increments only (optimal load).

OVL=*dir* Overlay load. The OVL parameter indicates an overlay load sequence is specified on *dir*. (See next section 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 card 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.

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

L=*ldn* Listing output. This parameter allows the user to specify the name of the dataset to receive the map output. 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.

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 a floating point error if used as a floating point operand. The value is 060505 000000 000777 000000 as 16-bit parcels.

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 file 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; i.e., the block map only.

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 1-13 in figure 9-1; the entry list includes items 14-17.

1. Job name
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. Name of load or library dataset containing modules to be loaded

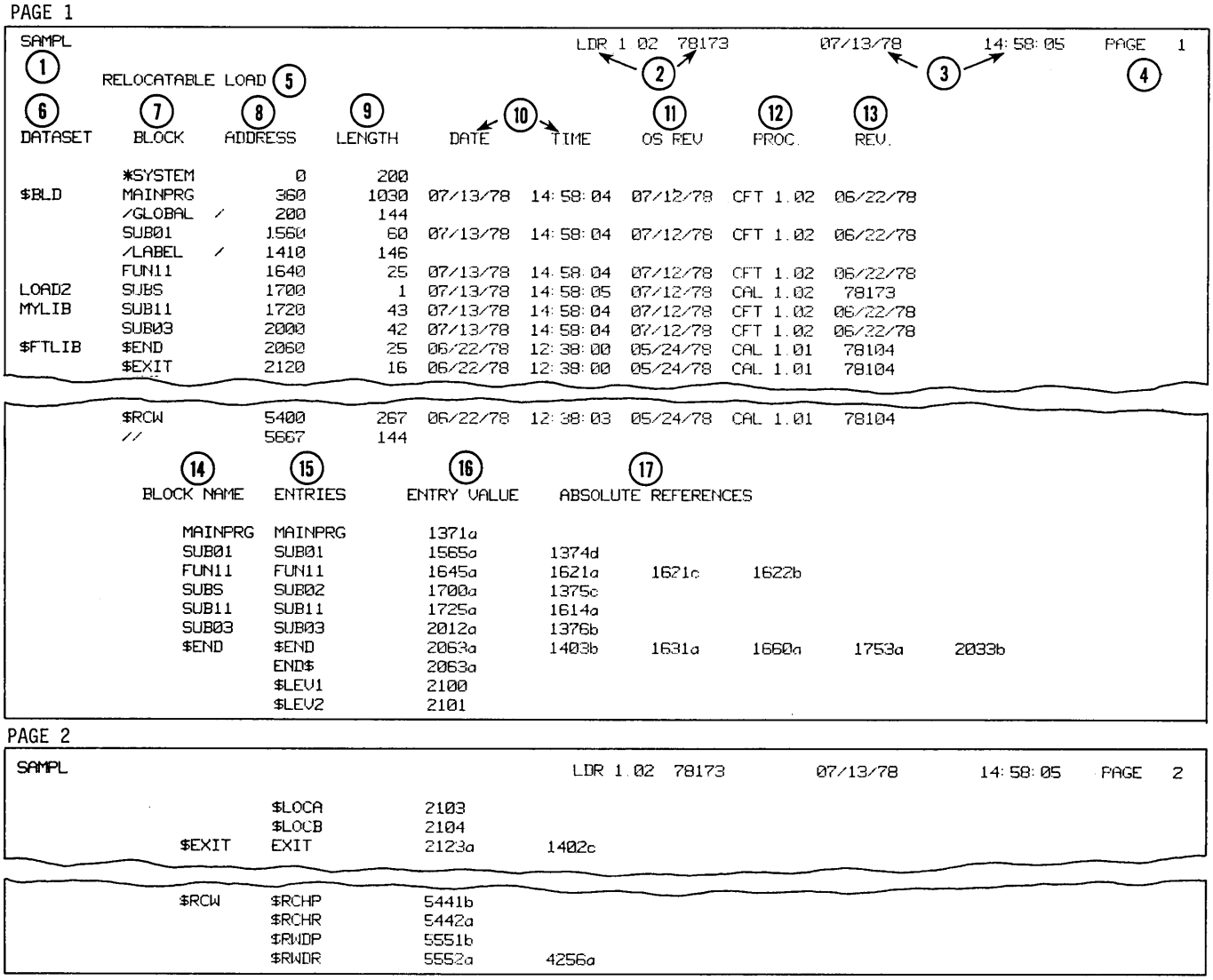


Figure 9-1. Example of a load map

7. Names of blocks loaded from the named dataset. These blocks may be common blocks (identified by the slashes around their names, e.g., /LABEL/) or may be 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).

8. Starting address of the block (in octal)
9. Word length of the block (in octal)
10. Date and time the object module was generated
11. Operating system revision date at the time the object module was generated
12. Name and revision level of the processor that generated the object module
13. Revision date of the processor that generated the object module
14. Name of program block referenced
15. Entry points in the program block
16. Word address, parcel address, or value of each entry point
17. Absolute parcel addresses of references to each entry point. Eight references are listed per line; some entry points have no references.

OVERLAY LOADER 10

Some 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.

OVERLAY STRUCTURE

Each 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; level 0 is the highest level.

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 (,), where is the primary number , and is in the range 1 through 999. All secondary overlays associated with a given primary 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 shows a sample overlay loading diagram. The primary and secondary overlays are shown in time sequence.

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.

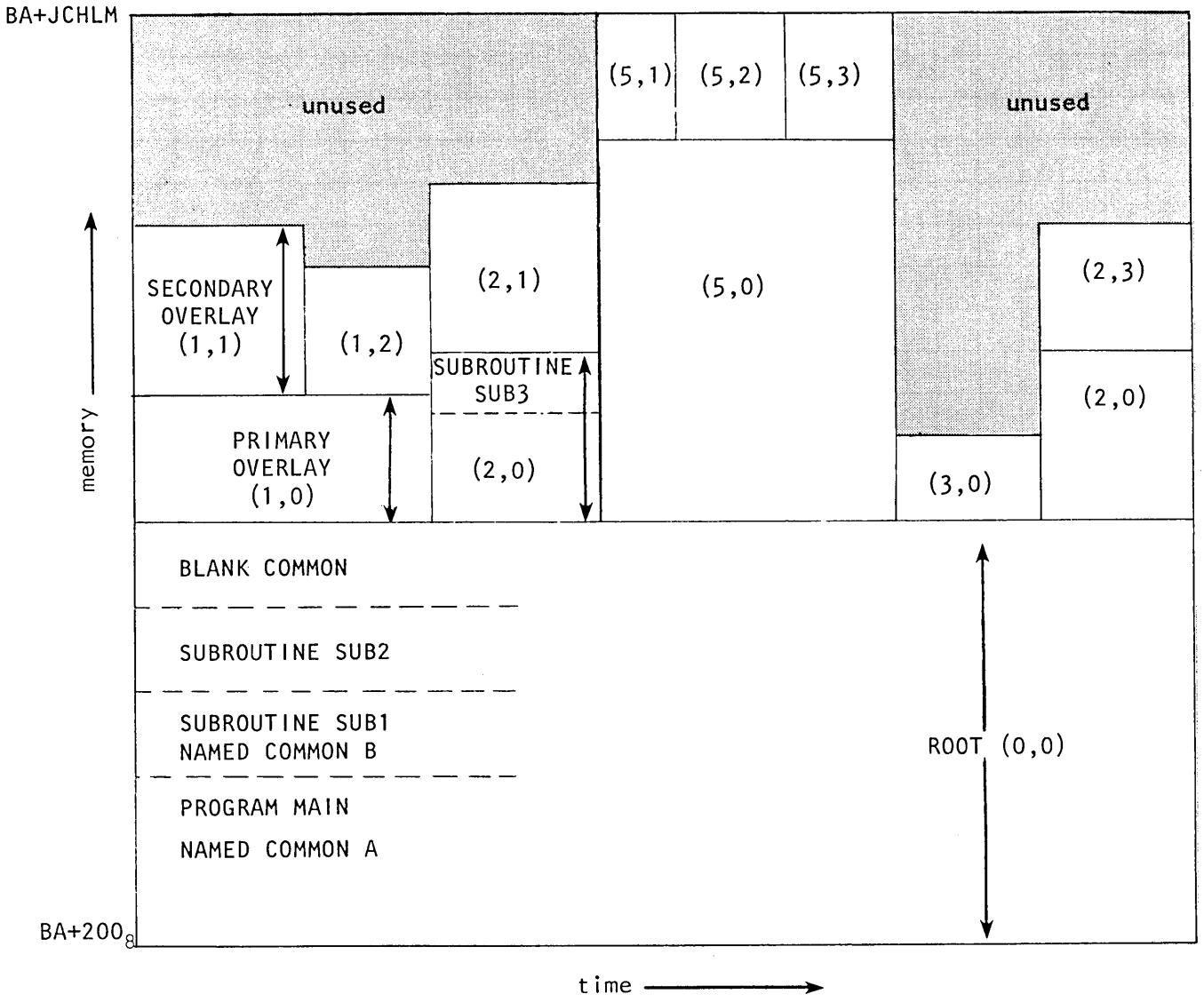


Figure 10-1. Example of overlay loading

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 above 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.

OVERLAY GENERATION

Overlay generation consists of a load operation in which the loader performs relocatable loading and writes the resulting binary image to disk as a named absolute overlay encompassing one binary record.

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 (i.e., OVL), then the loader reads overlay generation directives from $IN.

The format of the control statement is:

```
LDR,...,OVL=dir,....
```

OVERLAY GENERATION DIRECTIVES

An overlay generation directive consists of a keyword and a parameter. A blank or comma must separate the keyword from the parameter; two or more blanks or a period serve as a terminator.

FILE DIRECTIVE

The FILE directive indicates the dataset, *dn*, containing the routines to be loaded. 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 $BLD.

Format:

```
FILE,dn.
```

OVLDN DIRECTIVE

This directive names the dataset, *dn*, on which overlays are written. If no OVLDN directive is present, the default overlay binary dataset is assigned, i.e., $OBD. All overlays generated following an OVLDN directive reside as separate binary records on dataset *dn*. OVLDN directives may appear as often as desired.

Format:

```
OVLDN,dn.
```

ROOT DIRECTIVE

This directive defines programs, subroutines, 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*.

Format:

```
ROOT,member1,member2,...,membern.
```

POVL DIRECTIVE

This directive causes relocatable loading of the named blocks of a primary overlay with the name *pnumber*: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.

Format:

POVL,*pnumber*,*member*$_1$,*member*$_2$,...,*member*$_n$.

where *pnumber* is between 1 and 999.

SOVL DIRECTIVE

This directive causes relocatable loading of the named blocks of a secondary overlay with the name *pnumber*:*snumber*. The length of POVL (*pnumber*: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.

Format:

SOVL,*snumber*,*member*$_1$,*member*$_2$,...,*member*$_n$.

where *snumber* is between 1 and 999.

RULES FOR OVERLAY GENERATION

The following rules apply to overlay generation:

1. Root and overlay members are loaded from datasets named in FILE directives. Members are searched for in the most recently mentioned dataset only. Currently, the relocatable modules of all members 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 (*pnumber*) is generated; however, all secondary overlays (SOVL) associated with the *pnumber* must follow. The secondary overlay *snumbers* may be generated in any order.
5. An end-of-file ends the input of overlay directives.

6. Any directive other than FILE, OVLDN, ROOT, POVL, or SOVL causes a fatal error.

† 7. The list of members can be continued to another card by using a carat (^) character at the end of the listing card. The ^ does not replace a separator and must not appear within a member name.

† 8. Any number of cards can be used to name the members of an overlay. A maximum of 256 members may be named for an overlay.

EXAMPLE OF OVERLAY GENERATION DIRECTIVES

In the following example,

DSET1 contains routines SAM,TEST,CARROT,SUB1,MAIN,SUB2.

DSET2 contains routines NEW2,TURNIP,OVER,NEW1,JONES,ISABEL, MARY,UNAMIT...

