TABLE OF CONTENTS

SOFTWARE FOR CUSTOMER ENGINEERS 1

Intended Audience: Cray Customer Engineers

Duration: 5 Days

Max. Class Size: 10 Students

Prerequisites:
Is a Cray Employee
Knows Cray Architecture
Knows Cray and IOP Instruction Sets
Has worked with Cray Offline Diagnostics (DSS)
Has six months of Site Experience (COS site preferred)

Course Description: A user-level course which gives students the opportunity to practice the skills necessary to write programs on a Cray system. The class is centered around programming exercises the student will write and run on a Cray. COS, Job Control Language, Cray Assembly Language, UPDATE, and JCL Procs are discussed.

Course Content:

1. Introduction to Cray Software
2. Job Control Language Statements (JCL)
3. Cray Text Editor (TEDI)
4. Cray Assembly Language (CAL)
5. Program Libraries and UPDATE
6. Programming Exercises

Course Objectives:

1. To write simple programs using Job Control Language
2. To manipulate and process COS local datasets
3. To write a CAL program using TEDI and an interactive station
4. To write and submit a job several different ways
5. To program and read with Cray Assembly Language
6. To modify a Program Library using UPDATE

Motivation:

1. To communicate better with customers, operators, and analysts
2. To learn the basic skills for time-sharing a Cray
3. To improve understanding of system operation
4. To enable more efficient response to memory and disk errors
5. To help isolate problems that fail online only
6. To allow more time for analysts to spend on software problems
7. To improve machine availability by reducing offline line used by C.E.
8. To prepare for future Cray products which will require stronger software skills

SOFTWARE FOR CUSTOMER ENGINEERS 1

COURSE SCHEDULE

MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
Introduction COS Datasets Local Datasets Types Format Tape Programming	JCL cont'd. Dumping and Loading ACQUIRE FETCH DISPOSE SUBMIT Preview Exercises 1 - 3	TEDI Demo Preview Exercise 4 Terminal Time	PROCs Preview Exercises 5 - 7 Terminal Time	BUILD UPDATE Terminal Time Class Wrap-Up
Job Control Language COPYn SKIPn SAVE ACCESS DELETE FETCH AUDIT Terminal Orientation	Terminal Time	CAL Source Format Symbolics Pseudos Macros Terminal Time	Terminal Time	OPEN TERMINAL TIME

COURSE MATERIALS

Software for Customer Engineers 1	Workbook
COS Job Control Language	SR-0011
TEDI Reference	SG-0055
CAL Assembly Language	SR-0000
Macros and Opdefs	SR-0012
Library Reference	SR-0014 (optional)
Message Manual	SR-0039 (optional)
Segment Loader	SR-0066 (optional)
UPDATE Reference	SR-0013
JCL Ready Reference	SQ-0067
CAL Ready Reference	SQ-0023
MVS/SPF Editor Guide	

READING ASSIGNMENTS

Monday Night:

SR-0011	Part 1	pages 1-1 to 1-8	Introduction
		pages 2-1 to 2-13	Datasets
		pages 3-1 to 3-13	Job Steps
		pages 4-1 to 4-7	JCL Syntax
	Part 2	pages 1-1 to 1-16	JCL Statements
SR-0055		pages 1-1 to 2-9	TEDI

Tuesday Night:

SR-0000	Chapter 1		CAL Intro
	Chapter 2	pages 2-1 to 2-20	Format Conventions
	Chapter 3	pages 3-1 to 3-9	Symbolic Set
	Chapter 4	page 4-1, pages 4-43 to 4-48	Pseudos
SR-0011	Part 2	pages 9-1 to 9-13	Loader
SR-0012		pages 1-1, 2-1, 3-1, 4-1	Macros

Wednesday Night:

SR-0011	Part 3	pages 4-1 to 4-14	Procs

Thursday Night:

SR-0013	Chapters 1, 2, and 4		UPDATE
SR-0011	Chapter 6	pages 6-15 to 6-16	BUILD Intro
	Chapter 15		BUILD Statement

EVALUATION METHOD

Evaluation of your progress in gaining expertise in these skills is accomplished by assigning a competency level to each skill.

Level

0 No knowledge and no experience.

1 Has some knowledge and limited experience with this skill, but not sufficient to contribute in a work environment.

2 Can perform some parts of this skill satisfactorily, but requires instruction and supervision to perform the entire skill.

3 Can perform some parts of this skill satisfactorily, but requires periodic supervision and/or assistance.

4 Can perform this skill satisfactorily without assistance and/or supervision.

5 Can perform this skill with proficiency in speed and quality without supervision or assistance.

6 Can perform this skill with initiative and adaptability to special situations without supervision or assistance.

7 Can perform this skill and can lead others in performing it.

Successfully completing this course should give you a competency level of three (3) for most skills. Experience on the job will continue to increase your competency level.

Software for Customer Engineers I

Date: ____________________

Participant's Name:

____

Instructor's Name:

____

Region/Country:

____

LEARNING LOG

SWCE 1									
Skills At the end of the course the learner is able to:									
Program in JCL.									
Manipulate datasets.									
Program using an interactive station.									
Submit a job several different ways.									
Construct and modify program libraries.									
Program and read with CAL.									
Levels	0	1	2	3	* 4	5	6	7	No Basis For Judgement

Sessions attended/held _____/_____

Exercises completed/assigned _____/_____

Labs attended/held _____/_____

This learning log is intended as an aid to the learner in establishing goals and plotting progress. It is not intended as an indicator of job performance and therefore should not be used in determining future job actions.

*Maximum level discernible by the instructor in an instructional environment.

INSTRUCTOR'S FEEDBACK

# Sessions attended/held _____ / _____
# Exercises completed/assigned _____ / _____
# Labs attended/held _____ / _____

MET THE PREREQUISITES OF THE COURSE

not at all — yes — was over qualified

Specifics:

SELF APPRAISAL

too high — is correct — too low

3 levels — 2 levels — 1 level — 1 level — 2 levels — 3 levels

Specifics:

WAS ACTIVE AND ATTENTIVE IN CLASS

not at all — to a normal degree — exceptionally so

Specifics:

MADE GOOD USE OF TERMINAL TIME

not at all — to a normal degree — exceptionally so

Specifics:

KEPT UP WITH THE REST OF THE CLASS

fell behind the class — yes — was ahead of the class

Specifics:

SHOWS A POSITIVE ATTITUDE ABOUT WORKING AT CRAY

not at all — *to a normal degree* — exceptionally so

Specifics:

Comments:

These are subjective appraisals based on the instructors brief and limited observations of the learners behavior during the class.

Software for Customer Engineers I

Date: ______________________

Participant's Name:

____

Instructor's Name:

____

Region/Country:

____

LEARNING LOG

SWCE 1									
Skills At the end of the course the learner is able to:									
Program in JCL.									
Manipulate datasets.									
Program using an interactive station.									
Submit a job several different ways.									
Construct and modify program libraries.									
Program and read with CAL.									
Levels	0	1	2	3	* 4	5	6	7	No Basis For Judgement

Sessions attended/held _____/_____

Exercises completed/assigned _____/_____

Labs attended/held _____/_____

This learning log is intended as an aid to the learner in establishing goals and plotting progress. It is not intended as an indicator of job performance and therefore should not be used in determining future job actions.

*Maximum level discernible by the instructor in an instructional environment.

Introduction to Cray Software

1

MODULE OBJECTIVES

Upon completion of this introduction module, and with the aid of all furnished reference material, the learner should be able to:

1. Diagram a Cray computer system
2. Describe the function of each hardware component
3. Explain what the software components are
4. Describe the function of the software components
5. Describe the difference between machine code, assembly language, and high-level programming
6. Label a Cray memory map, including user areas
7. Identify the job default datasets
8. Analyze COS blocked datasets for BCW, EOR, EOF, and EOD control words

MENDOTA HEIGHTS COMPUTER CENTER CONFIGURATION

2 CRAY COMPUTERS:

1 CRAY X-MP/48

4 central processors (CPUs)
8 Million words of central memory
128 million words of SSD
7 front-end computers maximum

1 CRAY X-MP/22

2 central processors (CPUs)
2 million words of central memory
8 million words of SSD
3 front-end computers maximum

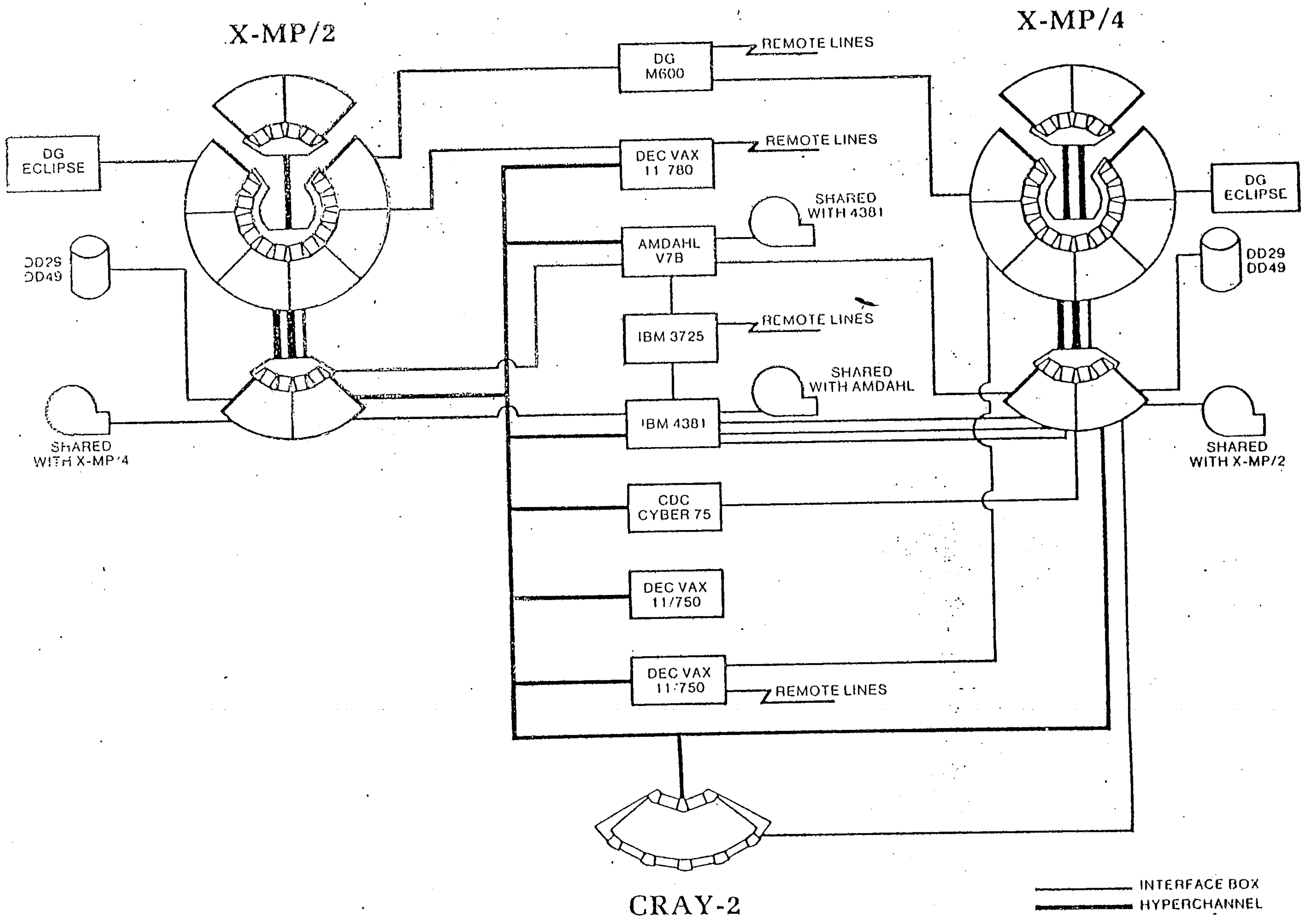
FRONT-END COMPUTERS

Digital Equipment VAX 11/780
Digital Equipment 11/44 (planned)
Amdahl
IBM 4381
CDC CYBER
Data General M600

NSC HYPERChannel to all computers but the Data General M600

The terminals used in the Mendota Heights Training Center communicate with the CRAYs through either the IBM 4381 or the Amdahl. (See the configuration layout on the following page.)

Computer Services Computer Center Configuration



SOFTWARE PRODUCTS

OPERATING SYSTEMS

COS	- Cray Operating System Multiprogramming, multiprocessing, multitasking
CXOS	- Cray Operating System for compatibility between Cray X-MP and Cray-2
IOS	- Input/Output Subsystem Peripheral Devices

PRODUCT SET

Languages

CFT	- FORTRAN Compiler, Vectorizing and Optimizing
CAL	- Cray Assembly Language
APML	- A-Processor Macro Language
PASCAL	- Structured Algorithm Compiler

Libraries

Program Libraries - Source Code

Product Set, System Generation, and Diagnostics

Binary Libraries - Common Routines

$SYSLIB	(System subroutines)
$SCILIB	(Scientific math subroutines)
$FTLIB	(FORTRAN subroutines)
$ARLIB	(Arithmetic subroutines)
$IOLIB	(System I/O subroutines)
$UTLIB	(Code conversion subroutines)

Utilities and Aids

Job Control Language
Permanent Dataset Security and Utilities
Local Dataset Utilities
Staging Datasets
Debugging Aids
Library Utilities
Operational Aids and Utilities

FRONT-END STATION SOFTWARE

Programs that run on front-end code

IBM - MVS, VM
Control Data - NOS, NOS/BE
DEC - VMS
Data General - RDOS, AOS

APPLICATIONS

NASTRAN	- Structural analysis
EISPACK	- Eigenwave matrices
LINPACK	- Simultaneous linear equations
SCILIB	- Linear algebra, FFT, and filtering
CSPICE	- Electronic circuit simulation
BYU.MOVIE	- General-purpose graphics
BETAII	- Geophysical simulation
AMOSLIB	- Atmosphere simulation
MORSE	- Nuclear simulation

CRAY RESEARCH SOFTWARE

STATION

Resides on the Front End

OPERATING SYSTEM

Resides in Central Memory, Local Memory and Buffer Memory

PRODUCT SET

Resides on Cray Disk Drives

APPLICATIONS

THE CRAY SOFTWARE PRODUCT SET

LANGUAGES

Set of characters, symbols, words, etc. used to communicate with a computer.

CAL	-	Cray Assembly Language
CFT	-	Cray version of Formula Translation (FORTRAN) Compiler
PASCAL	-	Structured Algorithm Compiler
C	-	Base language for CXOS

LIBRARIES

Set of general-purpose software to perform common routines. These are subroutines that already exist and are available for use by a programmer.