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 overlays are generated on and can be called from LEV00.
ROOT,MAIN,SUB1 ^ ,SUB2.	The absolute binary of MAIN,SUB1,SUB2 is 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 following overlays are generated on and called from LEV12.
SOVL,2,NEW2.	The binary of NEW2 is named 001:002 and is binary record 1 on dataset LEV12.
POVL,2,TURNIP,UNAMIT.	The binary of TURNIP,UNAMIT is named 002:000 and is record 2 on dataset LEV12.
⋮	
<eof>	

† Deferred implementation

EXECUTION OF 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 card as for normal relocatable loads.

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	The number of characters in the name
L	Indicates left justification
dn	The dataset on which this overlay resides; name must be left justified and zero filled
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, *r*=6LRECALL may be entered.

CAL LANGUAGE CALL

A sample call sequence from a CAL program is as follows:

```
| Location | Result          | Operand      |
|----------|-----------------|--------------|
|          | 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'DNAME'L    |
```

where OVLDN contains the dataset name, PLEV contains the primary level, and SLEV contains the secondary level.

OVERLAY must appear on an EXT statement. If recall is desired the parameter is transmitted to W.OVERLAY-4.

Example:

```
| Location | Result          | Operand      |
|----------|-----------------|--------------|
| RECALL   | CON             | A'RECALL'L   |
|          | :               |    :         |
|          | :               |    :         |
|          | S4              | RECALL       |
|          | 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)
```

causes OVERLAY to search dataset LEV12 for the absolute binary named 002:000. The search proceeds from the current position of LEV12 to <*eof*>. If the search is not successful, OVERLAY rewinds the dataset and searches forward again to <*eof*>. If it finds overlay (2,0), it loads it and transfers control to the first encountered entry point. 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 the loader cannot find overlay (2,0) 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.

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 (and no 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 loader PDTs for each module.

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 replaced 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 that match any character other than a blank. For example, GE* specifies a group of modules that includes GET and GEM but not GE or GEMS.

† Deferred implementation

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 put out in alphabetical order according to their new names. In the absence of a SORT parameter, modules are put out in the order in which they were read originally from the input dataset(s).

Note that as an exception, modules specified explicitly (not as part of a group or a range) are put out at the time their COPY directives are encountered if the IMMED parameter appears on the BUILD control statement. 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 there are two or more modules with the same name or unless a range is specified in a COPY directive.

If two or more modules with the same name are anywhere among the several input datasets, the last of the modules to be read in 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 (,*last*) 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.

† Deferred implementation

BUILD CONTROL STATEMENT

The general format of the BUILD control statement is:

```
BUILD,I=ddn,L=ldn,OBL=odn,B=bdn,NBL=ndn,IMMED,SORT,NODIR.
```

Keyword parameters:

I=*ddn* Identifies the source of 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, all directives are taken from the $IN dataset, starting at its current position and stopping when an <*eof*> is read.

If I=*ddn*, all directives are taken from the specified dataset, *ddn*, stopping when an <*eof*> is read.

If I=0 or is omitted, no directives are read. The default 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* Identifies the list output dataset.

If the L parameter 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* Identifies 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, the first input step is skipped.

B=*bdn* Identifies 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*.

If B=0, the second input step is skipped.

BUILD stops at <*eof*>; *bdn* is not required.

NBL=*ndn* Identifies 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.

IMMED† Specifies that any modules named individually in COPY directives are to be output immediately upon encountering the COPY directive. The default is to output all modules at the end, after all the directives have been read.

SORT Specifies that all modules (except any that were affected by the IMMED parameter) are to be output in alphabetical order according to their (new) names. The default is to output 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.

Any of the following errors causes BUILD to abort:

- A module specified explicitly in a COPY or OMIT directive does not actually exist in the current input dataset
- A module specified explicitly in a COPY directive has already been selected for output
- Bad syntax in the BUILD control statement or in the directive dataset
- A nonexistent directive or control statement keyword
- A dataset name or module name is too long or contains illegal characters

† Deferred implementation

BUILD DIRECTIVES

BUILD is controlled through directives in a dataset defined by the I parameter in 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, a blank must separate the keyword from the first name; subsequent names (if any) in the list are separated from each other by commas. Extra blanks are optional everywhere.

A line may contain more than one directive; periods or semicolons may be used to separate directives on the same line from each other. A directive may not overflow 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. The last dataset in the list is established as the current input dataset, just as if it had been specified alone in the FROM directive. If no COPY or LIST directive follows, the last dataset will also be 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 is what allows the user to copy several datasets with 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 is 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.

Note that 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 belonged 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 instead, 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 $fn_1,fn_2,\ldots,fn_n$

where each fn_i may be one of the following:

- A single name, such as $AB@CDEF or CAB22, by which files can be explicitly prevented from being copied
- † A group file name, such as F$- or *AB**, by which files are prevented from being copied unless they are specified explicitly (i.e., singly) in a COPY directive

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.

† Deferred implementation

COPY DIRECTIVES†

COPY directives cause BUILD to select the specified modules for copying from the 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 an individual module that is not in the current input dataset, a diagnostic message is included in the list output and BUILD aborts.

Format:

COPY $fn_1,fn_2,\ldots,fn_n$

where 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.

---

† Deferred implementation

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 to 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.
SAVE,DN=$NBL,PDN=MYLIB.
  :
  :
```

- Adding one or more modules to an already existing library dataset, again taking the input from $BLD.

  Control statements:

```
ACCESS,DN=$OBL,PDN=MYLIB.
BUILD.
SAVE,DN=$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 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=MYLIB.
   :
```

  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=$OBL,PDN=MYLIB.
BUILD,I,B=0.
SAVE,DN=$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 on the system text, $SYSTXT. The code generated by the macros represents a call to a system task or a system-provided subroutine, or 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 and 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.

A parameter shown in all UPPERCASE letters must be coded literally as shown. A parameter presented in *italics* must be supplied with a value, symbol, expression, or 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 is continued on the next line by placing a comma in column one of the next line and resuming the parameter list in column two, with no intervening blanks at the end of the first line.

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 function code octal value is stored in register S0; S1 and S2 provide optional arguments. The function 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 system action request macros can be divided into three main classes: those involved in job control, those related to dataset management, and those representing requests for time or date.

JOB CONTROL

Several system action request macros allow the user to set operating characteristics and control job processing. These include MEMORY, MESSAGE, MODE, SWITCH, JTIME, RECALL, DELAY, ABORT, and ENDP.

MEMORY - REQUEST MEMORY

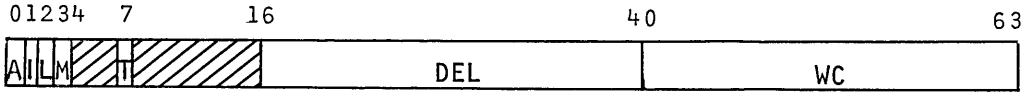
The amount of memory assigned to the job may be determined or changed by the memory request.

Format:

Location	Result	Operand
	MEMORY	*address*

address A symbol or an A, S, or T register that contains the address

The format of the word at location *address* is as follows:



A Abort-disable flag. Unless this flag is set by the user, the job is aborted if filling the request would cause the job to exceed its maximum allowable memory.

I† — Immediate flag. If set by the caller, the system returns with whatever memory is immediately available; no delay can occur. Currently, immediate flag is ignored.

L — Limit flag. The system sets this flag when the job has received the maximum allowable amount of memory.

M — Mode flag. The system sets this flag if it is controlling memory management for this job.

T — 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.

DEL — Deletion pointer. If the caller wants an increase in memory, DEL must be zero. If the caller wants a decrease in memory, DEL must contain the address relative to the user's BA of the beginning of the area to be deleted.

WC — Word count. Here, the caller must supply the absolute number of words to be added to or deleted from the user area. If WC=0, no action is taken other than to return the user's field length as described under RETURN, below.

Space is always added at the end of the program area. Deletions, on the other hand, are under full control of the user; DEL may point to any location between the job communication block and HLM (where HLM is the value in the JCHLM field of the Job Communication Block).

RETURN — In the memory request word, L and M may be set by the system as described above. If the total request is allocated, the system sets WC to the current total number of words in the user's field length and sets DEL to 0. If the system is unable to allocate all the memory that was requested, DEL contains the number of words that were not supplied. The total number of words in the user's field length does not include the Job Table Area and may or may not include the I/O buffers and tables.

If the user area is expanded, the additional memory is set to an installation-defined value before control returns to the user.

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.

---

† Deferred implementation

Format:

Location	Result	Operand
	MESSAGE	*address,loc*

address A symbol or an A, S, or T register that contains the address

loc Destination for message. May be any of the following:

U User logfile only
S System logfile only
US User and system logfiles; default, if *loc* is blank.

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*

m A single octal digit having one of the following values:

1 or 2 No interrupt on floating point errors

3 or 4 Interrupt on floating point errors

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

RECALL - RECALL JOB UPON I/O REQUEST COMPLETION

This function removes the job from processing. It does not become a candidate for processing until the previously issued I/O request for the specified dataset is completed or partially completed, i.e., the job is resumed when another block of data is transferred to or from the user's buffer or when the I/O requested is completed.

Format:

Location	Result	Operand
	RECALL	*address*

address Symbolic address of the ODN or DSP for this dataset or an A, S, or T register containing the ODN or DSP address.

DELAY - DELAY JOB PROCESSING

The job is removed from execution and is not made 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 that contains the address

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. If no such statement exists, the job is terminated.

Format:

Location	Result	Operand
	ABORT	

ENDP - END PROGRAM

The ENDP request is used for normal termination of the current program. Processing resumes with the next job control statement. If no such statement exists, the job is terminated.

Format:

Location	Result	Operand
	ENDP	

DATASET MANAGEMENT

The system action request macros involved with dataset management allow the user to open datasets, set up tables, and close, release, and dispose datasets.

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.

Format:

Location	Result	Operand
	DSP	*dn,first,nb*

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

OPEN - OPEN DATASET

OPEN prepares a dataset for processing.

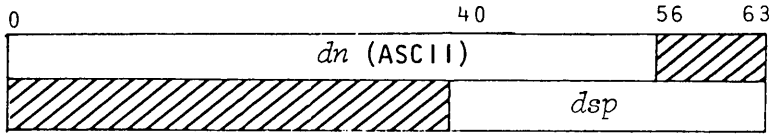
Format:

Location	Result	Operand
	OPEN	*dn,pd*

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 (figure 2-1).

The *dn* becomes the symbolic address of the ODN as well as being the dataset name. 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 S2) containing the ODN address.



Field	Word	Bits	Description
ODDN	0	0-55	Dataset name
ODDSP	1	40-63	DSP pointer negative or zero - negative DSP offset positive - absolute address of DSP

Figure 2-1. Open Dataset Name Table (ODN)

pd Processing direction.

I Dataset opened for input
O Dataset opened for output
IO Dataset opened for input/output

pd may alternatively be an A, S, or T register (not S2) with bit 0 set for input and/or bit 1 set for output.

If *dsp* is negative or zero, the OPEN call returns the negative DSP offset in *dsp*. The actual DSP address is equal to (JCDSP) - negative DSP offset, where (JCDSP) is the value in the JCDSP field of the Job Communication Block. Note that 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 *dsp* is positive and greater than zero, OPEN assumes *dsp* is the address of a 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.

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 S2; S1 contains processing direction information.

Location	Result	Operand	Comment
1	10	20	35
	OPEN	S2,S1	

CLOSE - CLOSE DATASET

CLOSE terminates I/O processing on a dataset and flushes buffers if needed.

Format:

Location	Result	Operand
	CLOSE	*dn*

dn Dataset name. Symbolic address of the ODN for this dataset or an A, S, or T register containing the address of the ODN.

RELEASE - RELEASE DATASET TO SYSTEM

The dataset whose DSP address is at the location specified in the macro call is returned to the system. The dataset is closed and buffers and DSP are 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.

Format:

Location	Result	Operand
	RELEASE	*address*

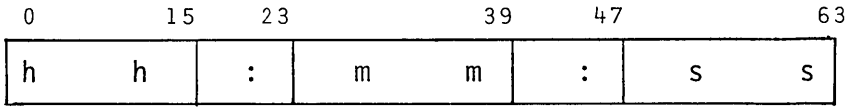
address Symbolic address of the ODN or DSP for this dataset or an A, S, or T register containing the ODN or DSP address.

TIME AND DATE REQUESTS

Several system action request macros inform the user of the current time or date and the Julian date.

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:



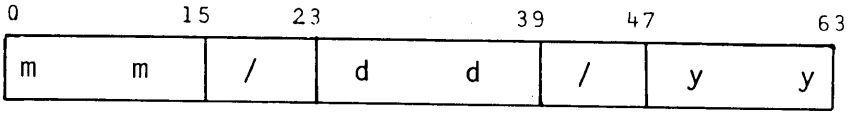
Format:

Location	Result	Operand
	TIME	*address*

address A symbol or an A, S, or T register that contains the address

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:



The order can be changed to day, month, and year (the European format) through an installation parameter.

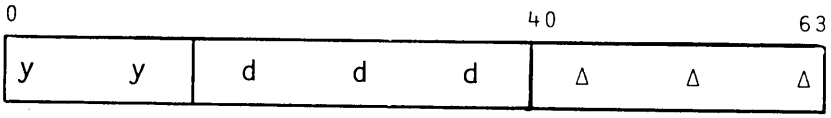
Format:

Location	Result	Operand
	DATE	*address*

address A symbol or an A, S, or T register that contains the address

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:



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 address

MISCELLANEOUS

This section describes macros that do not fit in the other categories. Currently only the SYSID macro is in this category.

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 System Task Processor (STP) revision date in ASCII and the second contains the STP assembly date in ASCII.

Format:

Location	Result	Operand
	SYSID	*address*

address A symbol or an A, S, or T register that contains the address

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.