$SYSLIB	-	System subroutines (e.g. access or delete a permanent dataset)
$SCILIB	-	Math routines used for scientific purposes (e.g. matrix multiply)
$FTLIB	-	FORTRAN subroutines (e.g. square root)
$ARLIB	-	Arithmetic routines (e.g. sine function)
$IOLIB	-	Dataset movement (e.g. copy datasets)
$UTLIB	-	Conversions (e.g. binary to decimal ASCII)

UTILITIES and AIDS (examples)

UPDATE	-	Create source libraries Modify existing libraries, operating systems, or current jobs Line-oriented source maintenance (text editor)
BUILD	-	Create binary libraries Modify/maintain libraries Works with object code
JCL	-	Job Control Language for submitting jobs to the Cray

```
AUDIT             COS 1.14
PDN               ID          ED        PDN               ID          ED

$APTEXT           V114BF1     1         $ARLIB            V114BF1     1
$DBHELP           V114BF1     1         $FTLIB            V114BF1     1
$IOLIB            V114BF1     1         $PSCLIB           V114BF1     1
$SCILIB           V114BF1     1         $SID              V114BF1     1
$SYSLIB           V114BF1     1         $SYSTXT           V114BF1     1
$UTLIB            V114BF1     1         $UTLTXT           V114BF1     1
ACCOUNT           V114BF1     1         ACCTDEF           V114BF1     1
ADSTAPE           V114BF1     1         APML              V114BF1     1
ARLIBPL           V114BF1     1         AUDIT             V114BF1     1
AUDPL             V114BF1     1         AUTODIR           V114BF1     1
BIND              V114BF1     1         BUILD             V114BF1     1
CAL               V114BF1     1         CALPL             V114BF1     1
CFT               V114BF1     1         CFTPL             V114BF1     1
CHARGES           V114BF1     1         COMPARE           V114BF1     1
COPYD             V114BF1     1         COPYF             V114BF1     1
COPYR             V114BF1     1         COPYU             V114BF1     1
COSPL             V114BF1     1         COSTXT            V114BF1     1
CSIM              V114BF1     1         CSIMPL            V114BF1     1
DEBUG             V114BF1     1         DSDUMP            V114BF1     1
DUMP              V114BF1     1         EXTRACT           V114BF1     1
FDUMP             V114BF1     1         FLODUMP           V114BF1     1
FTREF             V114BF1     1         GENPL             V114BF1     1
IOLIBPL           V114BF1     1         IOPPL             V114BF1     1
ITEMIZE           V114BF1     1         JCSDEF            V114BF1     1
LDR               V114BF1     1         LDRPL             V114BF1     1
MODSEQ            V114BF1     1         MODSET            V114BF1     1
PASCAL            V114BF1     1         PASCLPL           V114BF1     1
PDSDUMP           V114BF1     1         PDSLOAD           V114BF1     1
PRVDEF            V114BF1     1         SCILBPL           V114BF1     1
SEGLDR            V114BF1     1         SEGRLS            V114BF1     1
SETOWN            V114BF1     1         SIDPL             V114BF1     1
SKIPD             V114BF1     1         SKIPF             V114BF1     1
SKIPR             V114BF1     1         SKIPU             V114BF1     1
SKOL              V114BF1     1         SKOLPL            V114BF1     1
SKOLREF           V114BF1     1         SKOLTXT           V114BF1     1
SPAWN             V114BF1     1         STATS             V114BF1     1
STEP              V114BF1     1         SYSLBPL           V114BF1     1
SYSREF            V114BF1     1         TEDI              V114BF1     1
TEDIPL            V114BF1     1         TOOLPL            V114BF1     1
UNB               V114BF1     1         UPDATE            V114BF1     1
UPDPL             V114BF1     1         UTILPL            V114BF1     1
UTLIBPL           V114BF1     1         WRITEDS           V114BF1     1

          84 DATASETS,    17164 BLOCKS,            8787968 WORDS
```

WHAT IS A JOB?

1. Work for Cray to do

2. A text dataset

3. Originated with a text editor

4. Submitted interactively or batch

5. Exists in memory at execution time

6. Consists of:

   . Job Table Area (JTA), which contains -
   Job-related information
   User log
   User JCL
   Exchange package, B, T, and V registers
   Local dataset name tables
   Dataset allocation tables

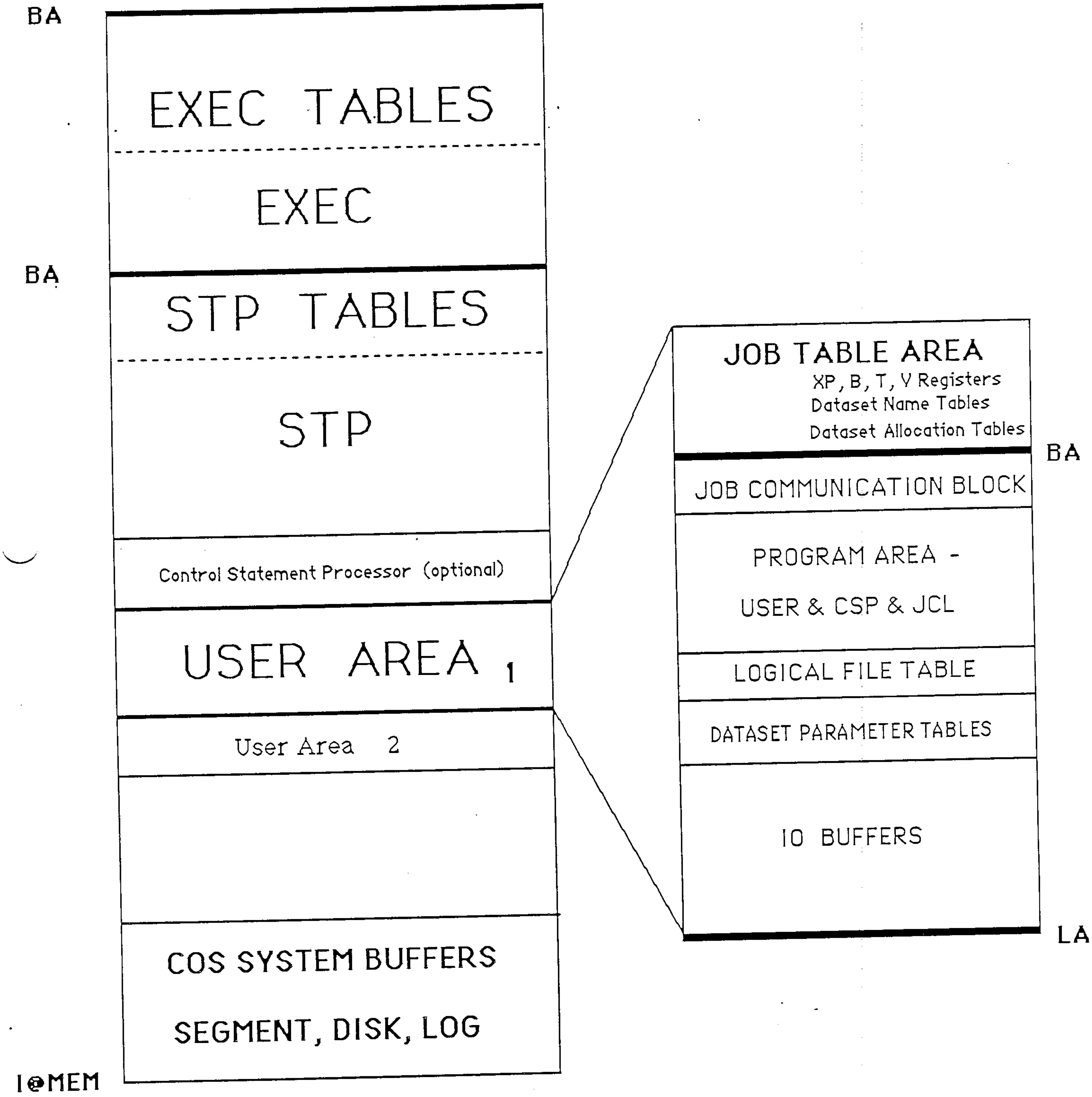
   The Job Table Area cannot be manipulated by the user, but its contents can be dumped.

   User Area, which contains -

   Job Comunication Block
   Program/data
   Control Statement Processor
   Product set
   User-written text/code
   Logical file tables
   Dataset parameter area
   I/O buffers

7. Created and maintained by COS task job scheduler

CENTRAL MEMORY



BA

EXEC TABLES

EXEC

BA

STP TABLES

STP

Control Statement Processor (optional)

USER AREA 1

User Area 2

COS SYSTEM BUFFERS

SEGMENT, DISK, LOG

I@MEM

JOB TABLE AREA

XP, B, T, V Registers

Dataset Name Tables

Dataset Allocation Tables

BA

JOB COMMUNICATION BLOCK

PROGRAM AREA -

USER & CSP & JCL

LOGICAL FILE TABLE

DATASET PARAMETER TABLES

IO BUFFERS

LA

JOB SUBMISSION

Front-end computer provides input to Cray and receives the Cray output.

Controlled from a station:
- IOP station
- Local operator console
- Batch entry station
- Interactive station
- Concentrator for several stations
- Remote batch entry station

First file of transfer must be Job Control Language

Cray Operating System (COS) handles transmission

Job Control Language file
- Specifies needed system resources
- Defines job processing steps
- Maintains database

JOB DEFAULT DATASETS

$CS

$CS is a copy of the job's control statement file from the input dataset and is used only by the system; the user cannot access $CS by name. Cray reads this dataset to get the job control statements.

$IN

This is the job input dataset. The job itself can access the input dataset, with read-only permission, by its local name, $IN, or as FORTRAN unit 5.

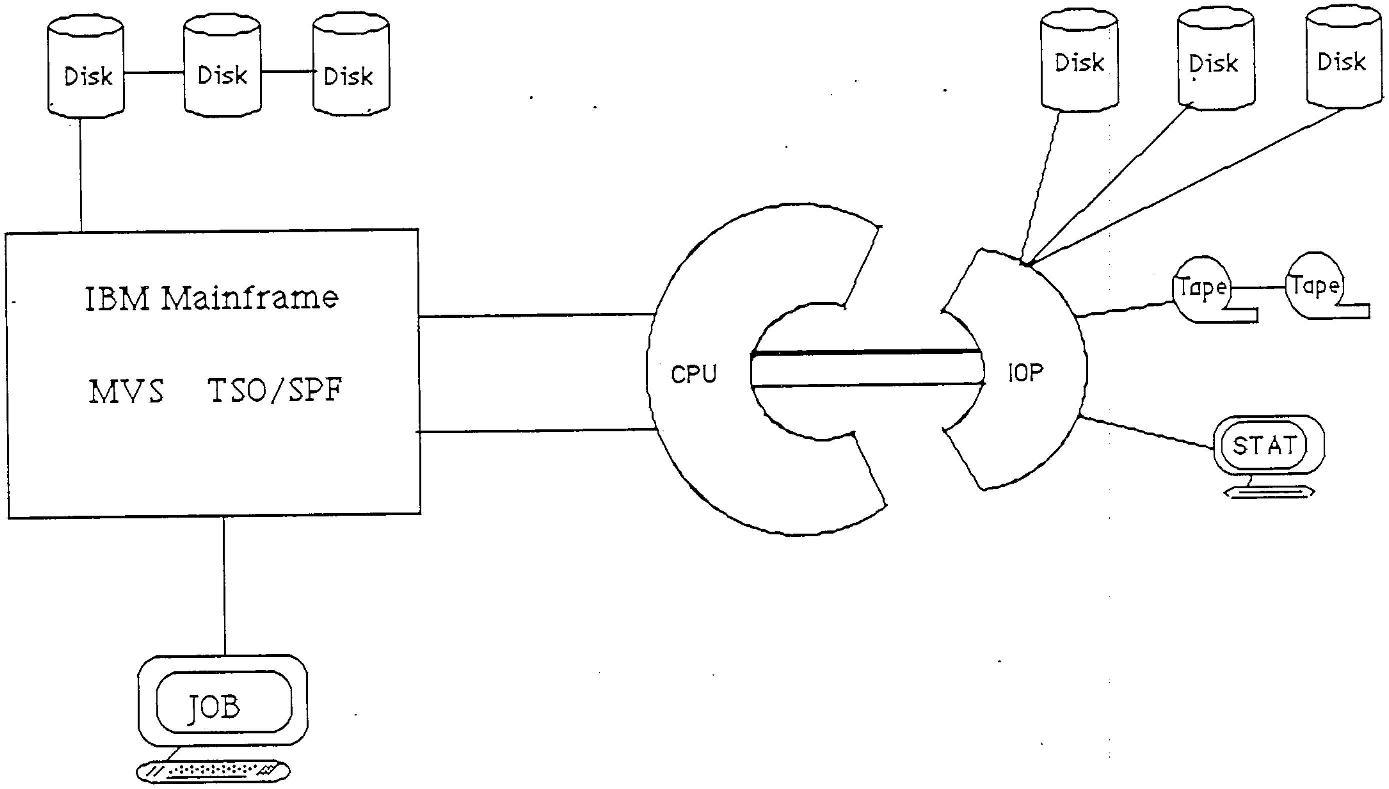
$OUT

This is the job output dataset. The job can access this dataset by name, $OUT, or as FORTRAN unit 6.

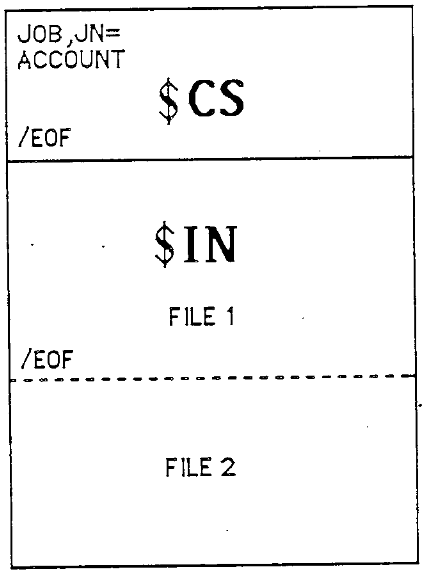
$LOG

The job's logfile contains a history of the job. This dataset is know only to COS and is not accessible to the user. (User messages can be added to the logfile, however, using the MESSAGE system action request macro or other user remark subroutines.)

SUBMITTING A JOB FROM A FRONT-END



FILE SENT TO CRAY



JOB,JN=
ACCOUNT

$CS

/EOF

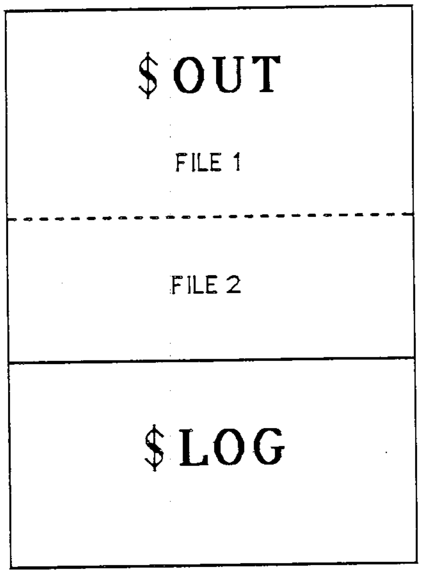
$IN

FILE 1

/EOF

FILE 2

FILE RETURNED TO USER



$OUT

FILE 1

FILE 2

$LOG

LOCAL DATASETS

Local datasets are known only to one job. They cannot be used by another job.