There are two main categories of logical I/O macro instructions: read/write macros and positioning macros.

READ/WRITE

The 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.

READ/READP - READ WORDS

The READ and READP macros transfer words of data resident on a dataset into the user's data area.

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.

Formats:

Location	Result	Operand
	READ	*dn,uda,ct*

Location	Result	Operand
	READP	*dn,uda,ct*

dn Dataset name (symbolic address of the ODN for this dataset) or an A, B, or S register containing the DSP address or negative DSP offset.

uda User data area 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

READC/READCP - READ CHARACTERS

The READC and READCP macros transfer characters 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 <*eor*> 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 <*eor*> or by satisfying the character count, whichever occurs first.

Formats:

Location	Result	Operand
	READC	*dn,uda,ct*

Location	Result	Operand
	READCP	*dn,uda,ct*

dn Dataset name (symbolic address of the ODN for this dataset) or an A, B, or S register containing the DSP address or negative DSP offset.

uda User data area 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

WRITE/WRITEP - WRITE WORDS

The WRITE macro generates a return jump to the $WWDR subroutine. Words are written from the user's data area. An end of record is written following each WRITE.

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.

Formats:

Location	Result	Operand
	WRITE	*dn,uda,ct*

Location	Result	Operand
	WRITEP	*dn,uda,ct*

dn Dataset name (symbolic address of the ODN for this dataset) or an A, B, or S register containing the DSP address or negative DSP offset

uda User data area 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

To write just an end of record, the WRITE macro with word count of 0 is used.

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 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.

Formats:

Location	Result	Operand
	WRITEC	*dn,uda,ct*

Location	Result	Operand
	WRITECP	*dn,uda,ct*

dn Dataset name (symbolic address of the ODN for this dataset) or an A, B, or S register containing the DSP address or negative DSP offset

uda User data area 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

To write just an end of record, the WRITEC macro with word count of 0 is used.

WRITEF - WRITE END OF FILE

The WRITEF macro generates a return jump to the $WEOF subroutine causing an end of record (if not previously written) and an end of file to be written.

Format:

Location	Result	Operand
	WRITEF	*dn*

dn Dataset name (symbolic address of the ODN for this dataset) or an A, B, or S register containing the DSP address or negative DSP offset

WRITED - WRITE END OF DATA

The WRITED macro generates a return jump to the $WEOD subroutine causing an end of record (if not previously written), an end of file (if not previously written), and an end of data to be written.

Format:

Location	Result	Operand
	WRITED	*dn*

dn Dataset name (symbolic address of the ODN for this dataset) or an A, B, or S register containing the DSP address or negative DSP offset

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.

REWIND - REWIND DATASET

The REWIND macro generates a return jump to the $REWD subroutine causing the dataset to be positioned at beginning of data.

Format:

Location	Result	Operand
	REWIND	*dn*

dn Dataset name (symbolic address of the ODN for this dataset) or an A, B, or S register containing the DSP address or negative DSP offset

BKSP - BACKSPACE RECORD

The 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.

Format:

Location	Result	Operand
	BKSP	*dn*

dn Dataset name (symbolic address of the ODN for this dataset) or an A, B, or S register containing the DSP address or negative DSP offset

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.

Format:

Location	Result	Operand
	BKSPF	*dn*

dn Dataset name (symbolic address of the ODN for this dataset) or an A, B, or S register containing the DSP address or negative DSP offset

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 RCWs and BCWs.

Format:

Location	Result	Operand
	GETPOS	*dn*

dn Dataset name (symbolic address of the ODN for this dataset) or an A, B, or S register containing the DSP address or negative DSP offset

SETPOS - POSITION DATASET

The SETPOS macro generates a return jump to the $SPOS subroutine. The dataset is positioned at the word address 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).

Format:

Location	Result	Operand
	SETPOS	*dn,pos*

dn Dataset name (symbolic address of the ODN for this dataset) or an A, B, or S register containing the DSP address or negative DSP offset

pos 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.

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, and DISPOSE macros involved in permanent dataset management use the PDD table. Thus, the PDD macro must accompany the use of the permanent dataset management macros.

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 nonexecutable and must accompany the use of the ACCESS, SAVE, DELETE, ADJUST, or DISPOSE 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*,R=*rd*,W=*wt*, M=*mm*,MSG={ON / OFF},AB={ON / OFF},UQ={ON / OFF}.

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.

ID=*uid* User identification; 1-8 alphanumeric characters assigned by the dataset creator.

MF=*mf* Mainframe identifier; 2 alphanumeric character identification. The default is the mainframe of job origin.

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 two-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.
- BC 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.

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 two-character alpha code which describes the destination of the dataset as follows:

- IN Input to mainframe. Dataset is placed in the job input queue for the mainframe designated 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.

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 default is no maintenance control word.

MSG={ON / OFF} Normal completion message suppression indicator. The default is OFF.

ON Indicator is set
OFF Indicator is cleared

AB={ON / OFF}† Abort indicator; abort job if an error is encountered. The default is ON.

ON Indicator is set
OFF Indicator is cleared

† Deferred implementation

UQ={ON / OFF} 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 is multi-read access if the read control word is specified.

PERMANENT DATASET MANAGEMENT

The user may access, save, adjust, dispose, 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 the ASSIGN† macro and any logical I/O macros for the 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 two-fold function:

1. Creation of an initial edition of a permanent dataset, or
2. Creation of an additional edition of a permanent dataset.

† Deferred implementation

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 maintenance permission 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 <*eod*> for the dataset. A dataset must be accessed with write permission within a job before an ADJUST may be issued.

Format:

Location	Result	Operand
	ADJUST	*pddtag*

pddtag Address of PDD macro call

DISPOSE - DISPOSE DATASET

The DISPOSE macro immediately places a dataset in the output queue for staging to the specified front-end computer system. This does not delay job processing. The DISPOSE macro may also be used to release a dataset or to alter its disposition characteristics.

Format:

Location	Result	Operand
	DISPOSE	*pddtag*

pddtag Address of PDD macro call

APPENDIX SECTION

CONTENTS
APPENDIX SECTION

JOB USER AREA

A

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.

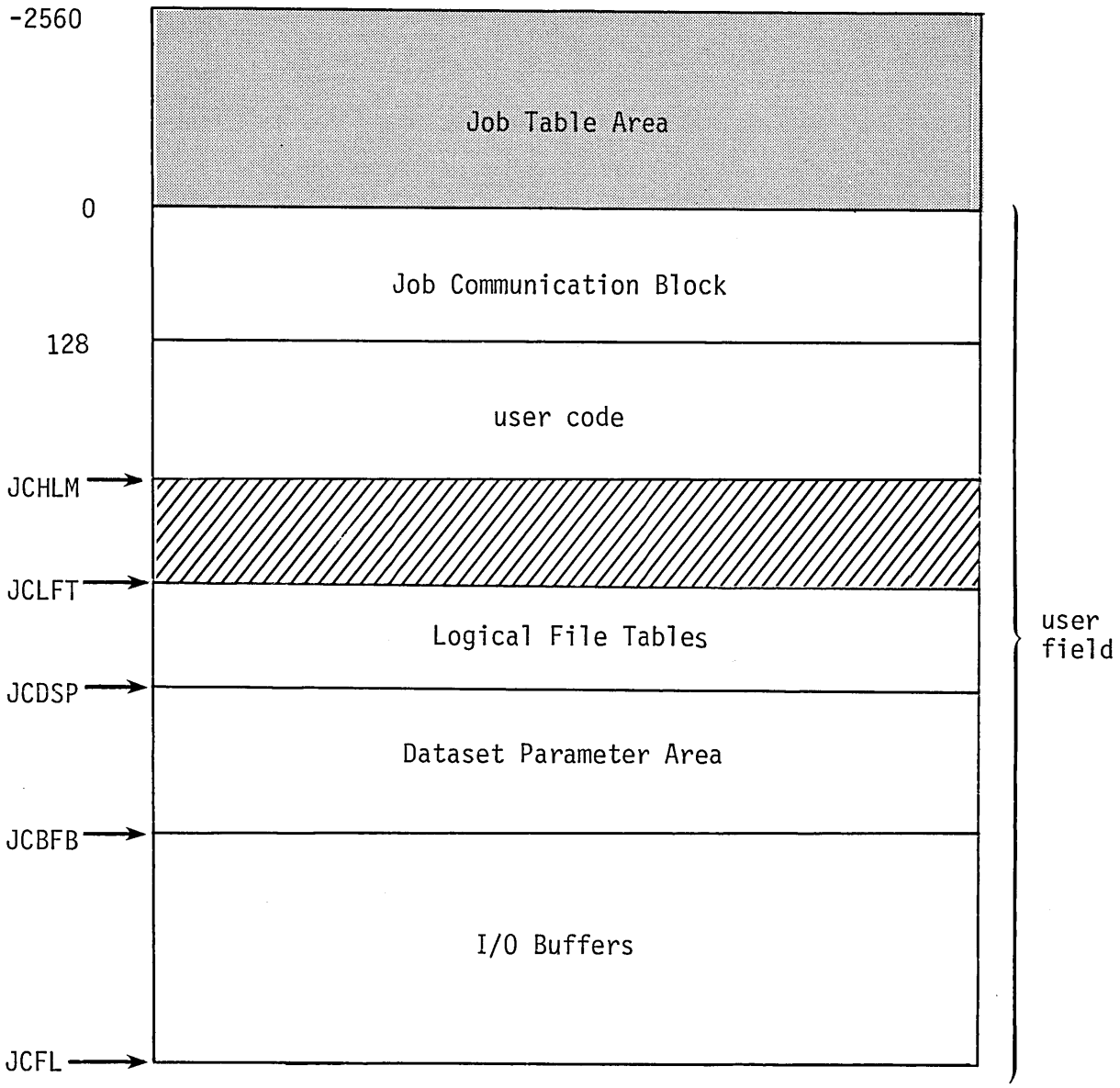


Figure A-1. User area of memory for a job

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 Job Table Area contains job-related information such as accounting data; a JXT pointer; sense switches; area for saving B, T, and V register contents; control statement, logfile, and EXU DSPs; a logfile buffer; and a Dataset Name Table (DNT) containing an entry for each dataset used by the system.

JOB COMMUNICATION BLOCK - JCB

Following the JTA is a 128-word block referred to as the Job Communication Block (JCB). The 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.

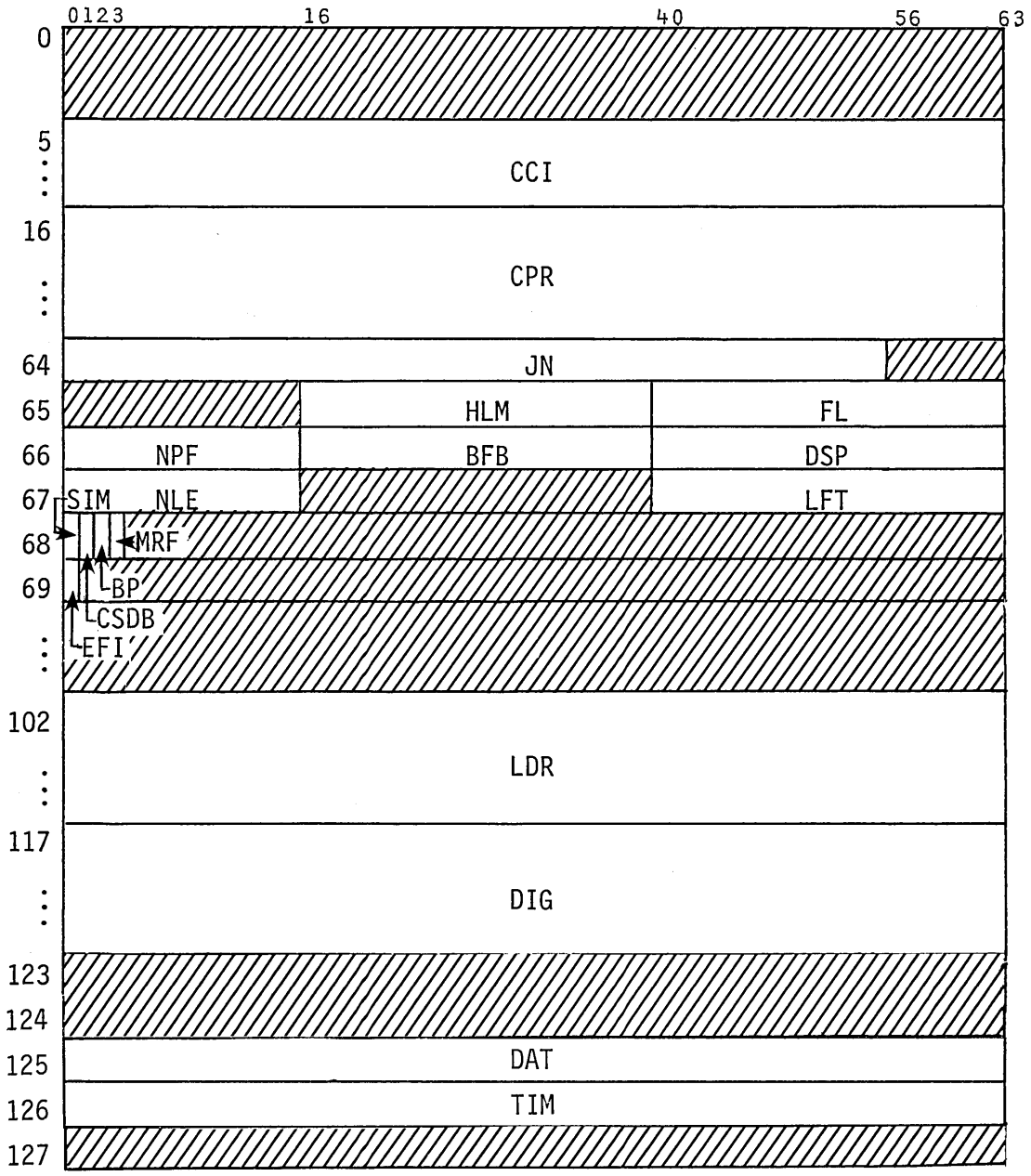


Figure A-2. Job Communication Block

Field	Word	Bits	Description
JCCCI	5-15	0-63	Current control card image, 8 characters per word
JCCPR	16-63	0-63	Control statement parameters as cracked from JCCCI
JCJN	64	0-55	Job name

Field	Word	Bits	Description
JCHLM	65	16-39	High limit of user code
JCFL	65	40-63	Field length
JCNPF	66	0-15	Number of physical buffers and datasets
JCBFB	66	16-39	Base of I/O buffers
JCDSP	66	40-63	Base of LFT
JCSIM	68	0	Simulator flag
JCCSDB	68	1	CSP debug flag
JCBP	68	2	Breakpoint flag
JCMRF	68	3	Memory request flag
JCEFI	69	0	Enable floating point interrupt flag
JCLDR	102-116	0-63	Unsatisfied externals
JCDIG	117-123	0-63	Reserved for diagnostics
JCDAT	125	0-63	Date of absolute load module generation
JCTIM	126	0-63	Time of absolute load module generation

LOGICAL FILE TABLE - LFT

The LFT contains an entry for each dataset name and alias. Each entry points to the DSP for a dataset. Figure A-3 illustrates an LFT for a dataset.

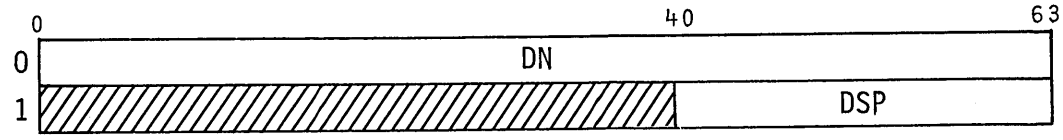


Figure A-3. Logical File Table (LFT)

Field	Word	Bits	Description
LFDN	0	0-63	Dataset name or alias
LFDSP	1	40-63	DSP address

DATASET PARAMETER AREA - DSP

Information concerning the status of a particular dataset and the 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.

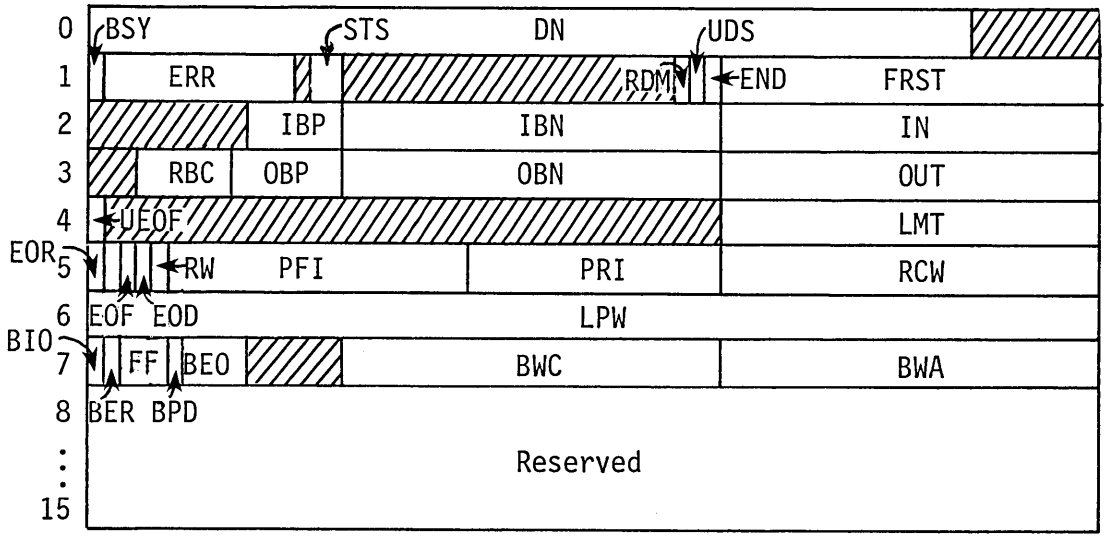


Figure A-4. Dataset Parameter Area (DSP)

Field	Word	Bits	Description
DPDN	0	0-55	Dataset name
DPBSY	1	0	Busy flag (circular I/O)
DPERR	1	1-12	Error flags:
DPEOI			Bit 01 End of data on read Write past allocated disk space on write
DPENX			Bit 02 Dataset does not exist
DPEOP			Bit 03 Dataset not open
DPEPD			Bit 04 Invalid processing direction
DPEBN			Bit 05 Block number error
DPEDE			Bit 06 Unrecovered data error
DPEHE			Bit 07 Unrecovered hardware error
DPERW			Bit 08 Attempted read after write or past end of data
DPEPT			Bit 09 Dataset prematurely terminated on read
			Bits 10 through 12 reserved

Field	Word	Bits	Description
DPSTS	1	14,15	Status 00 Closed 10 Open for input (I) 01 Open for output (O) 11 Open for I/O
DPRDM	1	37	Random dataset flag 0 Sequential dataset 1 Random dataset
DPUDS	1	38	Undefined dataset structure 0 COS blocked dataset structure 1 Undefined dataset structure
DPEND	1	39	Write end of data flag
DPFRST	1	40-63	Address of first word of buffer
DPIBP	2	10-15	Bit position in current input word (logical I/O)
DPIBN	2	16-39	Block number, read request System reads from block number until buffer is filled. The next block number is then in word 2. If the block number is $2^{24}-1 = 77777777_8$, the last block of the dataset is read.
DPIN	2	40-63	Address of current input word
DPRBC	3	3-9	Remaining bit count
DPOBP	3	10-15	Bit position in current output word
DPOBN	3	16-39	Block number, write request System writes from block number until buffer is empty. The next block number is then in word 3.
DPOUT	3	40-63	Address of current output word
DPUEOF	4	0	Uncleared end of file
DPLMT	4	40-63	Address of last word + 1 of buffer; LMT minus FRST defines buffer size
DPEOR	5	0	End of record flag
DPEOF	5	2	End of file flag
DPEOD	5	3	End of data flag
DPRW	5	4	Previous operation read/write flag 0 Read 1 Write

Field	Word	Bits	Description
DPPFI	5	5-24	Previous file index; backward index to block
DPPRI	5	25-39	Previous record index; backward index to block containing previous end of record
DPRCW	5	40-63	Control word access Previous RCW address in write mode Next RCW in read mode
DPLPW	6	0-63	Last partial word; used for character mode I/O
DPBIO	7	0	Buffer in/out busy 0 Buffer in/out operation complete 1 Buffer in/out operation not complete
DPBER	7	1	Buffered I/O error flag
DPFF	7	2-4	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	5	Processing direction 0 Read 1 Write
DPBEO	7	6-9	Termination condition 00 Partial 10 Record 12 File, write only 16 Data, write only
DPBWC	7	16-39	Word count; number of words at DPBWA to read or write. Actual number of words read when BIO=0 on read.
DPBWA	7	40-63	Word address

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.

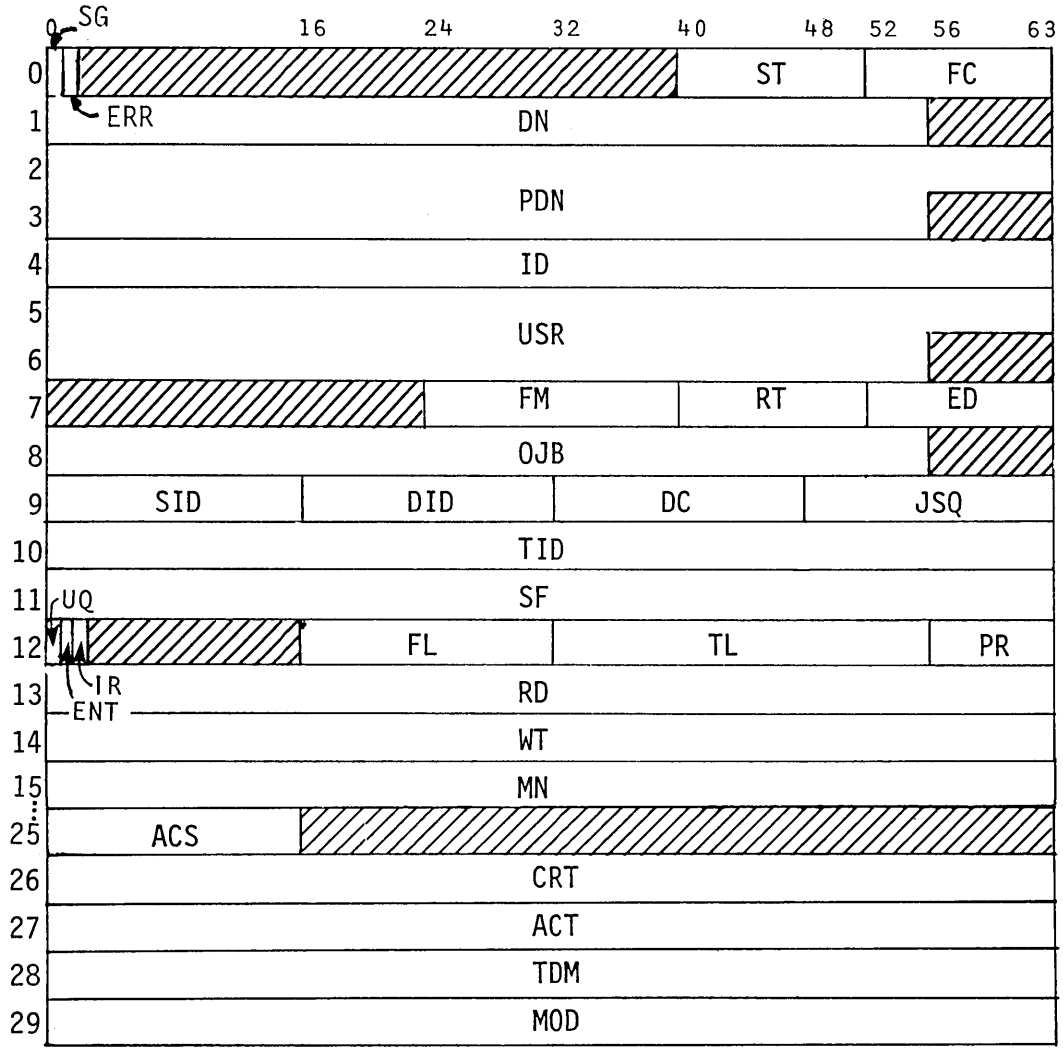


Figure A-5. Permanent Dataset Definition Table (PDD)

Note that for delete requests, the PDD consists of only the first two words (words 0 and 1) of figure A-5.

Field	Word	Bits	Description
PMSG	0	0	Normal completion message suppression indicator
PMERR	0	1	Error message suppression indicator
PMST	0	40-51	Return status
PMFC	0	52-63	Function code (octal):
			PMFCSU=10 Save user dataset
			PMFCSI=12 Save input dataset
			PMFCSO=14 Save output dataset
			PMFCAU=20 Access user dataset
			PMFCAI=26 Access spooled dataset
			PMFCAO=26 Access spooled dataset
			PMFCDU=30 Delete user dataset
			PMFCDI=36 Delete spooled dataset
			PMFCDO=36 Delete spooled dataset
			PMFCPG=40 Page request
			PMFCLU=50 Load user dataset
			PMFCLI=52 Load input dataset
			PMFCLO=54 Load output dataset
			PMFCRL=60 PDS/Release request
			PMFCPN=70 PDN request
			PMFCDT=100 Dump time request
			PMFCDQ=110 Dequeue SDT
			PMFCEA=120 Queue SDT to available queue
			PMFCEI=122 Queue SDT to input queue
			PMFCEO=124 Queue SDT to output queue
PMDN	1	0-55	Dataset name
PMPDN	2,3	0-63	Permanent dataset name - format 1 PDDs; 1-15 characters
PMNPG	2	32-47	Number of pages - format 2 PDDs
PMBPG	2	48-63	Beginning page number - format 2 PDDs

Field	Word	Bits	Description
PMSDT	2	24-47	SDT address; format 4 PDDs
PMSQJ	2	48-63	Job sequence number; format 4 PDDs
PMBUF	3	40-63	Buffer address
PMID	4	0-63	User identification
PMUSR	5,6	0-63	User number; 1-15 characters
PMFM	7	24-39	Format designator: 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
PMFL	12	16-31	Field length/512

Field	Word	Bits	Description
PMTL	12	32-55	Time limit
PMPR	12	56-63	Priority
PMRD	13	0-63	Read permission control word
PMWT	14	0-63	Write permission control word
PMMN	15	0-63	Maintenance permission control word
PMACS	25	0-15	Number of accesses (load saved datasets only)
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)

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 CHARACTER	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	
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	Upper-case letters
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		
b	142	12-0-2		
c	143	12-0-3		
d	144	12-0-4		
e	145	12-0-5		
f	146	12-0-6		
g	147	12-0-7		
h	150	12-0-8		
i	151	12-0-9		
j	152	12-11-1		
k	153	12-11-2		
l	154	12-11-3		
m	155	12-11-4		
n	156	12-11-5		Lower-case letters
o	157	12-11-6		
p	160	12-11-7		
q	161	12-11-8		
r	162	12-11-9		
s	163	11-0-2		
t	164	11-0-3		
u	165	11-0-4		
v	166	11-0-5		
w	167	11-0-6		
x	170	11-0-7		
y	171	11-0-8		
z	172	11-0-9		
{	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)

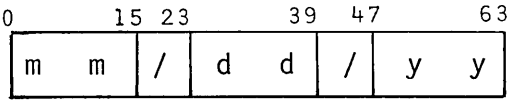
SYSTEM FUNCTION CODES C

Many of the macro instructions (see part 3) generate function codes that are calls to the operating system. The function code octal value is stored in register S0; S1 and S2 provide optional arguments. S1 conventionally contains the address of a user table, if required. The function is enabled when the program exit instruction is executed.

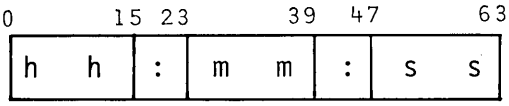
When the request completes, the user's S0 is set to zero to indicate the request was completed without error. If an error is encountered, the job normally aborts with appropriate messages issued to the logfile. For some errors, however, an error code is placed in the user's S0 and the user is allowed to continue processing. If the Control Statement Processor (CSP) is executing as the user, S0 returns an error code for all but a few fatal errors, which causes CSP to be reloaded.

System-defined mnemonic values are supplied for the function codes in S0 and should be used for all functions, as supplied in the list that follows.

FUNCTION CODE	TASK DESCRIPTION
F$ADV	Advance job. The current job step is terminated and the job is advanced to the next control statement.
F$ABT	Abort job. The job is advanced to the EXIT control statement, if one exists.
F$DAT	Get current date. The current date in ASCII format is returned at the location specified in S1 in the following format:



F$TIM	Get current time. The current time in ASCII format is returned at the location specified in S1 in the following format:



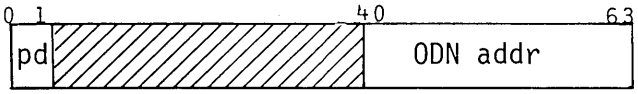
F$MSG	Enter message in logfile. A message beginning at the location specified by S1 is written to the logfile. S2 is used to determine the logfile to which the message is written.

(S2)	Significance
1	User logfile only
2	System logfile only
3	System and user logfiles

The message is 1-80 characters and is terminated by a zero byte.

F$RCL	Dataset recall. The job is removed from execution until another block of data has been transferred without error or until I/O is complete on the dataset specified. S1 contains the ODN or DSP address (see Appendix A for a description of the DSP and part 3, page 2-7 for a description of the ODN).

FUNCTION CODE	TASK DESCRIPTION
F$TRM	Terminate job. The job is terminated normally and its resources are returned to the system.
F$SSW	Set pseudo sense switch. S1 contains the number of the switch to be set.
F$OPN	Open dataset. S1 contains processing direction in bits 0 and 1 and the address of the Open Dataset Name (ODN) table (see part 3, page 2-7). Bits 40-63 of S1 contain the address.



An open call does the following processing for the dataset whose name is in the first word of the ODN table:

1. A DNT entry is created in the user's JTA if one does not already exist for this dataset.
2. An LFT entry is created for this dataset if one does not already exist.
3. A DSP entry is created for this dataset if one does not already exist.
4. A buffer is allocated for this dataset if one does not already exist.

Steps 2, 3, and 4 may result in moving existing LFT entries, DSP entries, and buffers. Additional user field is not allocated if insufficient room exists for adding the LFT, DSP, or buffer. Parameters in the JCB of the user field reflect any movement of these tables.

5. The negative DSP offset:

   = DSP base address (JCDSP) - DSP entry address

   is returned in bits 40-63 of word 2 of the ODN table.
6. The DNT and DSP are modified to reflect the processing direction requested.

FUNCTION CODE	TASK DESCRIPTION

NOTE