Local datasets contain information in various forms:

Text in ASCII
Source program in ASCII
Text, data, and source program in ASCII format
Binary load module - object program
Executable binary program
Binary data
Source program library
Procedure library
Object program library

Job local datasets have required characteristics. They are:

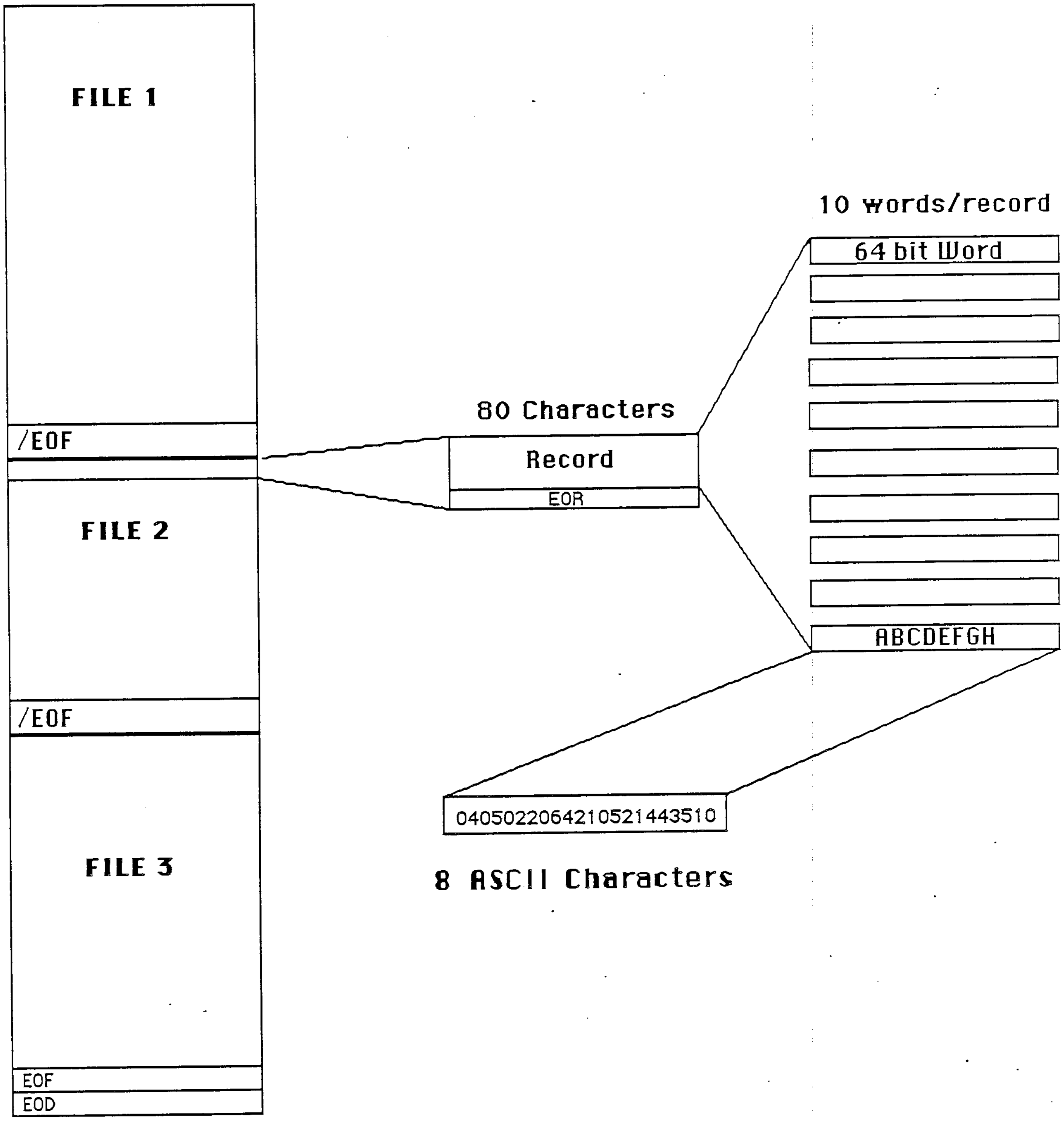
Identified by a dataset name table (DNT) in the Job Table Area (JTA)
Named using 1 - 7 alphanumeric characters; the first character must be
A - Z, $, @, or %
Available only to the job that created it
Deleted from the system at job end unless saved
Allocated in the users's I/O buffers

Four default datasets are local to a job:

$IN	- used for source programs, statement directives, or data
$OUT	- used for JCL outputs, assembler listings, and loader map
$CS	- Job statement file; must begin with the JOB and ACCOUNT statements
$LOG	- File containing records of each job step and system actions

LOCAL DATASETS

This example is $IN



FILE 1

/EOF

FILE 2

/EOF

FILE 3

EOF

EOD

80 Characters

Record

EOR

10 words/record

64 bit Word

ABCDEFGH

0405022064210521443510

8 ASCII Characters

DATASET FORMATS

Blocked Format

Blocked format is used by default for external types of datasets, such as user input and output datasets. Record positioning requires a blocked format. 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.

The data in a blocked dataset can be in the form of ASCII code and/or binary. Blanks are normally compressed in block coded datasets. Each block consists of 512 words.

Refer to SR-0011, Part 1, 2-6 for format.

Interactive Format

Interactive format closely resembles blocked format; however, each buffer begins with a block 0 Block Control Word (BCW).

Each record transmitted in an interactive mode to or from COS must contain a single record consisting of a Block Control Word, data, and an end-of-record Record Control Word.

Two formats for interactive output can be assigned when the dataset is created: character blocked and transparent. Character blocked mode is the default. In this mode, an end-of-record RCW is interpreted as a line feed or carriage return. In transparent mode, the end-of-record RCW is ignored and the user must provide carriage control characters.

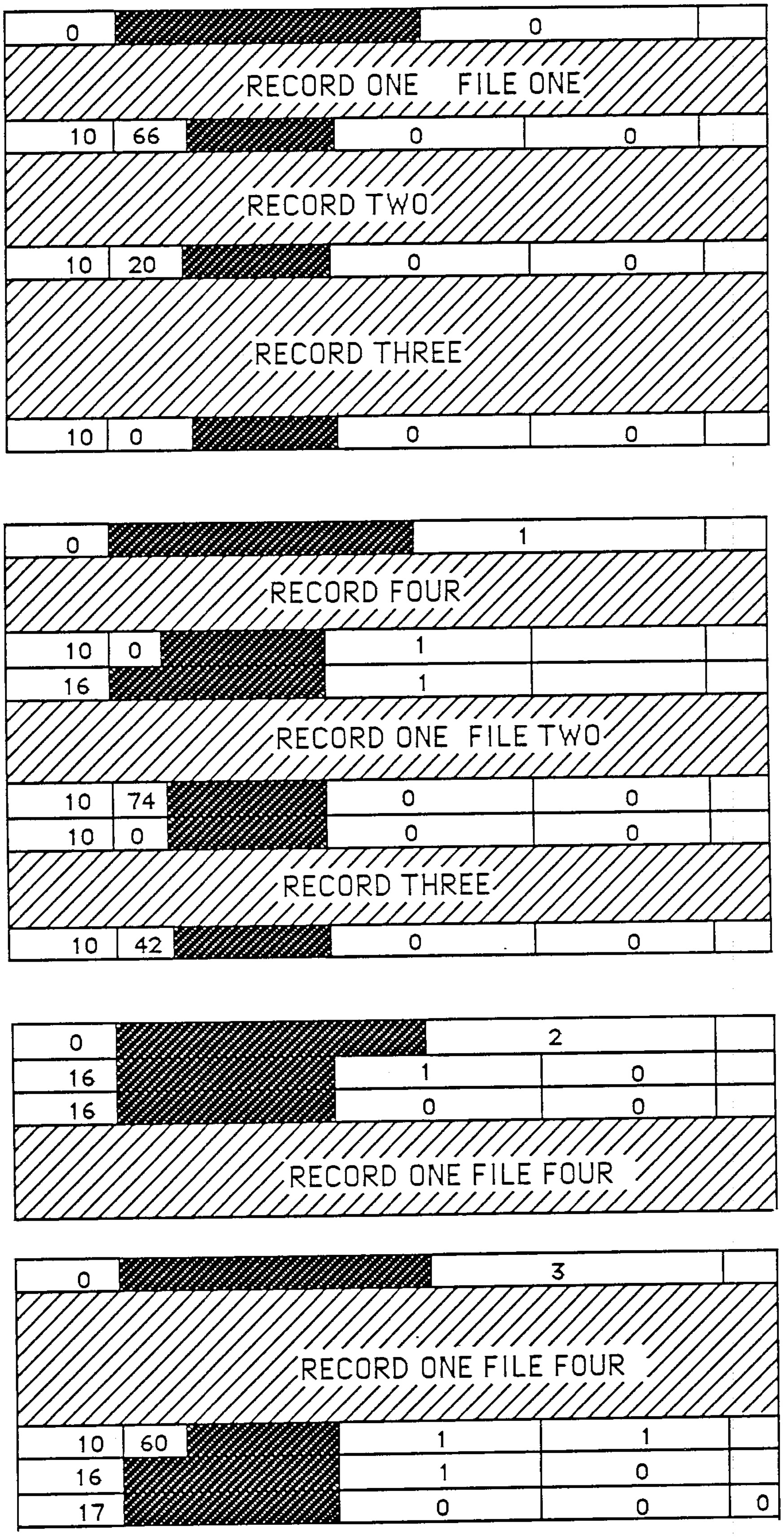
Unblocked Format

Dataset I/O can also be performed using unblocked datasets. The data stream for unblocked datasets does not contain RCWs or BCWs.

The stream does not allocate buffers in the job's I/O buffer area for unblocked datasets; the user must specify an area for data transfer.

When a read or write is performed on an unblocked dataset, the data goes directly to or from the user data area without passing through an I/O buffer. The word count of data to be transferred must be in multiples of 512.

BLOCKED DATASET FORMAT



TAPE DATASETS

Tape datasets can be read or written using two different formats:

Interchange

Transparent

Interchange Format

Interchange format enables reading and writing tapes that are also to be read and written on other vendors' systems.

In interchange format, each tape block of data corresponds to a single logical record in COS blocked format (that is, the data between record control words).

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

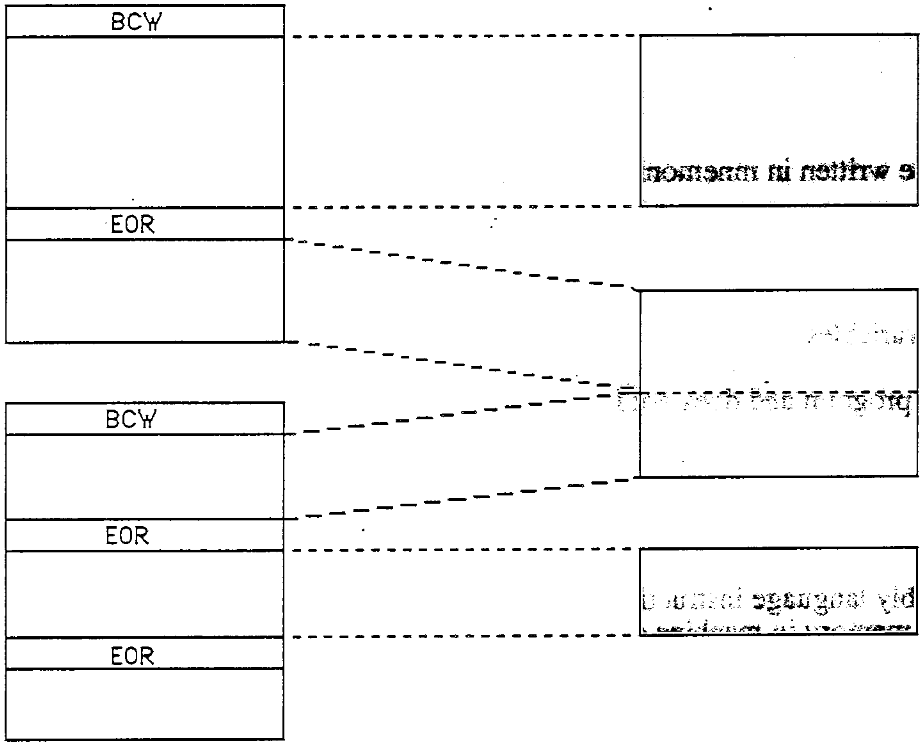
When a dataset is read in interchange mode, physical tape blocks are represented in the user's I/O buffer with block control words (BCWs) and record control words (RCWs) added by COS. The data in each tape block is terminated by an RCW. The unused bit count field in the RCW indicates the amount of data in the last word of the tape block that is not valid data. A BCW is inserted before every 511 words of data, including the RCWs. The format of RCWs and BCWs are described previously in this lesson.

Transparent Format

In transparent format (disk image), each tape block is a fixed multiple of 4096 bytes (512 words), generally based on the dataset density (i.e. 16,384 bytes at 1600 bpi and 32,768 bytes at 6250 bpi). The data in the tape block is transferred unaltered between the tape and the I/O buffer in the user field; no control words are added on reading or discarded on writing.

In transparent mode, the data can be in COS blocked format or COS unblocked format. Transparent format tapes are not generally read or written by other vendors' equipment.

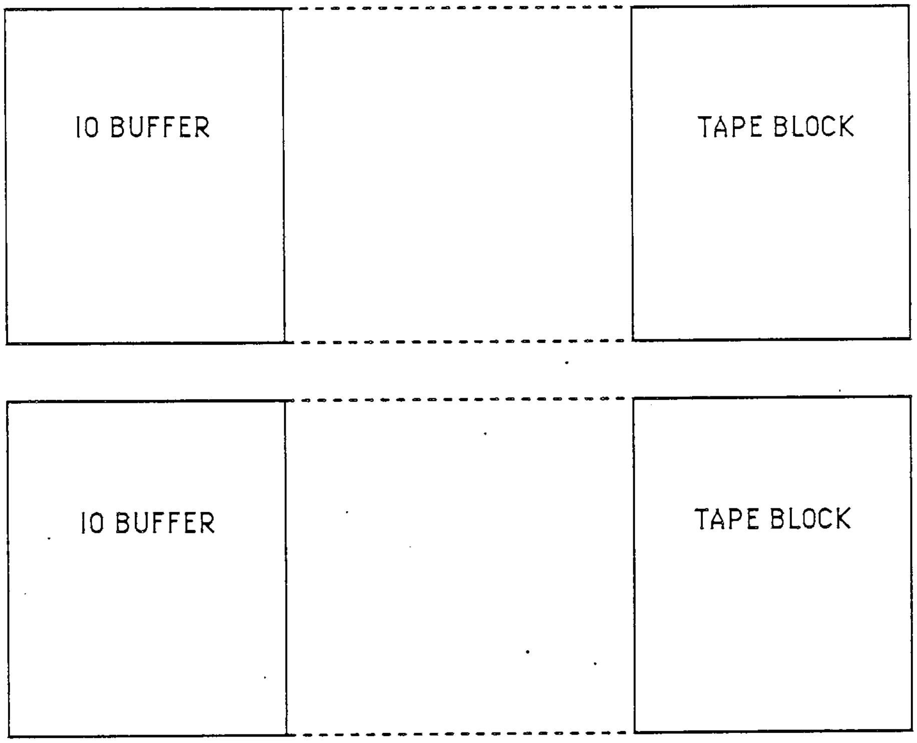
TAPE FORMATS



INTERCHANGE

1 COS Blocked Record

= 1 Tape Block



TRANSPARENT

Fixed Length Tape Blocks

6250 BPI = 32768 bytes

1600 BPI = 16384 bytes

ASSEMBLY LANGUAGE

Characteristics:

Machine dependent

Allows programs to be written in mnemonics or symbols

Performs a 1 to 1 interpretation - for every assembly language instruction a machine code instruction is generated

Can assign names to variables

Speeds writing of the program and does not force the programmer to keep track of all memory locations

Works in conjuction with a program called an assembler

The assembler:
  Interprets assembly language instructions and converts them to machine code
  Resides in main memory in machine code (binary) for use by a source program

Advantages to assembly language over machine language:

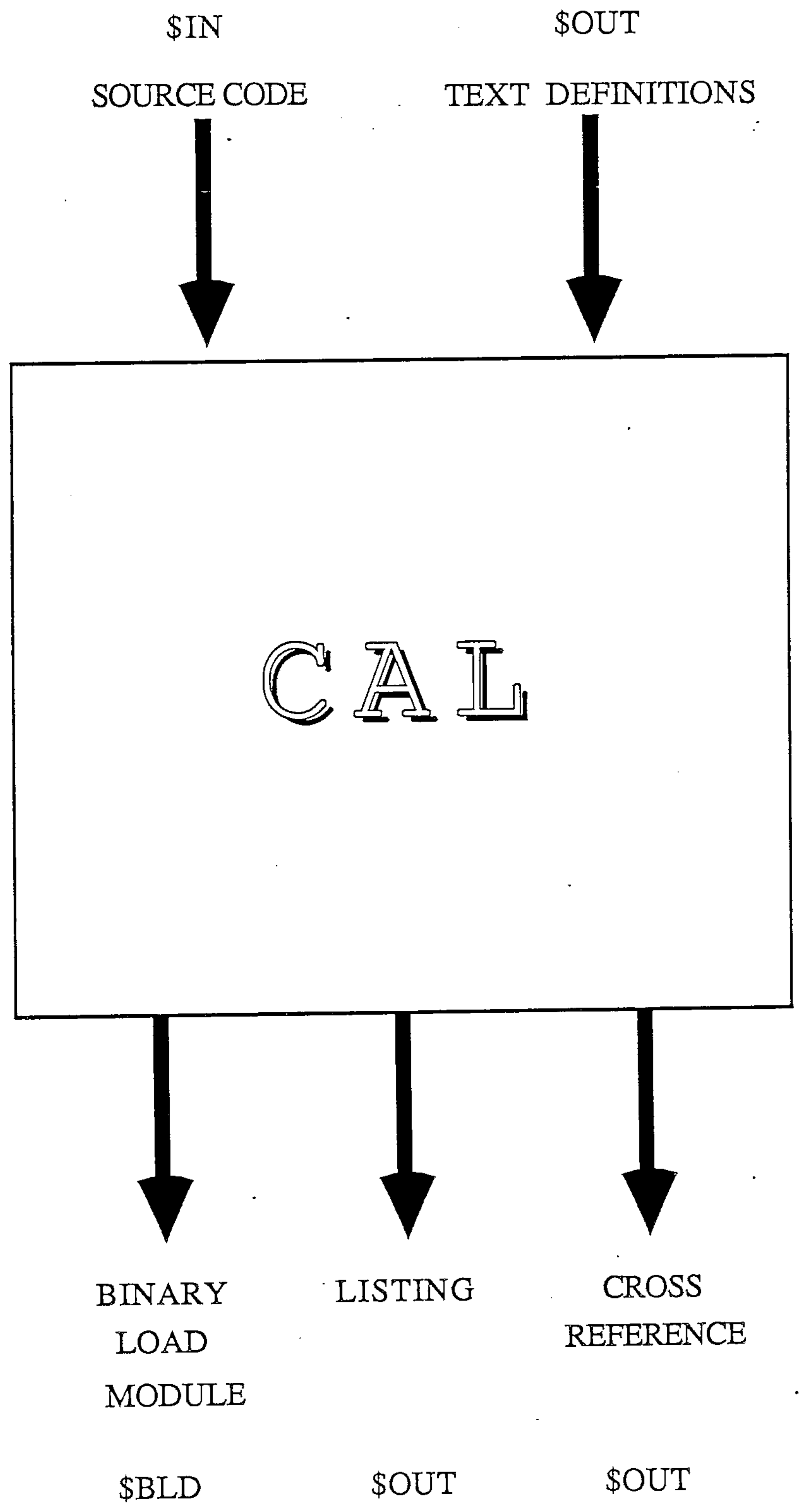
Alphanumeric operation codes are easier to remember than numeric codes

Storage locations for instructions or data can be given names rather than having to remember numeric addresses

Programs can be written in a more straightforward write-it-out manner

Modifications to a program are faster since remanipulation of addresses is not needed

CAL ASSEMBLER



$IN

SOURCE CODE

$OUT

TEXT DEFINITIONS

CAL

BINARY
LOAD
MODULE

$BLD

LISTING

$OUT

CROSS
REFERENCE

$OUT

HIGH - LEVEL LANGUAGES

Characteristics:

Machine independent

Depend on standard readable language

Allow a programmer to express many instructions with a given line of code
Example: Add B to C
Store A

Allow complex algorithms performed without repetitive coding

Work in conjuction with a compiler or interpreter
- A compiler is language dependent and produces binaries dependent upon the machine

Common high-level languages:

BASIC	- Beginner's All-Purpose Symbolic Instruction Code
FORTRAN	- Formula Translation
COBOL	- Common Business Oriented Language
PASCAL	- Structured Algol Programming
ALGOL	- Algorithmic Language

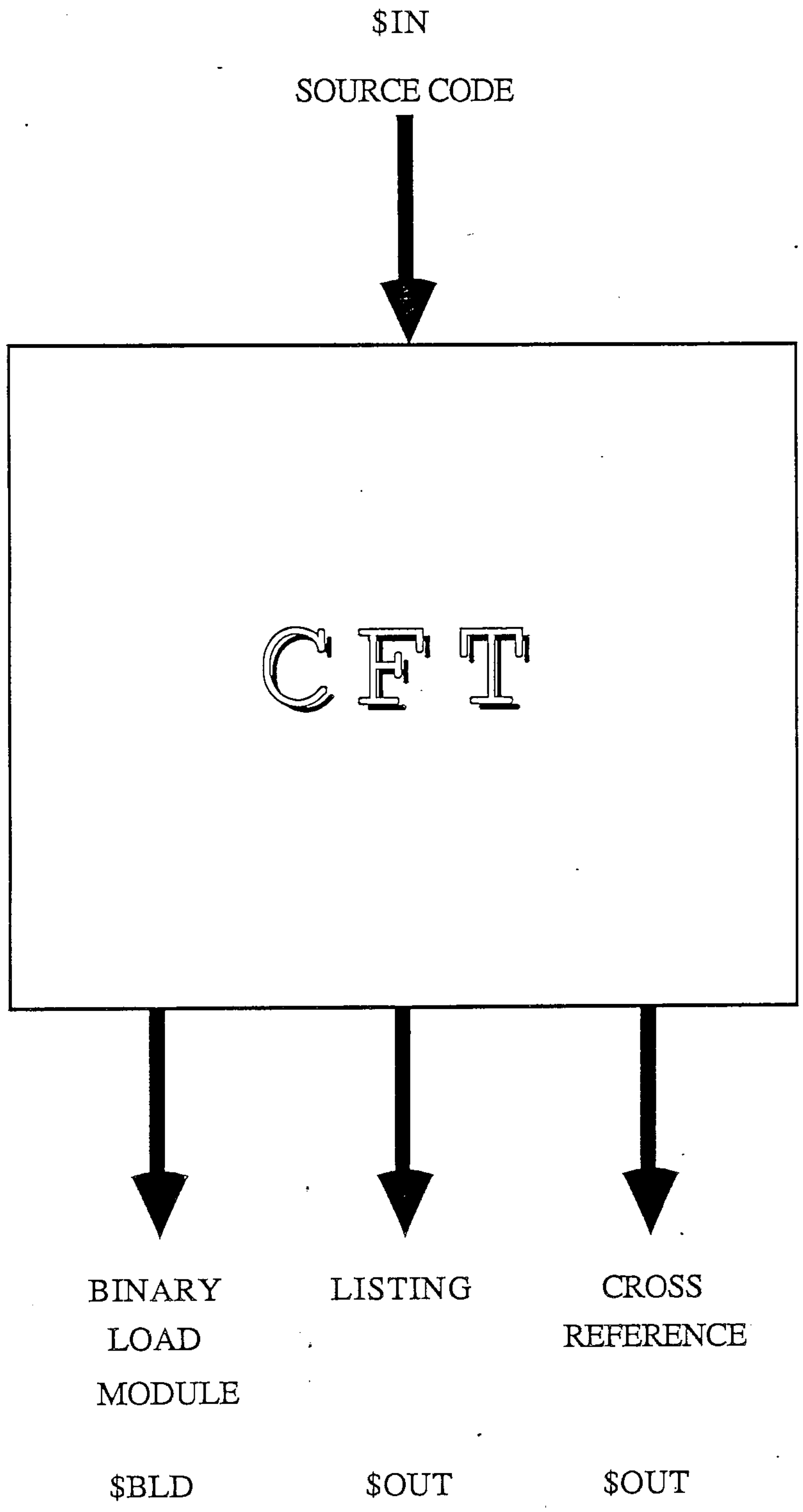
Advantages of High-Level Languages:

Programs will transfer from machine to machine

Allows programmers to write many instructions with one line of code

Make it easier for programmer to use other computers

CFT COMPILER



$IN

SOURCE CODE

CFT

BINARY LOAD MODULE

$BLD

LISTING

$OUT

CROSS REFERENCE

$OUT

LOADER

Functions:

Creates executable binaries

Plugs in binary modules from libraries

Saves time on commonly used routines

Is the second pass of the assembler

Links external symbols from module to module

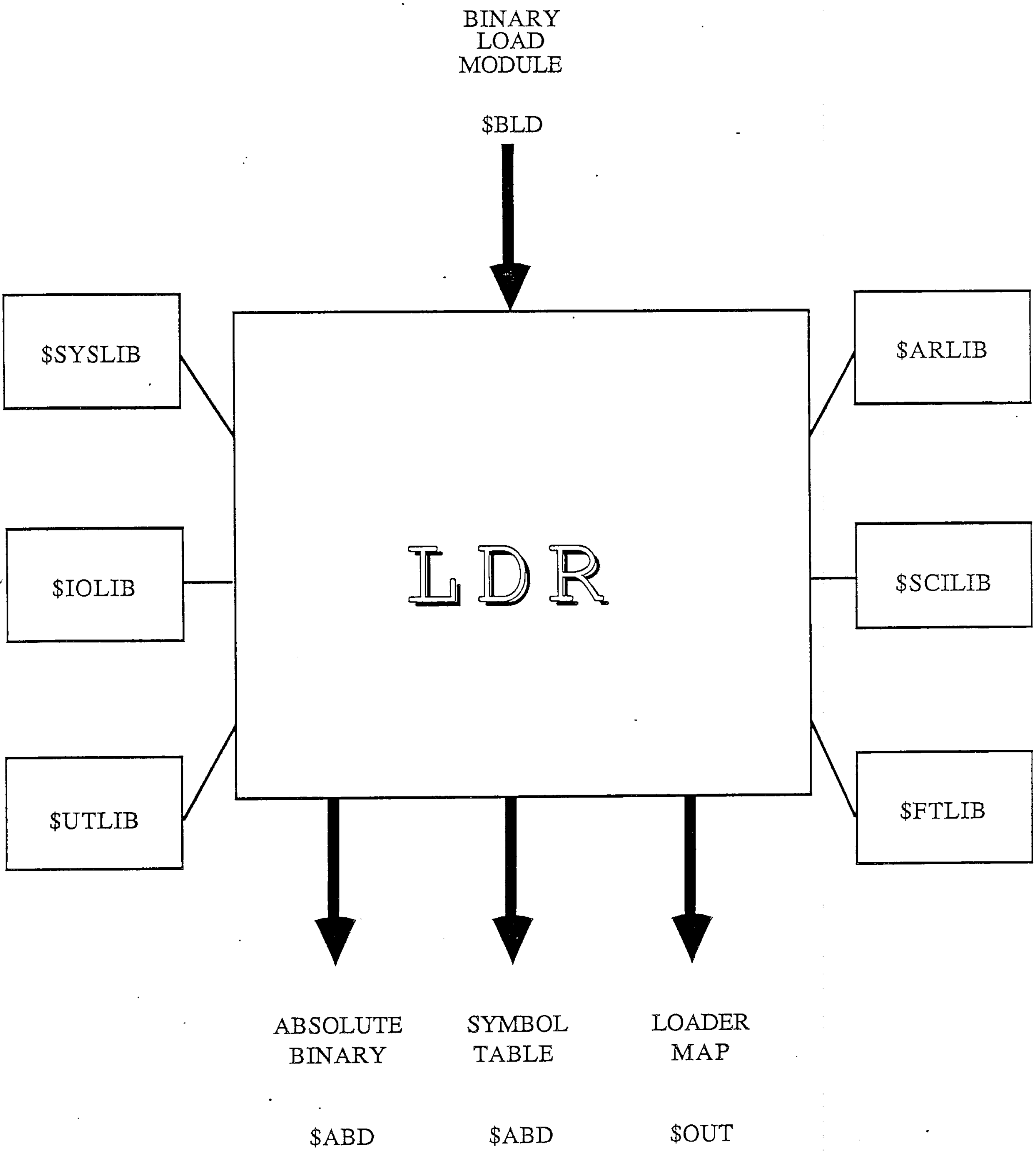
Links relative addresses together

Provides a loader map which gives addresses where each module is loaded

Both CAL and CFT use the loader

SEGLDR is the new product to replace LDR (Release 1.15)

LOADER


BINARY
LOAD
MODULE
$BLD
$SYSLIB
$IOLIB
$UTLIB
LDR
$ARLIB
$SCILIB
$FTLIB
ABSOLUTE
BINARY
$ABD
SYMBOL
TABLE
$ABD
LOADER
MAP
$OUT

INTRODUCTION QUIZ

1. Name five programs delivered with a Cray and what each does.

____________________ ________________________________

____________________ ________________________________

____________________ ________________________________

____________________ ________________________________

____________________ ________________________________

2. What is an operating system's purpose? (three things)

3. What is a local dataset?

4. What are the limitations on naming a local dataset?

5. What is a job?

6. What is the difference between a compiler and an assembler?

7. What is the difference between batch jobs and interactive jobs?

8. What is a station and its function?

9. What is the word size of a COS blocked dataset and why?

10. What four local datasets does every job have and what are each used for?

Job Control Language

2

MODULE OBJECTIVES

Upon completion of this introduction module, and with the aid of all furnished reference material, the learner should be able to:

1. Submit a job to COS
2. Copy a dataset or part of one
3. Manipulate the dataset pointers
4. Create a permanent dataset
5. Read a permanent dataset
6. Delete or modify a permanent dataset
7. Search the dataset catalog for a dataset
8. Stage a dataset to and from a front-end
9. Recognize a control block of JCL
10. Recognize a JCL procedure

JOB CONTROL LANGUAGE

The first file in $IN contains JCL for that job. Each statement is a record.

Job Definition

Control statements used in defining a job, its operating characteristics and job processing resource requirements.

Dataset Utilities

Utilities allowing the user a convenient means of copying, positioning, or initializing a local dataset.

Permanent Dataset Utilities

Utilities to archive, backup , or report status on permanent datasets.

Analytical Aids

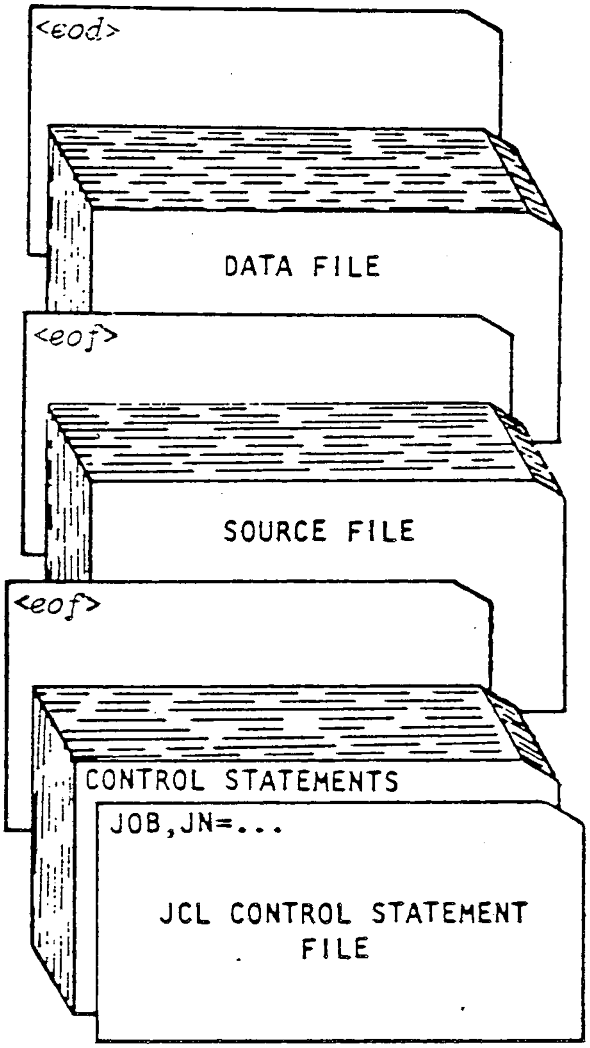
Utilities to analyze the user area and help debug a job.

Dataset Staging

Control statements to process the transfer of jobs or datasets from and to a front-end computer.

Control Statement Blocks

Allows JCL to be executed with conditional branches, looping and JCL subroutines called Procedure Libraries.


<eod>
DATA FILE
<eof>
SOURCE FILE
<eof>
CONTROL STATEMENTS
JOB,JN=...
JCL CONTROL STATEMENT
FILE

JOB Control Statement

Defines the job to the operating system. **MUST** be the first statement in a job and cannot be continued to subsequent lines.

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

The only required parameter is the job name of 1-7 characters.

Remaining parameters supply the system information about memory, time limit, priority, etc. for this job.