If word 2 of the ODN, bits 40-63, already contains a positive, nonzero address, then the user is requesting that a user's own DSP and buffer be used. The address in the ODN points to the DSP, and the DSP and buffer must be contained in the user field below the high limit of user code (JCHLM). In this case, steps 2, 3, and 4 are omitted.

Processing Direction

01	Output
10	Input
11	Input/Output

F$MEM

Request memory. The amount of memory assigned to a job may be determined or changed. S1 contains the address of the memory request word. The memory request word has the following format:

A Abort-disable flag. Unless this flag is set by the user, the job is aborted if filling the request would cause the job to exceed its maximum allowable memory.

I† Immediate flag. If set by the user, only the memory that is immediately available is assigned to the user.

L Limit flag. The system sets this flag when it has assigned the maximum allowable amount of memory to the user.

M Mode flag. The system sets this flag if it is performing automatic memory management for the job.

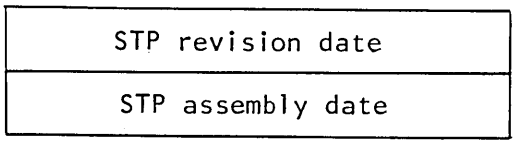
DEL Deletion pointer. If the user wants an increase in memory, DEL must be zero. If the caller wants a decrease in memory, DEL must contain the beginning address of the area to be deleted.

† Deferred implementation

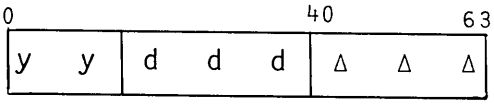
FUNCTION CODE	TASK DESCRIPTION
	WC Word count. Here, the user must supply the absolute number of words to be added to or deleted from the user's field length. Any words added to the user's field length are added to its high-address end. If WC=0, no action is taken other than to return the user's field length in WC.
F$LBN	Return last block number. S1 contains the address of the Open Dataset Name Table (ODN) (see part 3, page 2-7). On return, S2 contains the block number of the last block of the dataset. S2 contains -1 (all bits set) if the dataset is empty.
F$CLS	Close dataset. S1 contains the address of the Open Dataset Name Table (ODN). A close call does the following processing for the dataset whose name is in the first word of the ODN table: 1. End of data is written on a sequential blocked dataset if the dataset is in write mode. 2. If the dataset is in write mode and is a blocked dataset, data in the buffer is flushed to disk. 3. The buffer for the dataset is released. 4. The DSP for the dataset is released. 5. Any LFT entries for the dataset are released. 6. The DNT entry for the dataset is updated to indicate that the dataset is closed.
F$DNT	Create local dataset. S1 contains the address of the Dataset Definition List (DDL) in the user field. This call creates a Dataset Name Table (DNT) for the dataset if one does not already exist. If the DDNFE flag is set to 1, the DNT is searched for an existing dataset with the specified name. On return to the user,

FUNCTION CODE	TASK DESCRIPTION
	(S0)=0 if the dataset already exists, otherwise (S0)≠0. If the dataset already exists for this job, then the dataset must be closed. Parameters from the DDL are inserted in the DNT.
F$MDE	Set exchange package mode. S1 contains the address of the word containing the new mode setting. (See part 2, page 2-2 for mode settings.)
F$GNS	Get next control statement. Copy one card image from the control statement buffer to the address specified in S1. Error code EREFR (1) is returned in S0 if an end of file is encountered on the control statement file.
F$EXU	Load binary dataset at location specified in the PDT in the user field and begin execution. The address of a word containing the name of the dataset is in S1. Additional memory is allocated for the job if required to load the binary dataset.
F$RLS	Return dataset whose ODN table address is specified in S1. The dataset is closed and disposed of according to the disposition code contained in the DNT entry for this dataset. The dataset is no longer available to the job.
F$PDM	Permanent dataset management request. S1 contains address of the Permanent Dataset Definition (PDD) table. The contents of the PDD depend on the function requested. (See Appendix A for the formats of the PDD table.)
F$RDC	Read disk circular. S1 contains the DSP address. The error bits and the busy bit in the DSP must be monitored by the caller. Automatic recall is requested if bit 0 of S1 is set. If more than one sector is being read, automatic recall returns control at half buffer boundary.

FUNCTION CODE	TASK DESCRIPTION
F$WDC	Write disk circular. S1 contains the DSP address. The error bits and the busy bit in the DSP must be monitored by the caller. Automatic recall is requested if bit 0 of S1 is set. If more than one sector is being written on, automatic recall returns control at half buffer boundary.
F$GRN	Get system revision numbers. S1 contains the address of a two-word table. Information is returned in ASCII format, left justified and blank filled as follows:



F$DIS	Dispose dataset. S1 contains the PDD address.
F$JDA	Get current Julian date in ASCII format. The date is returned at the location specified in S1. The date is returned as follows:



F$JTI	Return accumulated CPU time for the job in the location specified by S1. The time is expressed in floating point seconds.
F$ACT	Accounting information from the JXT and the JTA is returned at locations starting with the address in S1.
F$SPS	Set P register and suspend user. The new program address is in S1.
F$CSW	Clear sense switch. S1 contains the switch number to be cleared.
F$TSW	Test sense switch. S1 contains the switch number to be tested.

On return (S1) ≠ 0 if sense switch is set
= 0 if sense switch is not set.

FUNCTION CODE	TASK DESCRIPTION
F$BIO	Buffered I/O request. S1 contains the DSP address.

Perform record oriented I/O request on a COS blocked dataset. A record or partial record is transferred to or from a user-specified data area. Control returns immediately to the user, allowing the user to do processing in parallel with the I/O. The user must check status in the DSP for completion of the request and for errors.

The DSP must contain the following fields set by the user when the call is made (see Appendix A for a description of the DSP):

DPBIO Buffered I/O busy flag must be zero indicating that any previous request has completed. This flag is set by the system when the call is made and cleared when the request is completed.

If a user wants to relinquish the CPU and wait for completion of the buffered I/O request, the user should continue to call recall (F$RCL) until the buffered I/O busy flag is cleared.

DPBER Buffered I/O error flag must be zero, indicating that any error on the previous request has been recognized by the user.

If an error has occurred when a request is completed, DPBER is set to 1. The user may then check DPERR to determine the nature of the error.

DPBF Function code:

- 000 Read partial record, logically equivalent to $RWDP
- 010 Read record, logically equivalent to $RWDR
- 040 Write partial record, logically equivalent to $WWDP
- 050 Write record, logically equivalent to $WWDR

FUNCTION CODE		TASK DESCRIPTION
		052 Write end of file, logically equivalent to $WEOF
		056 Write end of data, logically equivalent to $WEOD
		156 Rewind, logically equivalent to $REWD
	DPBWC	Word count is the number of words to transfer to or from the user's record area. On a read request, the system returns the actual number of words read. If a null record is read, a zero word count is returned in DPBWC. The user may then use DPEOR, DPEOF, and DPEOD to determine if end of record, end of file, or end of data has been reached.
	DPBWA	Word address of user's record area.
F$DLY		Delay job. The job is removed from processing for the number of milliseconds contained in the rightmost 24 bits of the location specified by S1.
F$DJA †		Dump job area. A local dataset $DUMP is created if it does not already exist. After rewinding $DUMP, the job's JTA and user field are written to $DUMP. The number of words written is L@JTA + the job's field length, LA-BA. The dataset created by this task is an unformatted dataset without record control words or block control words. See Appendix A for a description of the Job Table Area (JTA). The JTA contains the user exchange package and B, T, VM, and V registers at the time this call is made.

† Deferred implementation

LOGICAL I/O ROUTINES

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 DSP to the area beginning at FWA until either the word count in A3 is satisfied or an end of record is encountered.

SUBROUTINE NAME:	$RDWP	
ENTRY CONDITIONS:	(A1)	Address of DSP or negative DSP offset relative to DSP base (JCDSP), i.e., contents of second word of ODN table
	(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)	Actual LWA+1 (equals FWA if null record)
	(S0)	Termination mode
		< 0 Read terminated by end of record
		= 0 Null record, end of file, or end of data

>0 Read terminated by count. If count is exhausted simultaneously with reaching end of record, the EOR takes precedence.

(S6) Contains RCW if (S0)≤ 0

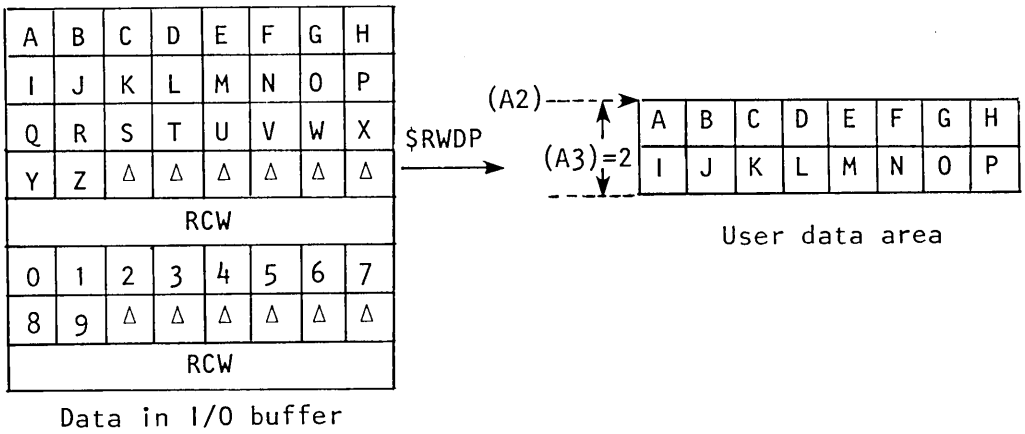
REGISTERS MODIFIED: A0, A1, A4, A5, A6

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:



$RWDR - Read words, record mode

This routine resembles $RWDP. However, following the read, the dataset is positioned after the *<eor>* that terminates the current record.

SUBROUTINE NAME: $RWDR

ENTRY CONDITIONS: Same as $RWDP

RETURN CONDITIONS: Same as $RWDP

REGISTERS MODIFIED: Same as $RWDP

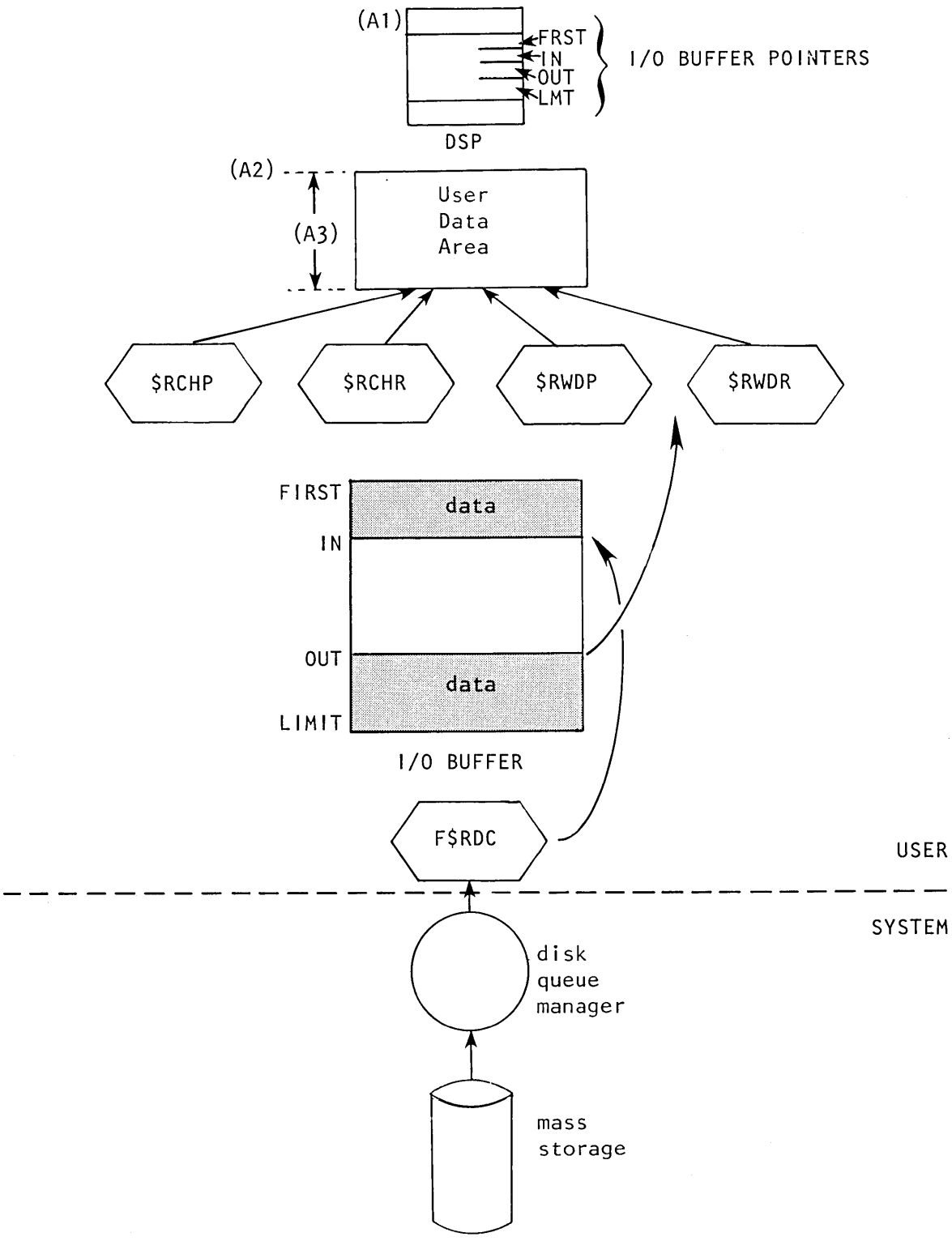


Figure D-1. Logical read

$RCHP - Read characters, partial mode

The $RCHP routine unpacks characters from the I/O buffer defined by the DSP and inserts them into the user data area beginning at the *fwa* specified by (A2) until either the count is satisfied or an <*eor*> is encountered. If an <*eor*> is encountered first, the remainder of the field specified by the character count is filled with blanks.

SUBROUTINE NAME: $RCHP

ENTRY CONDITIONS:

- (A1) Address of DSP or negative DSP offset relative to DSP base (JCDSP), i.e., contents of second word of ODN table
- (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 or end of data
  - >0 Read terminated by count. If count is exhausted simultaneously with reaching end of record, the <*eor*> takes precedence
- (S6) Contains RCW if (S0) $\leq$ 0

REGISTERS MODIFIED:

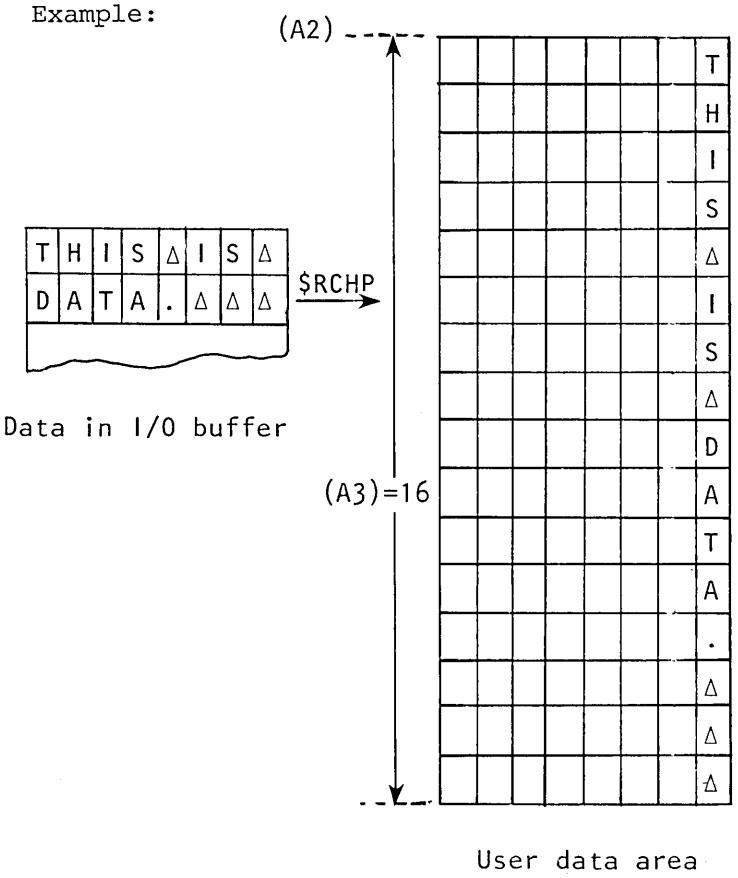
A0, A1, A4, A5, A6

B.ZA, B.ZB (within $B70_8 \ldots B77_8$)

S0, S1, S2, S3, S4, S5, S6

T.ZA (within $T70_8 \ldots T77_8$)

Example:



$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.

SUBROUTINE NAME:	$RCHR
ENTRY CONDITIONS:	Same as for $RCHP
RETURN CONDITIONS:	Same as for $RCHP
REGISTERS MODIFIED:	Same as for $RCHP

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 *fwa* and is written in the I/O buffer defined by DSP.

SUBROUTINE NAME: $WWDP

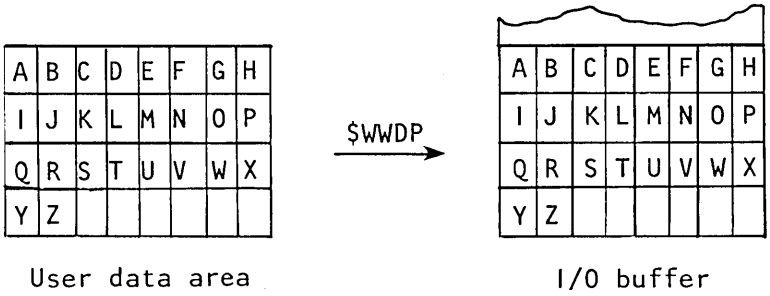
ENTRY CONDITIONS:
- (A1) Address of DSP or negative DSP offset relative to DSP base (JCDSP), i.e., contents of second word of ODN table
- (A2) *fwa* of user data area
- (A3) Word count

  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 $B70_8 \ldots B77_8$)
S0, S1, S2, S3, S4, S5, S6
T.ZA (within $T70_8 \ldots T77_8$)
V0, V1

Example:



User data area I/O buffer

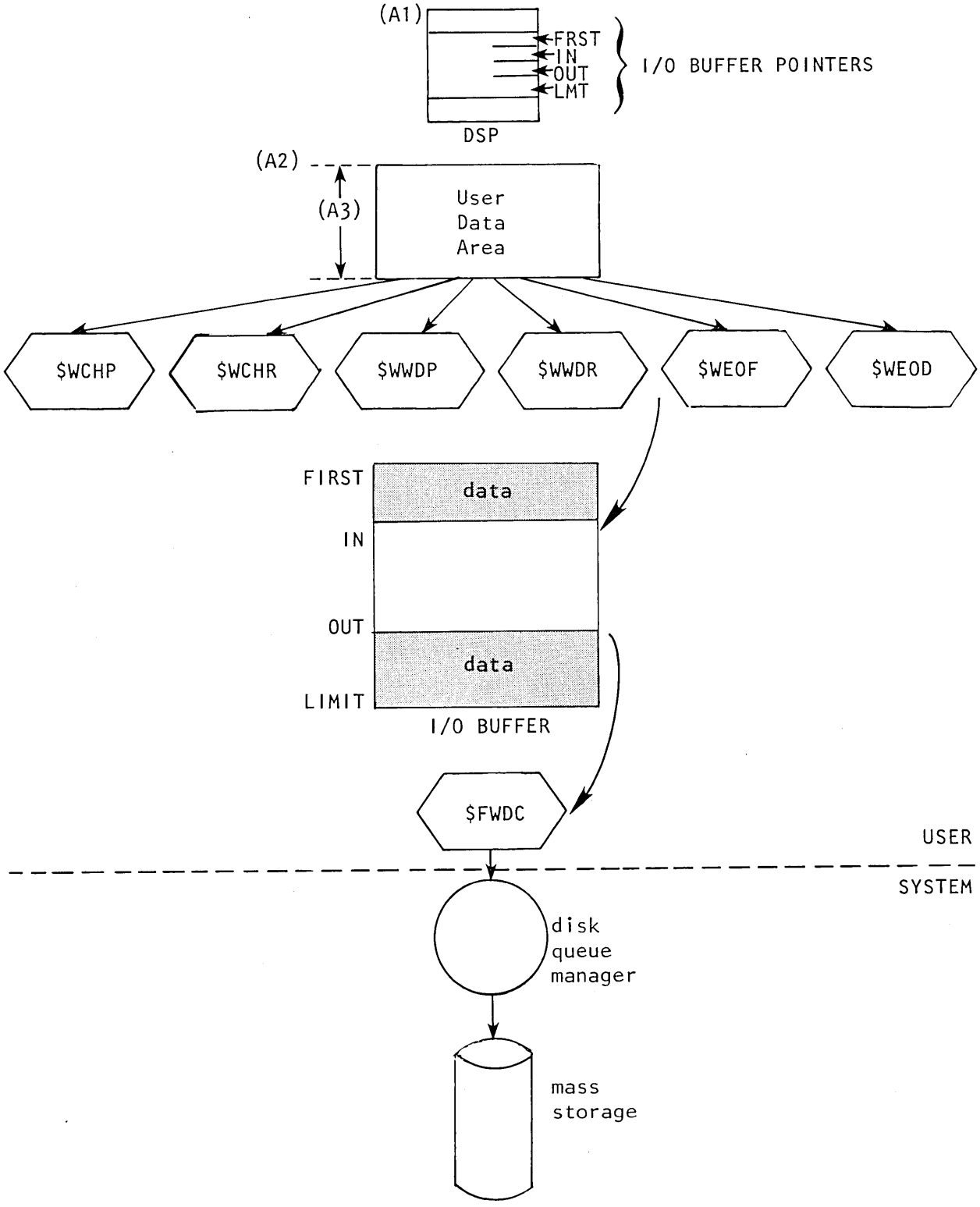


Figure D-2. Logical write

$WWDR - Write words, record mode

The $WWDR routine resembles $WWDP. However, an *<eor>* RCW terminating the record is inserted in the I/O buffer in the next word following the data. To simply write an *<eor>*, 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 use 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 - 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 *fwa* to the I/O buffer for the dataset defined by 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), i.e., contents of second word of ODN table (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

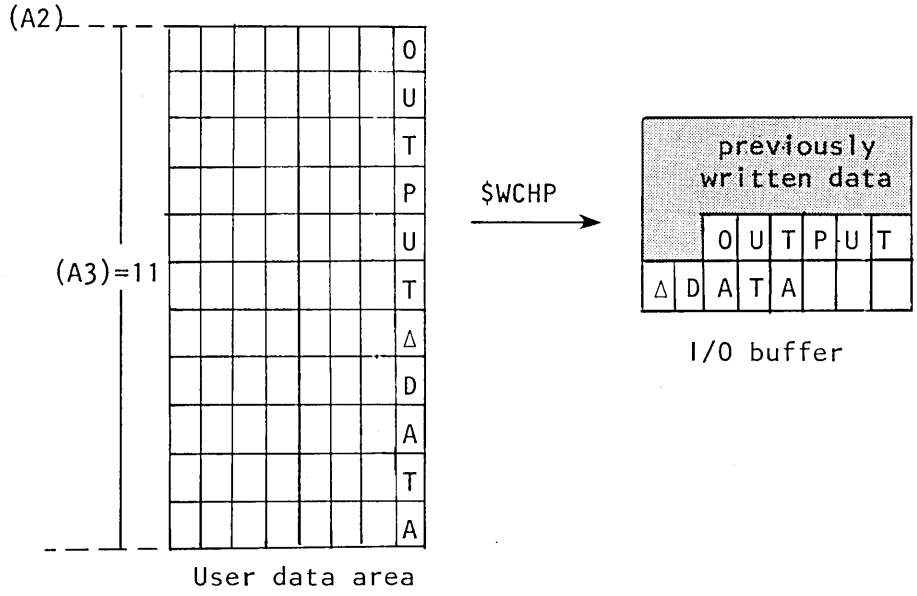
(A4) *lwa*+1

REGISTERS MODIFIED: A0, A1, A4, A5, A6

B.ZA, B.AB (within $B70_8 \ldots B77_8$)

S0, S1, S2, S3, S4, S5, S6

Example:



$WCHR - Write characters record mode

The $WCHR routine resembles $WCHP. However, an <*eor*> 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 simply write an <*eor*>, 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 <*eof*> RCW preceded by an <*eor*> RCW if necessary as the next words in the I/O buffer.

SUBROUTINE NAME:	$WEOF
ENTRY CONDITIONS:	(A1) Address of DSP or negative DSP offset relative to DSP base (JCDSP), i.e., contents of second word of ODN table
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 <*eod*> RCW preceded by an <*eor*> and an <*eof*> if necessary as the next words in the I/O buffer. 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 <*eod*>.

SUBROUTINE NAME:	$WEOD
ENTRY CONDITIONS:	(A1) Address of DSP or negative DSP offset relative to DSP base (JCDSP), i.e., contents of second word of ODN table
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 end of data is written 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 (BOD).

SUBROUTINE NAME:	$REWD
ENTRY CONDITIONS:	(A1) Address of DSP or negative DSP offset relative to DSP base (JCDSP), i.e., contents of second word of ODN table
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 *<eor>* RCW. The function is a no-op if the dataset is at BOD. If the dataset is at the first record of a file, backspace positions the dataset ahead of the *<eof>* RCW.

SUBROUTINE NAME:	$BKSP
ENTRY CONDITIONS:	(A1) Address of DSP or negative DSP offset relative to DSP base (JCDSP), i.e., contents of second word of ODN table
RETURN CONDITIONS:	(A1) Address of DSP
	(S6) Contains RCW after which dataset is positioned (equals 0 if at BOD)
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 <*eof*> RCW. The function is a no-op if the dataset is at BOD.

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, file, or dataset boundary.

This routine does not alter the dataset position.

SUBROUTINE NAME: $GPOS

ENTRY CONDITIONS:

(A1) Address of DSP or negative DSP offset relative to DSP base (JCDSP), i.e., contents of second word of ODN table

RETURN CONDITIONS:

(A1) DSP address

(S1) Dataset position

Flags - the upper four bits of S1 indicate record, file, and dataset boundaries:

<*eor*>	bit 0	End of record flag. 1 indicates the dataset is positioned at a record boundary, i.e., following an RCW. 0 indicates the dataset is either at beginning of data or in the middle of a record.
	bit 1	Unused
<*eof*>	bit 2	End of file flag. 1 indicates the dataset is at a file boundary, i.e., following the end of file RCW.
	bits 3-30	Unused

WA bits 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 (BOD).

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, i.e., 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 DSP or negative DSP offset relative to DSP base (JCDSP), i.e., contents of second word of ODN table.

(S1) Dataset position

bits 0-30 Unused

WA bits 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 *<eod>*, i.e., before the *<eod>* RCW.

(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 (BOD)

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 that 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 suffix; the termination routine name by an F suffix.

Transfer routines are of two types: call by address and call by value. Those called by address have names suffixed with an A. Those called by value have names suffixed with a V. 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

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

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. 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 Cray Research publication 2240009, CRAY-1 (CFT) FORTRAN Reference Manual.

If an end-of-file <*eof*> record is read, zeros or blanks are supplied in place of valid values or characters. An optional <*eof*> exit address may be supplied to the read-initiation routine to suppress this action. Acknowledgement of an <*eof*> record's having been read must occur before initiating another read operation at the same unit. This is done by:

- Providing an <*eof*> 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. This function provides arguments for subsequent $DFA and $DFV calls.
Type of call	By address
Entry	(EP-1) = address of record length
	(EP-2) = address of FORMAT specification
	(EP-3) = address of character string
Exit	No arguments returned

Routine	$DFA (multiple entry points)
Function	Decode formatted, call by address. This function decodes items in a character string, placing results into an array.
Type of call	By address
Entry	(EP-1) = address of array
	(EP-2) = address of item count
	(EP-3) = address of item increment
Exit	Items are at user item addresses

Routine	$DFV (multiple entry points)
Function	Decode formatted, call by value. This function 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. This function terminates a decoding sequence.
Type of call	No arguments required

Routine	$EFI
Function	Encode formatted initialize. This function provides arguments for subsequent $EFA and $EFV calls.
Type of call	By address
Entry	(EP-1) = address of record length
	(EP-2) = address of FORMAT specification
	(EP-3) = address of item increment
Exit	Content of character string buffer

Routine	$EFV (multiple entry points)
Function	Encode formatted, call by value. This function 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. This function terminates an encoding sequence.
Type of call	No arguments required

Routine	$RFI
Function	Read formatted initialize. This function 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 *<eof>* exit address (optional)
Exit	No arguments returned

Routine	$RFA (multiple entry points)
Function	Read formatted, call by address. This function 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	(EP-1) = address of array
	(EP-2) = address of item count
	(EP-3) = address of array address increment
Exit	Decoded items are at user item addresses

Routine	$RFV (multiple entry points)
Function	Read formatted, call by value. This function 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

Routine	$RFF
Function	Read formatted final. This function terminates a read formatted sequence.
Type of call	No arguments required

Routine	$RUI
Function	Read unformatted initialize. This function provides arguments for subsequent $RUA and $RUV calls.
Type of call	By address
Entry	(EP-1) = address of unit name or number
Exit	No arguments returned

Routine	$RUA (multiple entry points)
Function	Read unformatted, call by address. This function 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	(EP-1) = address of array
	(EP-2) = address of word count
	(EP-3) = address of array address increment
Exit	Requested words are in the array

Routine	$RUV (multiple entry points)