Examples:	Comments:
JOB,JN=JCLTST.	All parameters defaulted.
JOB,JN=JCLTST,T=8.	Time limit of 8 seconds.
JOB,JN=JCLTST,P=7.	Priority of 7.
JOB,JN=JCLTST,US=TNG.	User number of TNG.
JOB,JN=JCLTST,SSD=500.	Asking for 500 blocks of SSD storage.

ACCOUNT Statement

Validates the user's account number and optional password. The statement must immediately follow the JOB statement if accounting is mandatory.

```
ACCOUNT,AC=ac,PW=pw,NPW=npw,US=us,UPW=upw,NUPW=nupw,
        APW=apw,NAPW=napw.
```

The only required parameter is the account number of 1-15 alphanumeric characters. If COS security is enabled the UPW parameter is also required.

```
16:53:42   0.0000  CSP   ---------------------------------------------------------------------------------------
16:53:42   0.0000  CSP
16:53:42   0.0000  CSP    ***  Use the    NEWS     control statement for general CRAY news. ***
16:53:42   0.0000  CSP
16:53:42   0.0000  CSP   03/14/84 - The amount of BUFFER MEMORY available to the user
16:53:42   0.0000  CSP              as DVN=BMR-0-20 on the X-MP is now 368 tracks. It
16:53:42   0.0000  CSP              was previously 383 tracks.
16:53:42   0.0000  CSP
16:53:42   0.0001  CSP            - An archive of datasets not accessed in 21 days was
16:53:42   0.0001  CSP              run at 1230, 03/14.
16:53:42   0.0001  CSP
16:53:42   0.0001  CSP   03/13/84 - There will be no BATCH on the X-MP between the hours
16:53:42   0.0001  CSP              of 0945 - 1115 on Wednesday, March 21, due to a demo.
16:53:42   0.0001  CSP
16:53:42   0.0001  CSP   ---------------------------------------------------------------------------------------
16:53:42   0.0001  CSP           CRAY X-MP SERIAL-101/6     CRI - MENDOTA HEIGHTS, MINN. 03/19/84
16:53:42   0.0001  CSP
16:53:42   0.0001  CSP       _   CRAY OPERATING SYSTEM           COS X.13  ASSEMBLY DATE 03/14/84
16:53:42   0.0001  CSP
16:53:42   0.0001  CSP
16:53:42   0.0001  CSP   JOB,JN=U1502A.                                                             00010038
16:53:43   0.0008  CSP   ACCOUNT,AC=,US=,UPW=.                                            00020012
16:53:45   0.0025  CSP   CAL.                                                                       00030015
16:53:45   0.0027  USER  CA001 - [CAL] CAL VERSION X.13 (03/12/84) - CRAY XMP
16:53:46   0.5070  USER  CA002 - [CAL] ASSEMBLY TIME:     0.5044 CPU SECONDS
16:53:46   0.5071  USER  CA003 - [CAL] MEMORY WORDS: 78205 + I/O BUFFERS:  6348
16:53:48   0.5076  CSP   LDR,MAP.                                                                   00040042
16:53:53   0.5724  USER  LD000 - BEGIN EXECUTION
16:53:54   0.5725  EXP   DUMPJOB.                                                                   00050038
16:53:55   0.5730  CSP   DUMP,JTA,FW=200,LW=300,DSP,CENTER.                                         00060042
16:54:02   0.6160  CSP   END OF JOB
16:54:02   0.6160  CSP
16:54:02   0.6160  CSP
16:54:02   0.6162  USER      JOB NAME -                          U1502A
16:54:02   0.6162  USER      USER NUMBER -                       CRT
16:54:02   0.6163  USER      TIME EXECUTING IN CPU -             0000:00:00.6162
16:54:02   0.6163  USER      TIME WAITING TO EXECUTE -           0000:00:07.2187
16:54:02   0.6163  USER      TIME WAITING FOR I/O -              0000:00:12.4334
16:54:02   0.6163  USER      TIME WAITING IN INPUT QUEUE -       0000:02:29.4690
16:54:02   0.6163  USER      MEMORY * CPU TIME (MWDS*SEC) -            172.41110
16:54:02   0.6164  USER      MEMORY * I/O WAIT TIME (MWDS*SEC) -         0.65008
16:54:02   0.6164  USER      MINIMUM JOB SIZE (WORDS) -             28160
16:54:02   0.6164  USER      MAXIMUM JOB SIZE (WORDS) -             88576
16:54:02   0.6164  USER      MINIMUM FL (WORDS) -                   24576
16:54:02   0.6164  USER      MAXIMUM FL (WORDS) -                   85504
16:54:02   0.6164  USER      MINIMUM JTA (WORDS) -                   3072
16:54:02   0.6164  USER      MAXIMUM JTA (WORDS) -                   3584
16:54:02   0.6164  USER      DISK SECTORS MOVED -                     585
16:54:02   0.6164  USER      USER I/O REQUESTS -                      110
16:54:02   0.6164  USER      OPEN CALLS -                              30
16:54:02   0.6164  USER      CLOSE CALLS -                             30
16:54:02   0.6164  USER      MEMORY RESIDENT DATASETS -                 0
16:54:02   0.6165  USER      TEMPORARY DATASET SECTORS USED -          15
16:54:02   0.6165  USER      PERMANENT DATASET SECTORS ACCESSED -      72
16:54:02   0.6165  USER      PERMANENT DATASET SECTORS SAVED -          0
16:54:02   0.6165  USER      SECTORS RECEIVED FROM FRONT END -          0
16:54:02   0.6165  USER      SECTORS QUEUED TO FRONT END -              0
```

COPYR, COPYF, COPYD Statements

Copies a specified number of records, files, or a dataset to another dataset beginning at the current dataset pointer position.

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

Following the copy, the dataset pointer is positioned **AFTER** the EOR of the copied record.

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

Following the copy, the dataset pointer is positioned **AFTER** the EOF of the copied field.

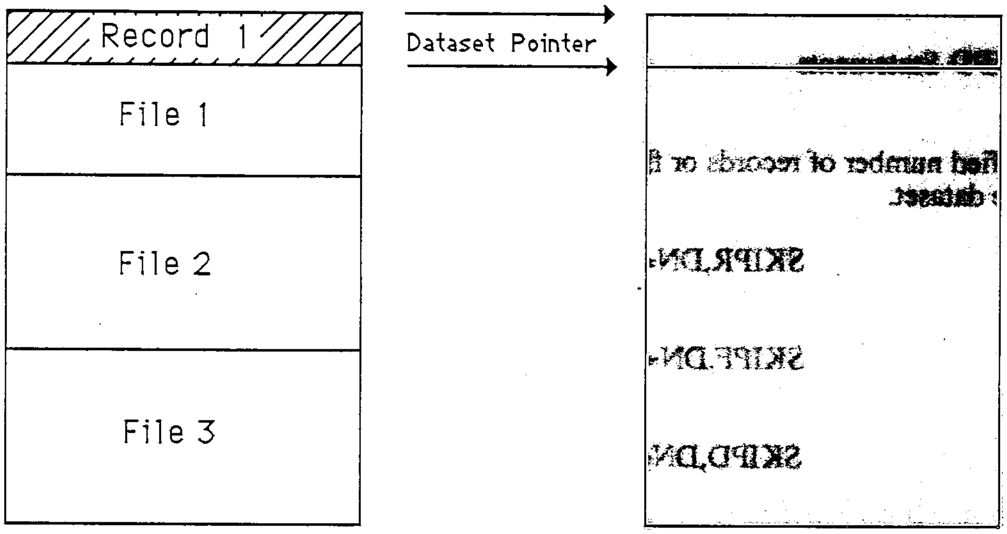
```
COPYD,I=idn,O=odn.
```

Following the copy, the dataset pointer is positioned **AFTER** the EOF of the last copied file. EOD is **NOT** written.

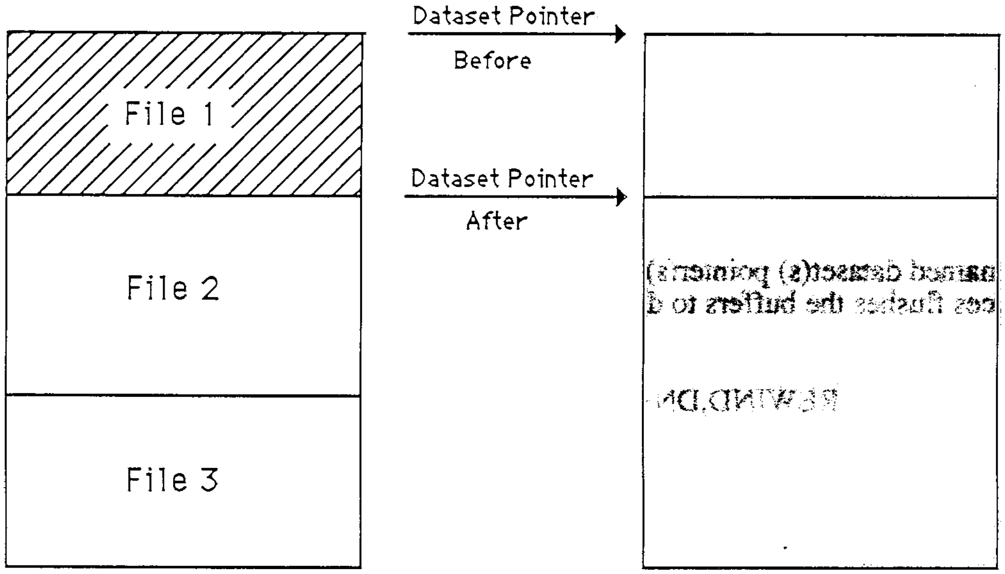
The copy utilities are for use with CRAY blocked datasets.

Examples	Comments:
COPYD,I=$IN,O=SWCE1.	Copies $IN to SWCE1.
COPYF,I=SWCE1,NF=2.	Copies 2 files in SWCE1 from the current pointer.
COPYR,NR=1.	Copies 1 record in $IN (default) from the current pointer.

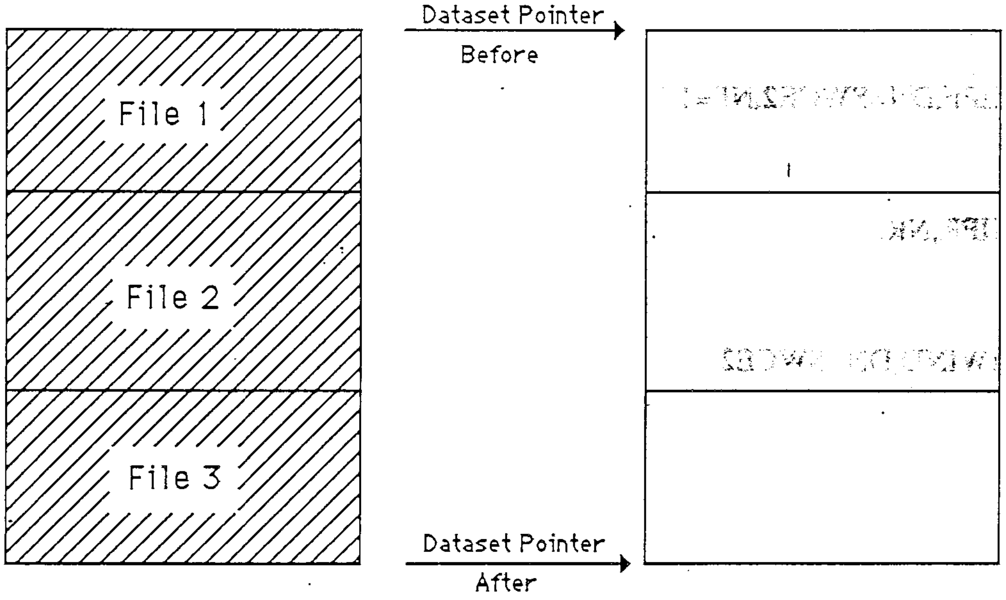
COPYR



COPYF



COPYD



SKIPR, SKIPF, SKIPD Statements

Skips a specified number of records or files. It can also position a dataset pointer after the last file of the dataset.

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

SKIPF,DN=dn,NF=n.