Function	Read unformatted, call by value. This function 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)

Routine	$RUF
Function	Read unformatted final. This function terminates a read unformatted sequence.
Type of call	No arguments required

Routine	$WFI
Function	Write formatted initialize. This function 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 <*eof*> exit address (optional)
Exit	No arguments returned

Routine	$WFA (multiple entry points)
Function	Write formatted, call by address. This function 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	(EP-1) = address of array (EP-2) = address of item count (EP-3) = address of array address increment
Exit	Encoded items are in the I/O buffer

Routine	$WFV (multiple entry point)
Function	Write formatted, call by value. This function 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

Routine	$WFF
Function	Write formatted final. This function terminates a write formatted sequence.
Type of call	No arguments required.

Routine	$WUI
Function	Write unformatted initialize. This function provides arguments for subsequent $WUA and $WUV calls.
Type of call	By address
Entry	(EP-1) = address of unit name or number
Exit	No arguments returned

Routine	$WUA (multiple entry points)
Function	Write unformatted, call by address. This function 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	(EP-1) = address of array (EP-2) = address of word count (EP-3) = address of array increment
Exit	No arguments returned

Routine	$WUV (multiple entry points)
Function	Write unformatted, call by value. This function 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	No arguments returned

Routine	$WUF
Function	Write unformatted final. This function terminates a write unformatted sequence.
Type of call	No arguments returned

BUFFERED I/O ROUTINES

Buffered I/O routines perform operations on logical records.

Routine	$RB
Function	Read buffered. Reads up to (EP-4)-(EP-3)+1 words 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 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 <*eod*> 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 described below perform dataset positioning and control operations:

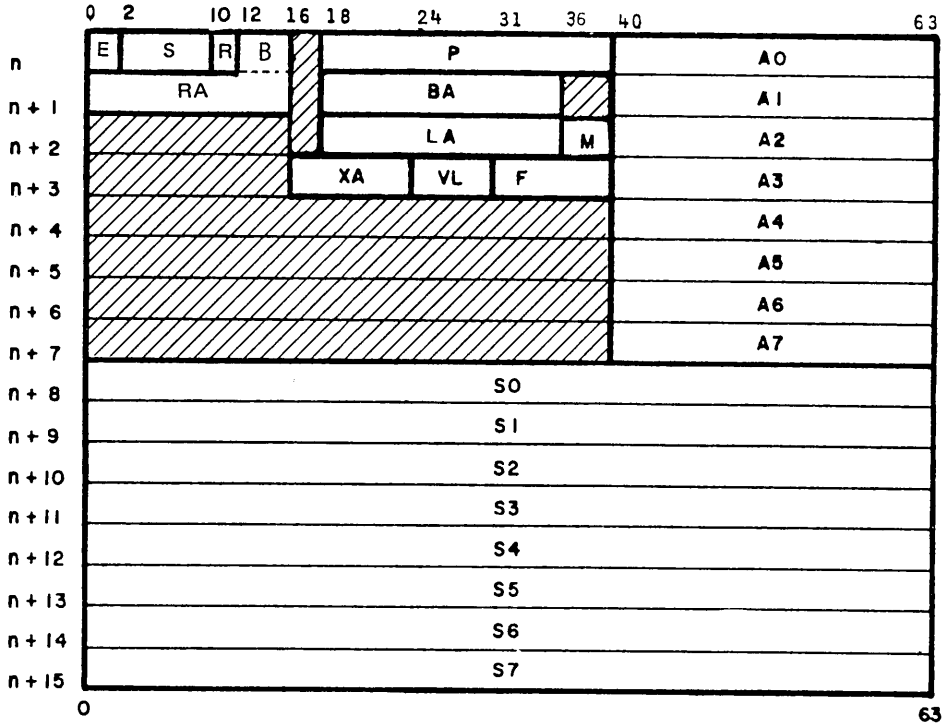
Routine	$EOFW
Function	Write end-of-file. This function writes an end-of-file <*eof*> record on the specified dataset.
Type of call	By address
Entry	(EP-1) = address of unit name or number
Exit	No arguments returned

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 <*bod*> 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



Registers

S	Syndrome bits
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)

10	Uncorrectable memory
01	Correctable memory

R - Read mode (bits 10,11)

00	Scalar
01	I/O
10	Vector
11	Fetch

M - Modes†

36	Interrupt on correctable memory error
37	Interrupt on floating point
38	Interrupt on uncorrectable memory error
39	Monitor mode

F - Flags†

31	PCI interrupt††
32	MCU interrupt
33	Floating point error
34	Operand range
35	Program range
36	Memory error
37	I/O interrupt
38	Error exit
39	Normal exit

†Bit position from left of word

†† Supports Programmable Clock option

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 $, %, and @, as well as the 26 upper-case letters A through Z.

Alphanumeric - A character set including all alphabetic characters and the digits 0 through 9.

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 group of contiguous characters recorded on and read from magnetic tape as a unit. Blocks are separated by record gaps. A 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 1000_8 words (one sector on the DD-19 disk).

Block control word - A word occurring at the beginning of each block in the CRAY-1 blocked format which identifies the sequential position of the block in the dataset and points forward to the next block control word.

Buffer - A storage device used to compensate for a 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.

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 characters 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.

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: local and permanent. A local dataset is a dataset 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.

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.

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.

Dispositon 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 in external format, 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.

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.

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 17_8 in the mode field.

Filemark - Refer to tapemark.

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.

I

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.

J

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 control statement sector - After the job has entered the system and has become a candidate for processing, the job control statements are made into a separate file of the job's input dataset. The user cannot access this file.

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 name $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 other dataset (excluding write operations). The control statements are a separate inaccessible portion of this dataset.

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 - This is a unit of work within a job, such as source language compilation or object program execution.

L

Labeled common - A common block into which data can be stored at load time.

Library - A dataset composed of sequentially organized files. Files other than the last may contain a binary copy of a program or text. The last file contains a directory giving the relative starting position of each of the program or text files as well as information required by the loader for satisfying externals.

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.

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 of a loader map, presetting of unused memory to a user-specified value, and generation of overlays.

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.

M

Macro - 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.

Main frame - 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 - (on line) - The storage of a large amount of data which 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, i.e., permitting more than one program to time-share machine components.

O

Operating system - (1) The executive, monitor, utility, and any other routines necessary for the performance of a computer system. (2) A resident executive program which automates certain aspects of machine operation, particularly as they relate to initiating and controlling the processing of jobs.

$OUT - A dataset that contains the list output from compilers and assemblers unless the user designates some other dataset. At job end, the job log-file is added to the $OUT dataset and the dataset is sent to a front-end computer.

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

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.

Permanent dataset - A dataset known to the operating system as being permanent; the dataset survives deadstart.

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 that usually contains 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 referrred to as IDENT name or deck name, it is the name contained in the loader PDT table at the beginning of each load module.

Program library - see Library.

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 record. (4) For a binary load dataset, each module is a record.

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 assember. 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.

T

Table - A collection of data, each item being uniquely identified either by some label or by its relative position.

Tapemark - A special hardware bit configuration recorded on magnetic tape. It indicates the boundary between datasets and labels. It is sometinmes called a filemark.

Time slice - The maximum amount of time during which the CPU can be assigned to a job without re-evaluation as to which jobs should have the CPU next.

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.

V

Volume - A physical unit of storage media. The term volume is synonymous with reel of magnetic tape.

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

CRAY RESEARCH, INC.

TECHNICAL COMMUNICATIONS
7850 Metro Parkway, Suite 213, Minneapolis, MN 55420 • (612) 854-7472

PUBLICATION CHANGE NOTICE
July 15, 1978

TITLE: CRAY-OS Version 1.0 Reference Manual

PUBLICATION NO. 2240011 REV. E

Revision E obsoletes all previous printings of this publication. This printing brings the manual into full agreement with the July 1978 release of the CRAY-1 Operating System (COS) Version 1.02.

Comment Sheet

Publication Number: 2240011 E

Title: CRAY-OS Version 1.0 Reference Manual

Please feel free to share with us your comments, criticisms, or compliments regarding this publication. We value your feedback. Thank you.

Comments:

Mail to: Publications
CRAY RESEARCH, INC.
7850 Metro Parkway
Suite 213
Minneapolis, MN 55420

CRAY RESEARCH, INC.

MACRO INSTRUCTIONS

CRAY RESEARCH, INC.

HEADQUARTERS • 7850 Metro Parkway, Suite 213, Minneapolis, MN 55420 • (612) 854-7472
DEVELOPMENT LABORATORY • P.O. Box 169, Chippewa Falls, WI 54729 • (715) 723-0266