SKIPD,DN=dn.
```

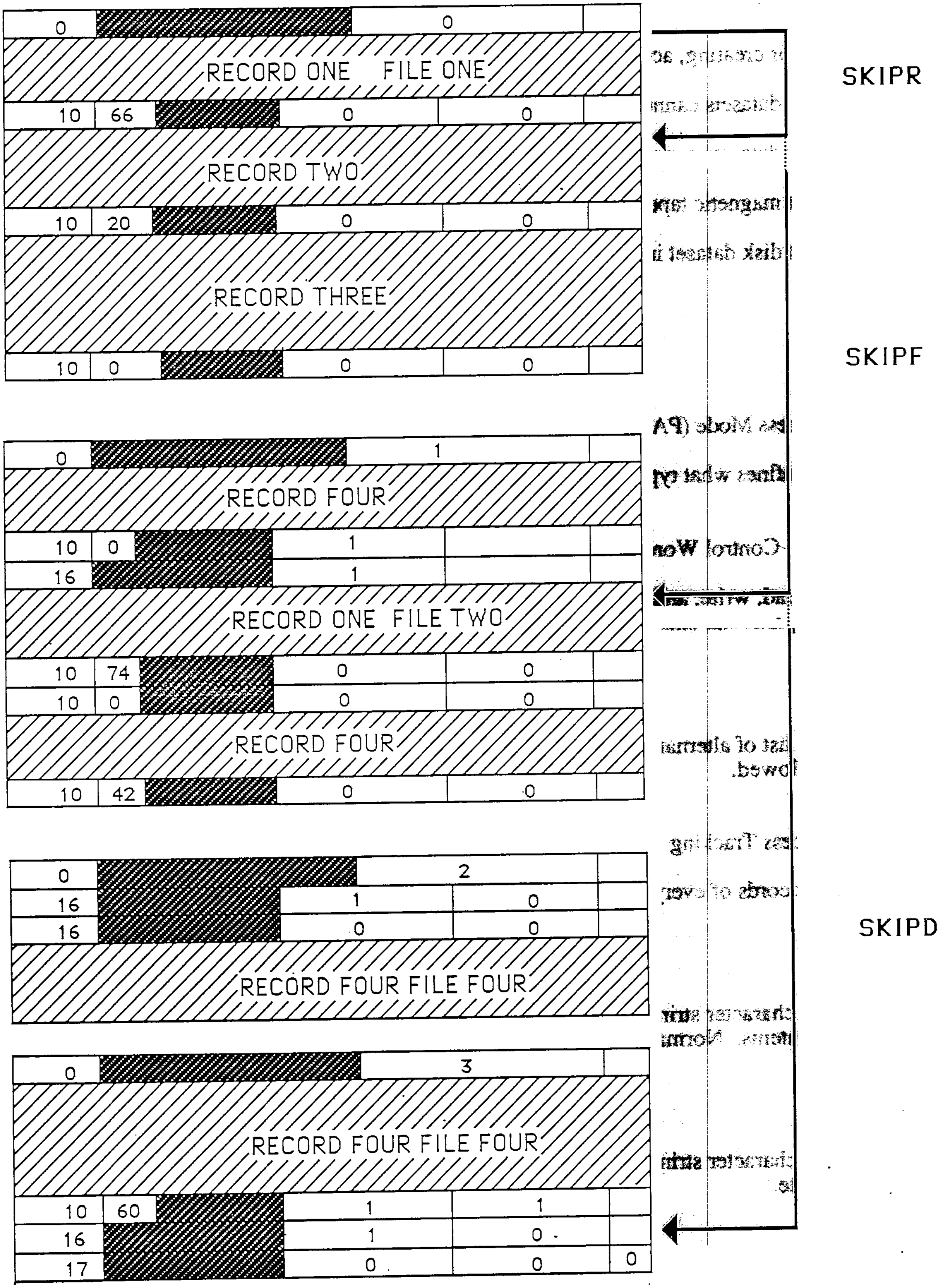
The skip utilities are for use with CRAY blocked datasets.

REWIND Statement

Positions the named dataset(s) pointer(s) prior to the first block control word (BCW) and in some intances flushes the buffers to disk.

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

Examples:	Comments:
SKIPF,DN=SWCE2,NF=1.	Skips 1 file on SWCE1 from current pointer.
SKIPR,NR.	Positions the pointer after the last record of the current file in $IN.
REWIND,DN=SWCE2.	Rewinds SWCE2 to the beginning of its first record.


0
0
RECORD ONE FILE ONE
10
66
0
0
RECORD TWO
10
20
0
0
RECORD THREE
10
0
0
0
0
1
RECORD FOUR
10
0
1
16
1
RECORD ONE FILE TWO
10
74
0
0
10
0
0
0
RECORD FOUR
10
42
0
0
0
2
16
1
0
16
0
0
RECORD FOUR FILE FOUR
0
3
RECORD FOUR FILE FOUR
10
60
1
1
16
1
0
17
0
0
0
SKIPR
SKIPF
SKIPD

PERMANENT DATASET MANAGEMENT

Provides for creating, accessing, and protecting disk permanent datasets.

Permanent datasets cannot be destroyed by normal system activity or engineering maintenance.

Permanent datasets cannot be affected by front-end systems.

Permanent magnetic tape dataset information can be maintained on a front-end system.

Permanent disk dataset information is maintained on disk in a dataset catalog (DSC).

PERMANENT DATASET ATTRIBUTES

Public Access Mode (PAM)

Defines what type of minimum access all users can have to a particular dataset.

Permission Control Words

Read, write, and maintenance passwords that, if used, must be supplied to gain access to a particular dataset in the mode desired.

Permits

A list of alternate users of a particular dataset and which PAM each alternate user is allowed.

Public Access Tracking

Records of every user who accesses a public access dataset can be maintained.

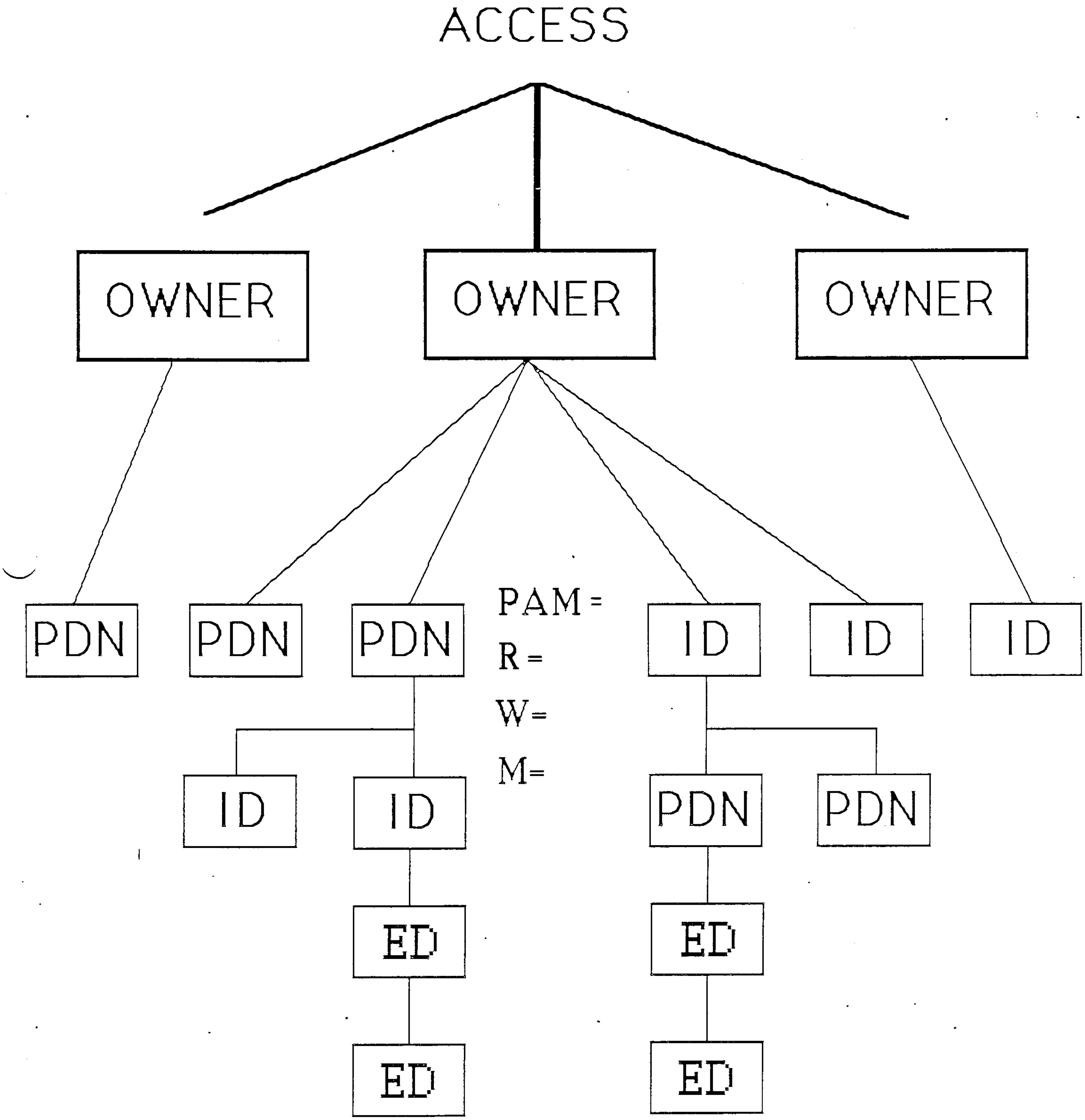
Text

A character string passes to a front-end system during transfers of datasets between systems. Normally contains instructions to the front-end.

Notes

A character string of up to 480 characters. There is no restriction as to the contents of the note.

Permanent Dataset



ACCESS

OWNER

OWNER

OWNER

PDN

PDN

PDN

PAM =

R =

W=

M=

ID

ID

ID

ID

ID

PDN

PDN

ED

ED

ED

ED

SAVE Statement

Creates an initial or additional disk dataset catalog (DSC) entry for a local dataset making a new edition of a permanent dataset.

```
SAVE,DN=dn,PDN=pdn,ID=id,ED=ed,RT=rt,R=rd,W=wt,M=mn,UQ,
     NA,EXO=ON/OFF,PAM=mode,ADN=adn(m),TA=opt,
     TEXT=text,NOTES=notes.
```

The only required parameter is the dataset name of 1-7 alphanumeric characters, but it is recommended that you use an ID.

Examples:

```
SAVE,DN=SWCE1,ID=TNG00,PAM=R.
```

(Makes local dataset SWCE1 a permanent dataset with the same name and gives it an ID of TNG00 and a public access mode of READ.)

```
SAVE,DN=A,PDN=SWCE3,ID=TNG00,M=TNG.
```

(Makes local dataset A a permanent dataset with the name of SWCE3 and gives it an ID of TNG00 and a maintenance control word of TNG.)

```
SAVE,DN=C,PDN=ABSOLUTELY,ID=TNG00,R=ME.
```

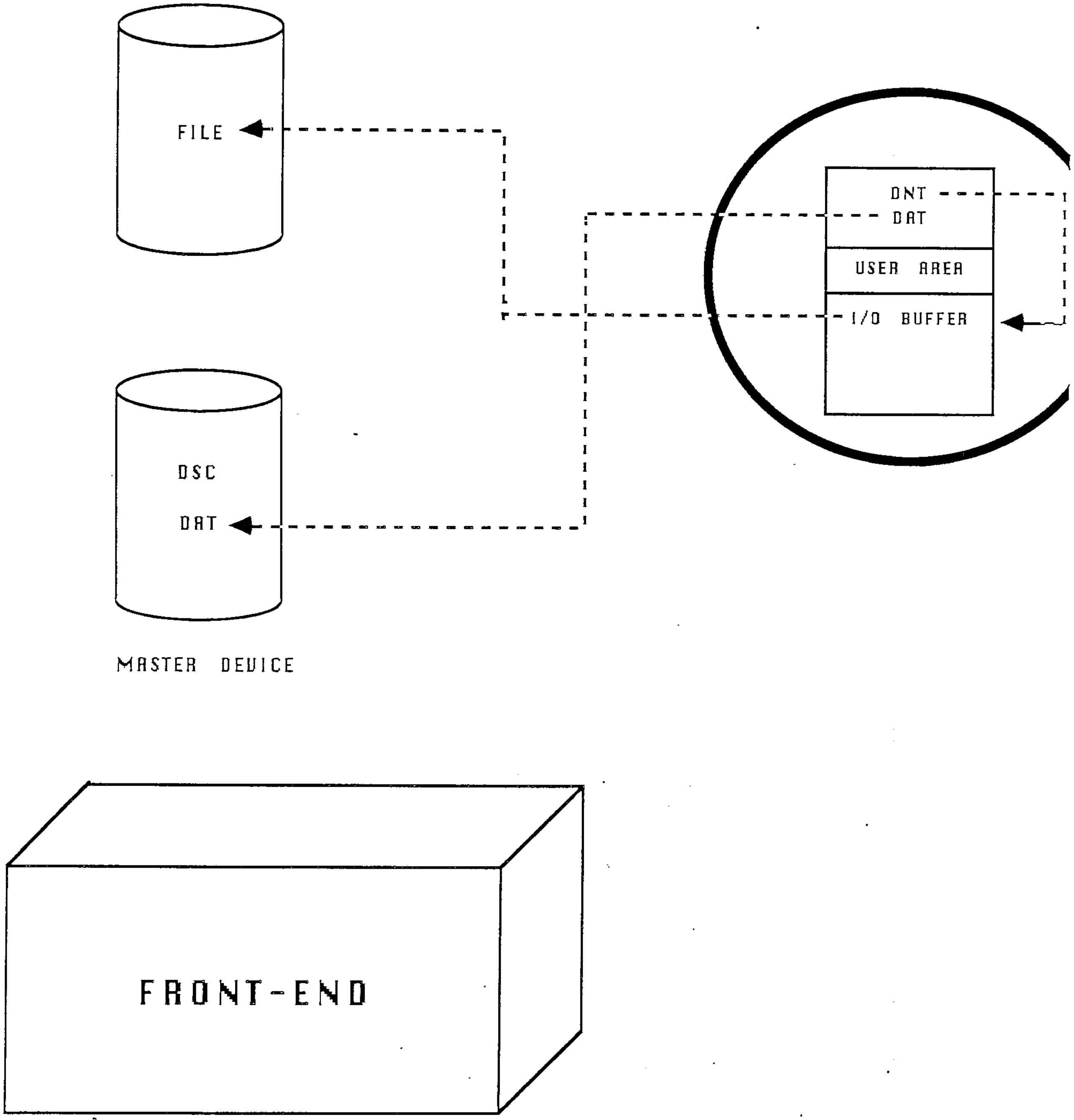
(Makes local dataset C a permanent dataset with the name of ABSOLUTELY and gives it an ID of TNG00 and a read control word of ME.)

```
SAVE,DN=MINE,PAM=R,ID=TNG00,M=NO.
```

(Makes local dataset MINE a permanent dataset with the same name and gives it a public access mode of READ, an ID of TNG00, and a maintenance control word of NO.)

SAVE

SAVE,DN=MINE,PDN=FILE,ID=TNG01,R=TNG,PAM=R.



ACCESS Statement

Allows the user to make an existing permanent dataset local to a job by assuring the user is authorized to use the dataset and copying DSC information to a user JTA area.

ACCESS,DN=dn,PDN=pdn,ID=uid,ED=ed,R=rd,W=wt,M=mn,
UQ,LE,NA,OWN=ov,CS=cs,DF=df,DT=dt,
FSEC/VSEQ=fsec,LB=lb,MBS=mbs,NEW,MF=FES,
RS=rs,DEN=den, XDT=yyddd,RT=rt,CT=ct,RF=rf,
VOL=vol$_1$:vol$_2$:...VOL$_n$.

The only required parameter is the dataset name of 1-7 alphanumeric characters.

Examples:

```
ACCESS,DN=SWCE1,ID=TNGSWCE.
```

(Makes the permanent dataset SWCE1 with an ID of TNGSWCE local to a job and gives it a local dataset name of SWCE1.)

```
ACCESS,DN=A,PDN=CRAY1SYSTEMDUMP,R=READDUMP,ED=268.
```

(Makes the Cray Systemdump dataset local to a job and calls it A. The PDN has a READ control word of READDUMP and an EDITION number of 268.

```
ACCESS,DN=A,PDN=CRAY1SYSTEMDUMP,M=MAINDUMP,UQ,
      ED=103.
```

(Makes the Cray Systemdump dataset local to a job and calls it A. The PDN has a MAINTENANCE control word of MAINDUMP and an EDITION number of 103.

RELEASE Statement

Relinquishes access to a permanent dataset or removes a local dataset from the job area.

RELEASE,DN=dn$_1$:dn$_2$:dn$_3$,HOLD.

Examples:

```
RELEASE,DN=A.
```

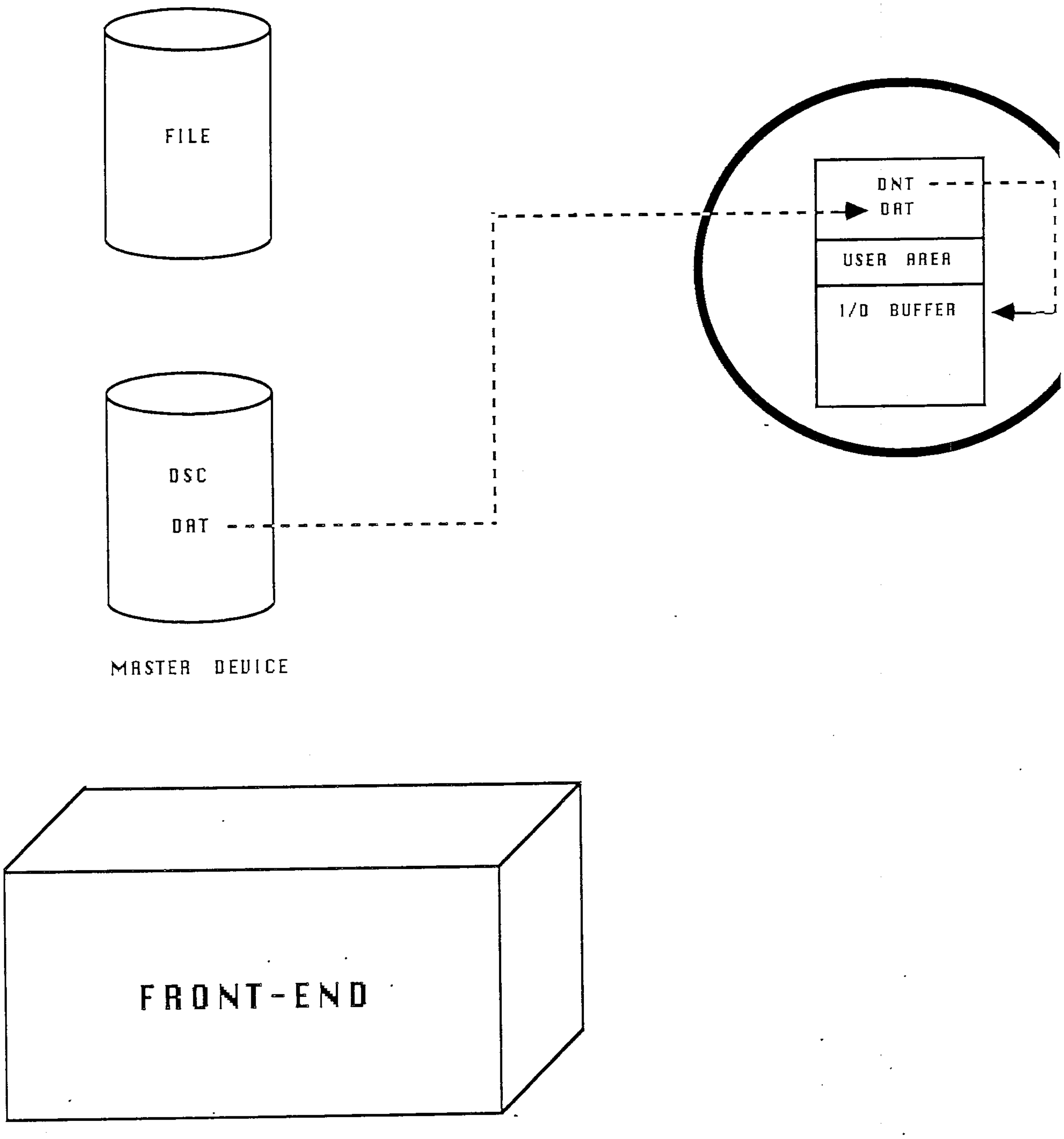
(Removes local dataset A from the job area.)

```
RELEASE,DN=$BLD.
```

(Removes local dataset $BLD from the job area.)

ACCESS

ACCESS,DN=MINE,PDN=FILE,R=PASS,UQ.



MODIFY Statement

Alters permanent dataset information established in the DSC when the dataset has been accessed with unique permission (UQ).

```
MODIFY,DN=dn,PDN=pdn,ID=uid,ED=ed,RT=rt,R=rd,W=wt,
          M=mn,NA,EXO=ON/OFF,PAM=mode,ADN=adn(m),
          TA=opt,TEXT=text,NOTES=notes.
```

The only required parameter is the dataset name of 1-7 characters.

Examples:

```
MODIFY,DN=A,PAM=R.
    (Alters permanent dataset A to give it a PUBLIC ACCESS MODE of
     READ.)

MODIFY,DN=SWCE1,ID=TNG.
    (Alters permanent dataset SWCE1 to give it an ID of TNG.)

MODIFY,DN=GENPL,RT=40,ED=1.
    (Alters permanent dataset GENPL to give it a RETENTION period of
     40 days and an EDITION number of 1.)
```

DELETE Statement

Removes a permanent dataset from the DSC when the dataset has been accessed with unique access (UQ) and possibly maintenance permission.

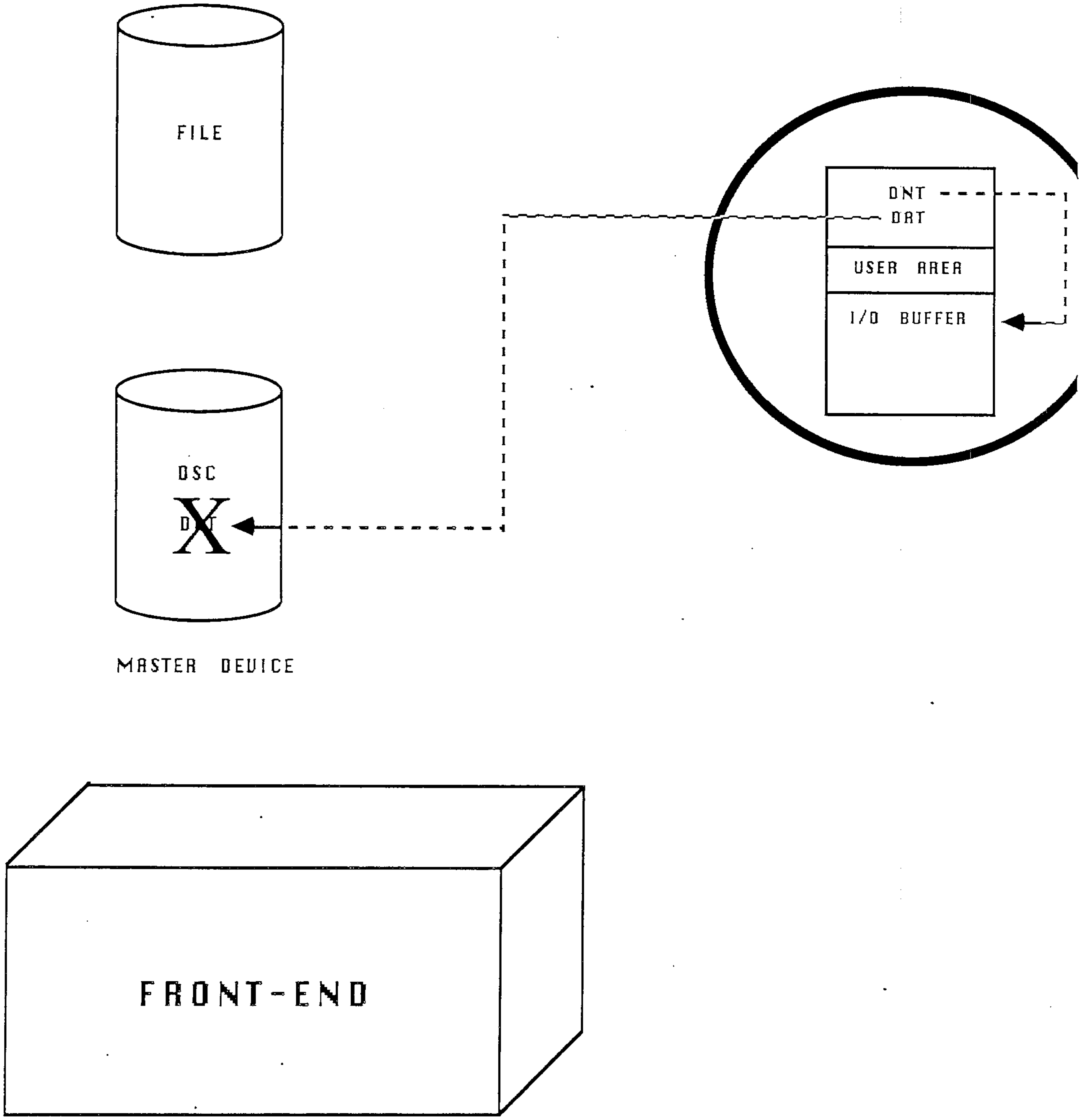
```
DELETE,DN=dn,NA,PARTIAL.
```

Example:

```
DELETE,DN=A.
    (Removes any knowledge of the permanent dataset A from the DSC.)
```

DELETE

ACCESS,DN=MINE,PDN=FILE,UQ.
DELETE,DN=MINE.



AUDIT Statement

Provides a listed report on the status of each specified permanent dataset known to the system (on the DSC).

```
AUDIT,L=ldn,B=bdn,PDN=pdn,ID=uid,US=usn,DV=dvn,SZ=dsz,
        X=mm/dd/yy:'hh:mm:ss',TCR=mm/dd/yy':'hh:mm:ss,
        CW=ccw,OWN,ov,LO=opt:opt,BO=opt:opt.
```

The listed report will be those datasets that match the user number.

Example:	Comments:
AUDIT.	Lists all permanent datasets.
AUDIT,PDN=JCLTST,ID=TNG.	Lists only name JCLTST with an ID of TNG.
AUDIT,PDN=JCL-,ID=TNG.	Lists all names beginning with JCL which have an ID of TNG.

```
AUDIT               COS X.13              03/14/84  14:20:11          PAGE       1

 OWN = U9909
 ID     = DIAGSYS

------------------------------------------------------------------------------------------
PDN                  ID               ED                    LAST       LAST      LAST        DEVICE
            SZ      RT     ACC  TA  PAM     CREATED  ACCESSED  MODIFIED   DUMPED
------------------------------------------------------------------------------------------

ARBU                  DIAGSYS            1   02/22/84  03/09/84              03/10/84   DD-A2-20
          1024     45        51  N   R       13:28:58  16:04:32              13:26:26

CMXU                  DIAGSYS            1   03/07/84  03/09/84              03/10/84   DD-A1-23
          3584     45         8  N   R       08:35:08  13:22:49              13:17:06

CRAYPL                DIAGSYS           16   03/08/84  03/14/84              03/10/84   DD-A2-21
        757486     45        73  N   RWM     13:51:58  14:13:31              13:15:22

CRAYPL                DIAGSYS           17   03/13/84  03/14/84              03/13/84   DD-A2-23
        704000     45         5  N   RWM     14:33:04  14:12:27              21:12:51

ECD                   DIAGSYS           12   03/13/84  03/13/84              03/13/84   DD-A1-22
         26010     45         1  N   RWM     14:34:29  14:34:29              21:13:58

ECD                   DIAGSYS           13   03/13/84  03/14/84              03/13/84   DD-A1-22
         26010     45         7  N   RWM     15:12:01  08:41:45              21:14:57

FGA                   DIAGSYS            1   02/28/84  03/14/84              03/10/84   DD-A2-22
         24916     45        60  N   E       18:15:52  07:23:11              13:21:09

FGLIST                DIAGSYS            1   02/29/84  03/14/84              03/10/84   DD-A2-25
         19369     45        54  N   E       14:27:23  14:01:19              13:18:41

I200PL                DIAGSYS            1   01/10/84  03/14/84              03/10/84   DD-A1-21
        461312     45        26  N   R       08:11:20  14:03:45              13:22:34

IOPPL                 DIAGSYS           10   03/13/84  03/13/84              03/13/84   DD-A1-22
        460288     45         1  N   RWM     14:30:40  14:30:40              21:13:09

IOPPL                 DIAGSYS           11   03/13/84  03/14/84              03/13/84   DD-A2-20
        460288     45         7  N   RWM     15:09:23  14:05:32              21:14:57

X200PL                DIAGSYS            2   03/07/84  03/14/84              03/10/84   DD-A2-23
        573865     45       126  N   R       12:37:29  14:17:19              13:23:27

        12 DATASETS,      6873 BLOCKS,            3518152 WORDS
```

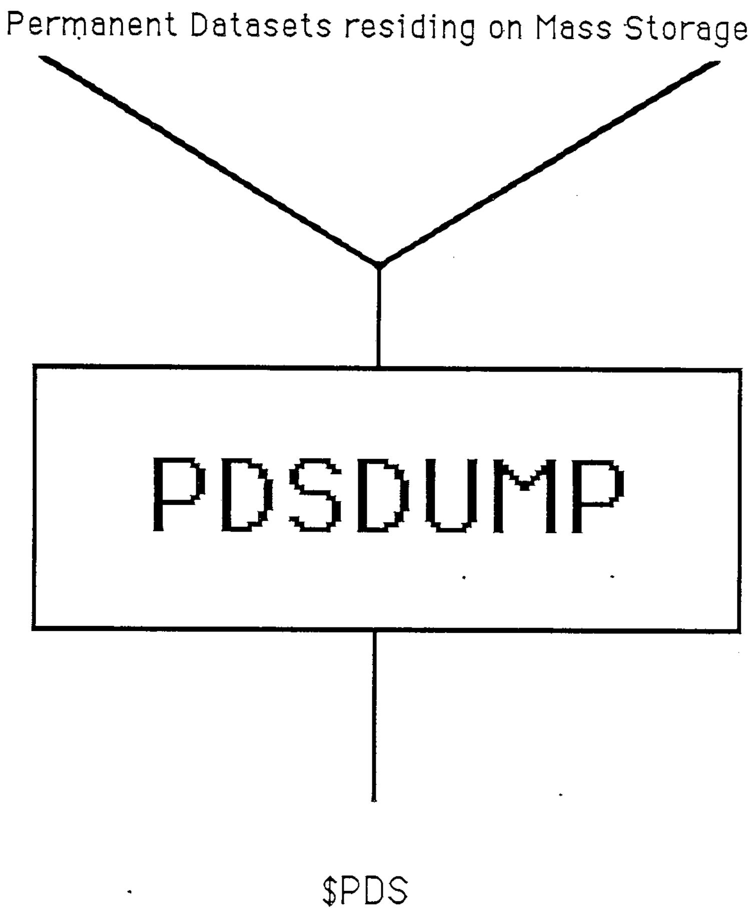
PDSDUMP Statement

Dumps a specified permanent dataset to another dataset that may then be saved or staged to a front-end.

```
PDSDUMP,DN=dn,DV=ldv,PDN/PDS=$pds,CW=cw=ID=uid,US=usn,
        ED=ed,X,C,D,I,O,S,SO,INC=mn/dd/yy:'hh:mm:ss',
        OWN=ov,TX=opt,ARC=mmddyy:'hh:mm:ss'.
```

By use of certain parameters the utility provides backup datasets or a convenient method of deleting groups of permanent datasets.

Examples:	Comments:
PDSDUMP,PDN=JCLTST,D.	Copies JCLTST to $PDS and deletes JCLTST from the DSC.
PDSDUMP,PDN=JCLTST,ID=TNG.	Copies JCLTST with an ID of TNG to $PDS.
PDSDUMP,PDN=JCLTST,D,X.	Copies JCLTST to $PDS if expired and deletes it from the DSC.


Permanent Datasets residing on Mass Storage
PDSDUMP
$PDS

PDSLOAD Statement

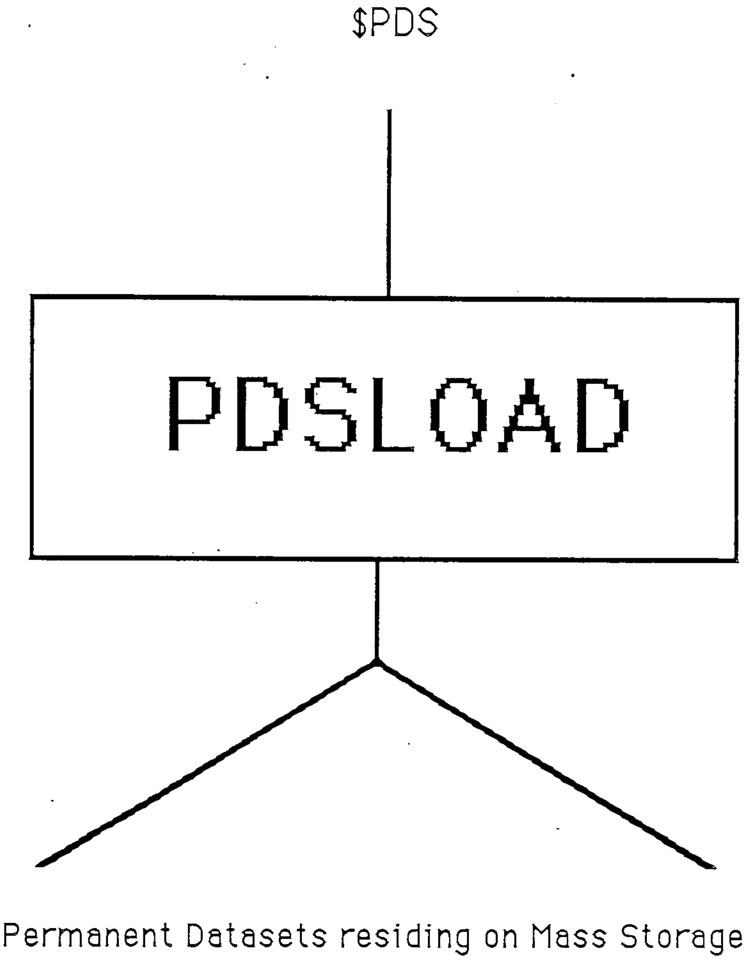
Loads (creates DSC entries) permanent datasets from a dataset created by the PDSDUMP utility.

```
PDSDUMP,DN=dn,PDN/PDS=$pds,CW=cw,ID=uid,US=usn,
        ED=ed,A,I,O,S,OWN=ov,NOWN=nov,DV=dvn,RP,
        CR,NA,SO.
```

If the dataset already exists in the DSC no action is taken.

These two utilities (PDSDUMP, PDSLOAD) are used to archive permanent datasets on a front-end and restore the DSC.

Example jobs:	Comment:
JOB-------. PDSDUMP,US=CRT.	Dumps all permanent datasets with a user name of CRT.
DISPOSE,DN=$PDS -----. EXIT.	
JOB------. ACQUIRE,DN=$PDS------. PDSLOAD.	Loads the datasets in $PDS into the DSC.
EXIT. /EOF	


$PDS
PDSLOAD
Permanent Datasets residing on Mass Storage

DUMPJOB Statement

Causes creation of a local dataset named $DUMP and forces the entire user area (JTA-LA) to be written to disk as an unblocked dataset.

```
DUMPJOB.
```

The DUMPJOB statement cannot be the first statement following a job statement or be used for execute-only datasets. By convention it should be used following the EXIT statement to aid in debugging an abort condition.

Once $DUMP is created it may be used with the statements DUMP, DEBUG, and FLODUMP.

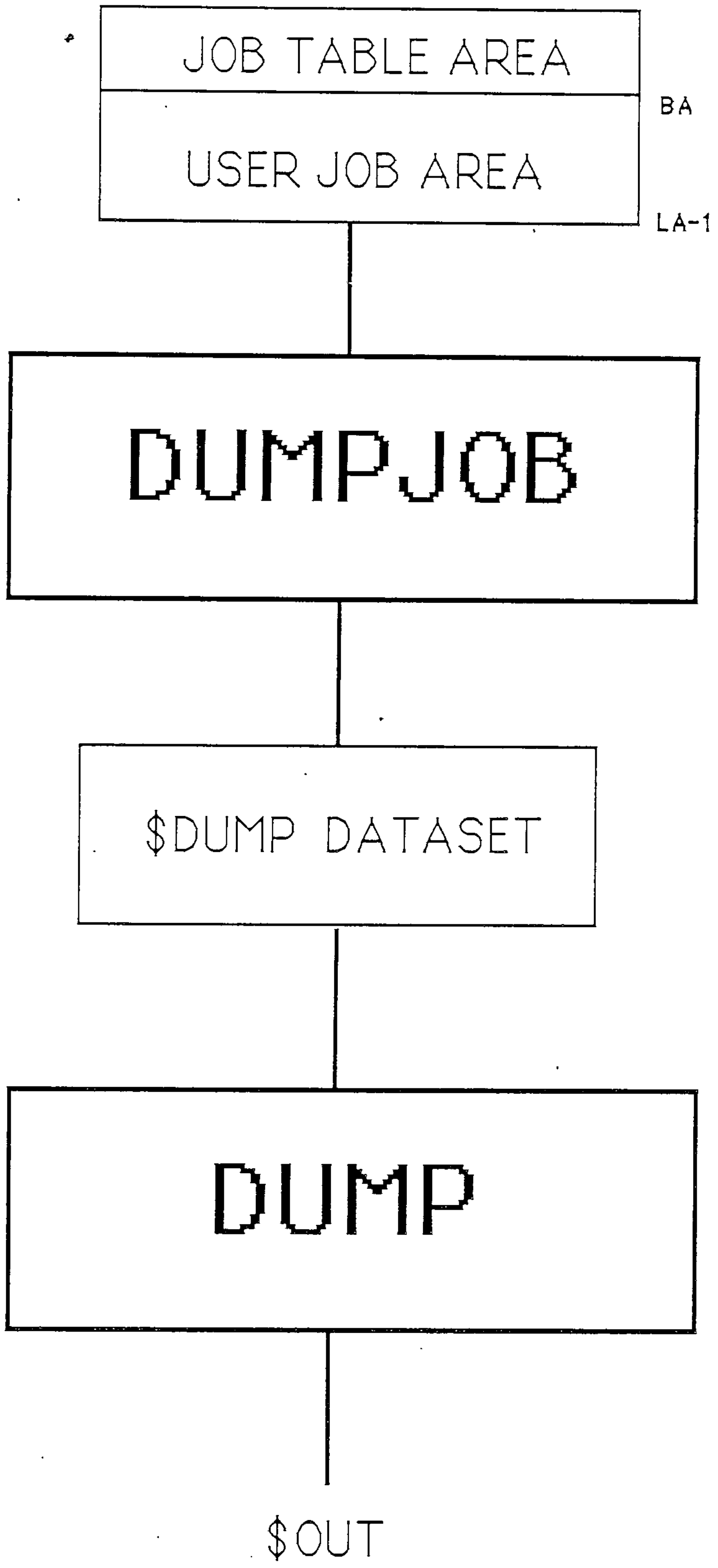
DUMP Statement

Reads and formats selected portions of the dataset $DUMP and writes the information to another dataset, normally $OUT.

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

By convention this statement usually follows the EXIT and DUMPJOB statement in the case of a job step abort.

Example:	Comment:
JOB, -----.	
....	
....	
....	
....	
EXIT.	Normal job end.
DUMPJOB.	Dumps entire user area to $DUMP in the event of a job abort.
DUMP.	Dumps words 0-200 of the user area (by default).


JOB TABLE AREA
BA
USER JOB AREA
LA-1
DUMPJOB
$DUMP DATASET
DUMP
$OUT

DSDUMP Statement

Dumps specified portions of a dataset to another dataset in either blocked or unblocked format.

```
DSDUMP,I=idn,O=odn,DF=df,IW=n,NW=n,NR=n,IF=n,IS=n,NS=n,NF=n.
```

The only required parameter is the input dataset name of 1-7 alphanumeric characters.

Examples:	Comments:
REWIND,DN=SWCE3.	Positions pointer at beginning of SWCE3.
DSDUMP,I=SWCE3,DF=B,NW,NR,NF.	Dumps all words of all files of dataset SWCE3 to $OUT.
REWIND,DN=SWCE3.	Positions pointer at beginning of SWCE3.
DSDUMP,I=SWCE3.	Dumps only word one of the first record of the first file in dataset SWCE3.

```
A                                                              DSDUMP 1.13 84158     06/14/84  09:59:12     PAGE     1

       SECTOR      1

000001  0000000000000000000012 0201012044150421243107 0441112244551423247117 0501212445152425253127        ABCDEFGHIJKLMNOPQRSTUVW
000005  0541312642004010020040 0200401002004010020040 0200401002004010020040 0200401002004010020040 XYZ
000009  0200401002004010020040 0200401002004010020040 0200401002004010020040 1000000000000000000012
000013  0200572124750610020040 0200401002004010020040 0200401002004010020040 0200401002004010020040  /EOF
000017  0200401002004010020040 0200401002004010020040 0200401002004010020040 0200401002004010020040
000021  0200401002004010020040 0200401002004010020040 1000000000000000000012 0200601423106315032466                   0123456
000025  0334701622004010020040 0200401002004010020040 0200401002004010020040 0200401002004010020040 789
000029  0200401002004010020040 0200401002004010020040 0200401002004010020040 0200401002004010020040
000033  0200401002004010020040 1000000000000000000012 0200572124750610020040 0200401002004010020040              /EOF
000037  0200401002004010020040 0200401002004010020040 0200401002004010020040 0200401002004010020040
        *****
000045  1000000000000000000012 0201062224610510052110 0511052122012221241517 0511041004751621220040        FILE THREE RECORD ONE
000049  0200401002004010020040 0200401002004010020040 0200401002004010020040 0200401002004010020040
000053  0200401002004010020040 0200401002004010020040 0200401002004010020040 1000000000000000000000
000057  1600000000000000000000 1600000000000000000000 1700000000000000000000 0000000000000000000000
000061  0000000000000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000
        *****
000509  0000000000000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000

        **** END OF DSDUMP ****
```

FETCH Statement

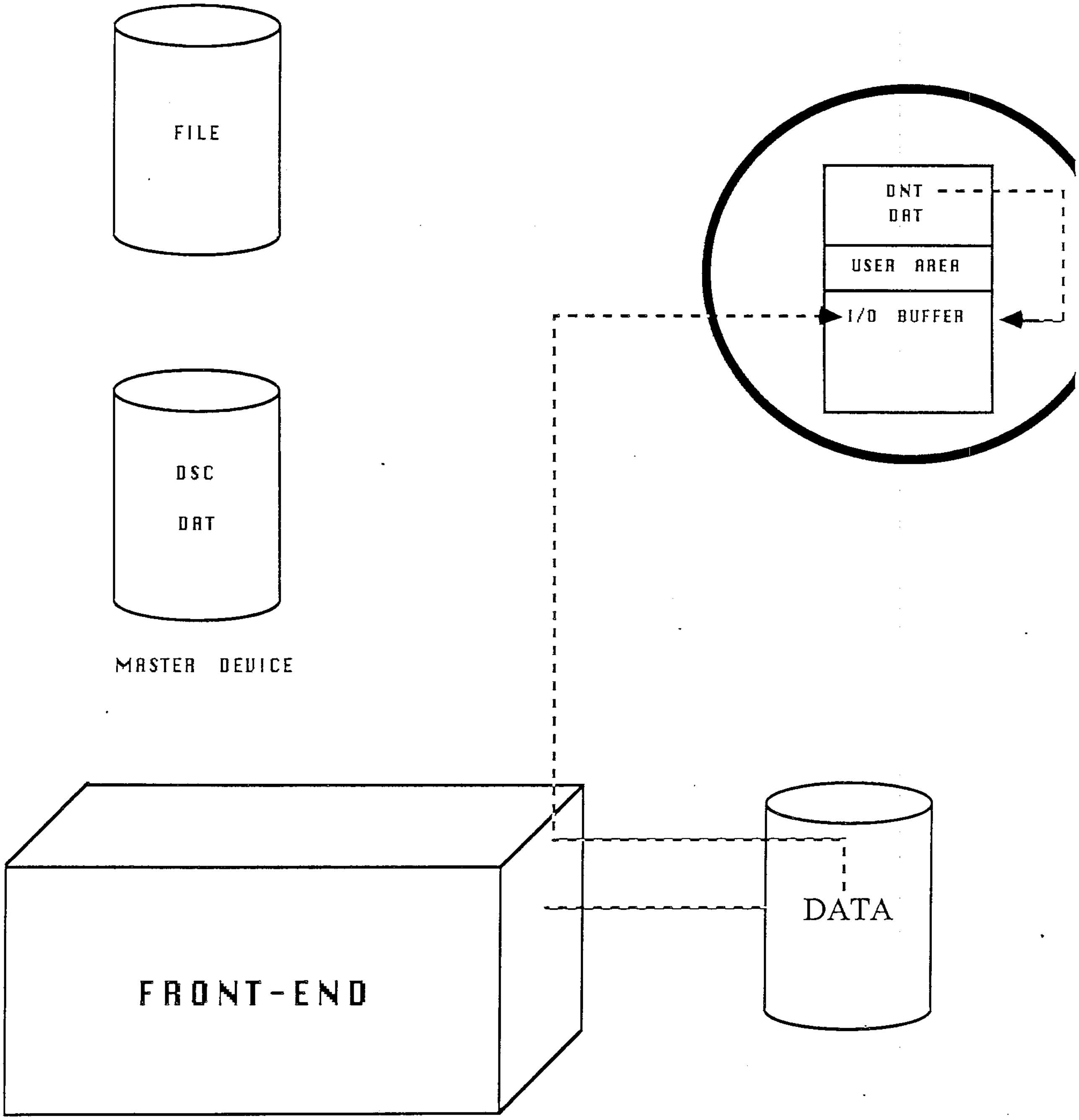
Allows the user to make a dataset stored on a front-end local to a job. It DOES NOT make the dataset permanent on the Cray system.

```
FETCH,DN=dn,SDN=sdn, TEXT=text,MF=mf,TID=tid,DF=df.
```

Example:	Comment:
FETCH,DN=$PROC,MF=M4,TEXT=MT0:0.	A copy of $PROC is "fetched" from front-end M4 and is made local to the job. The text field is code for the front-end.

FETCH

FETCH,DN=DATA,MF=M4,TEXT='DSN= '.



ACQUIRE Statement

Allows the user to make a dataset stored on a front-end local to a job and at the same time makes it a permanent dataset on Cray mass storage. The statement causes a search of Cray mass storage for the dataset before it looks to the front-end for it.

```
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,OWN=own,PAM=mode,ADN=adn(m),TA=opt,
          NOTES=notes.
```

The only required parameter is the dataset name of 1-7 alphanumeric characters.

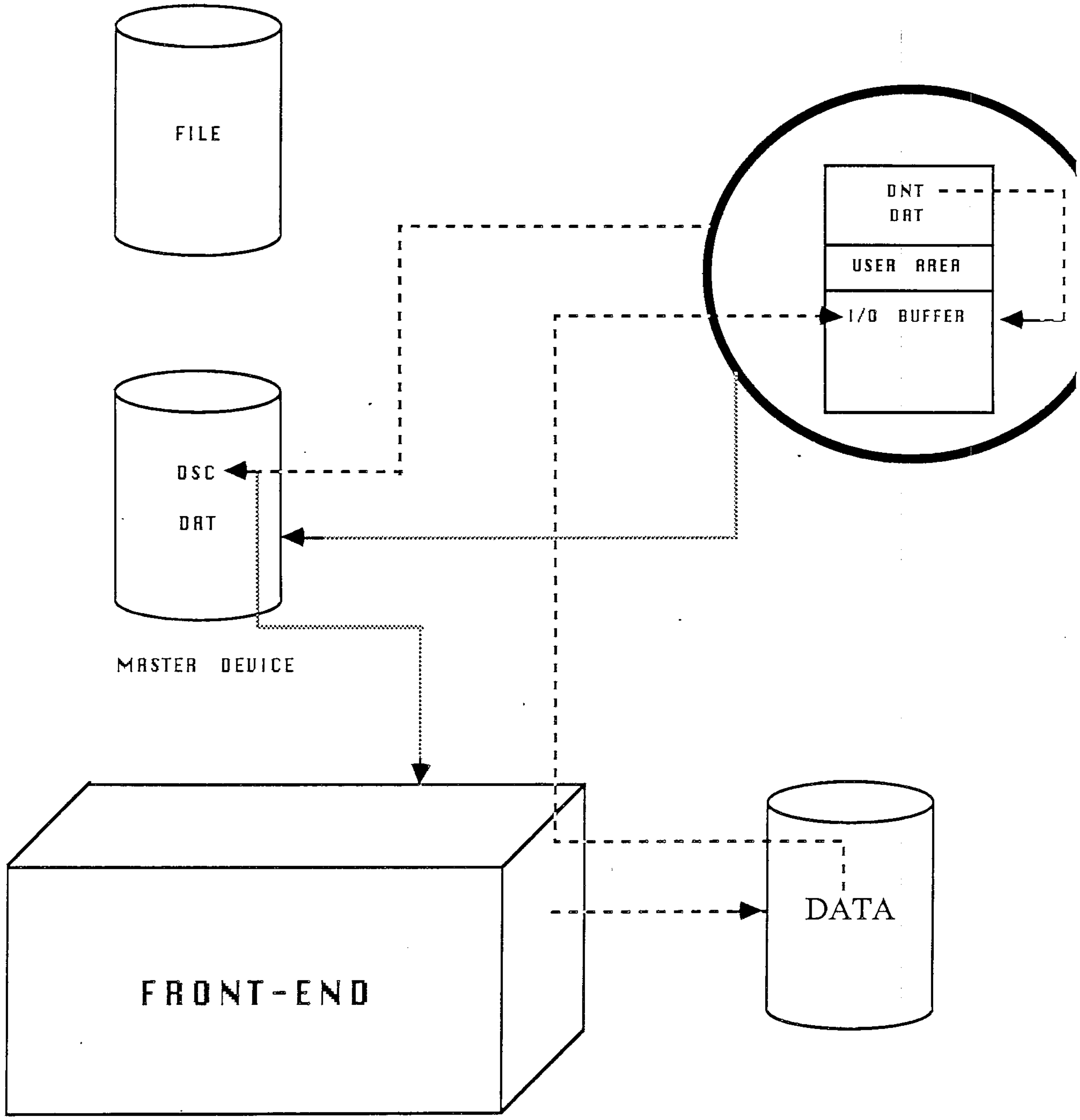
Example:	Comments:

`ACQUIRE,DN=JCLTST,ID=TNG,MF=M6,^`
`          TID=':LD2:TNG'.`
	A copy of dataset JCLTST with an ID of TNG is "acquired" from front-end M6. The TID is the destination terminal.

`ACQUIRE,DN=JCLTST,MF=AP,^`
`          TEXT='DSN=____', PAM=N.`
	A copy of dataset JCLTST is "acquired" from front-end AP. The text field is code for the front-end and the public access mode is NO PUBLIC ACCESS.

ACQUIRE

ACQUIRE,DN=DATA,MF=M4,TEXT='DSN=   '.



DISPOSE Statement

Directs a dataset to the CRAY output queue or may be used to alter dataset disposition characteristics.

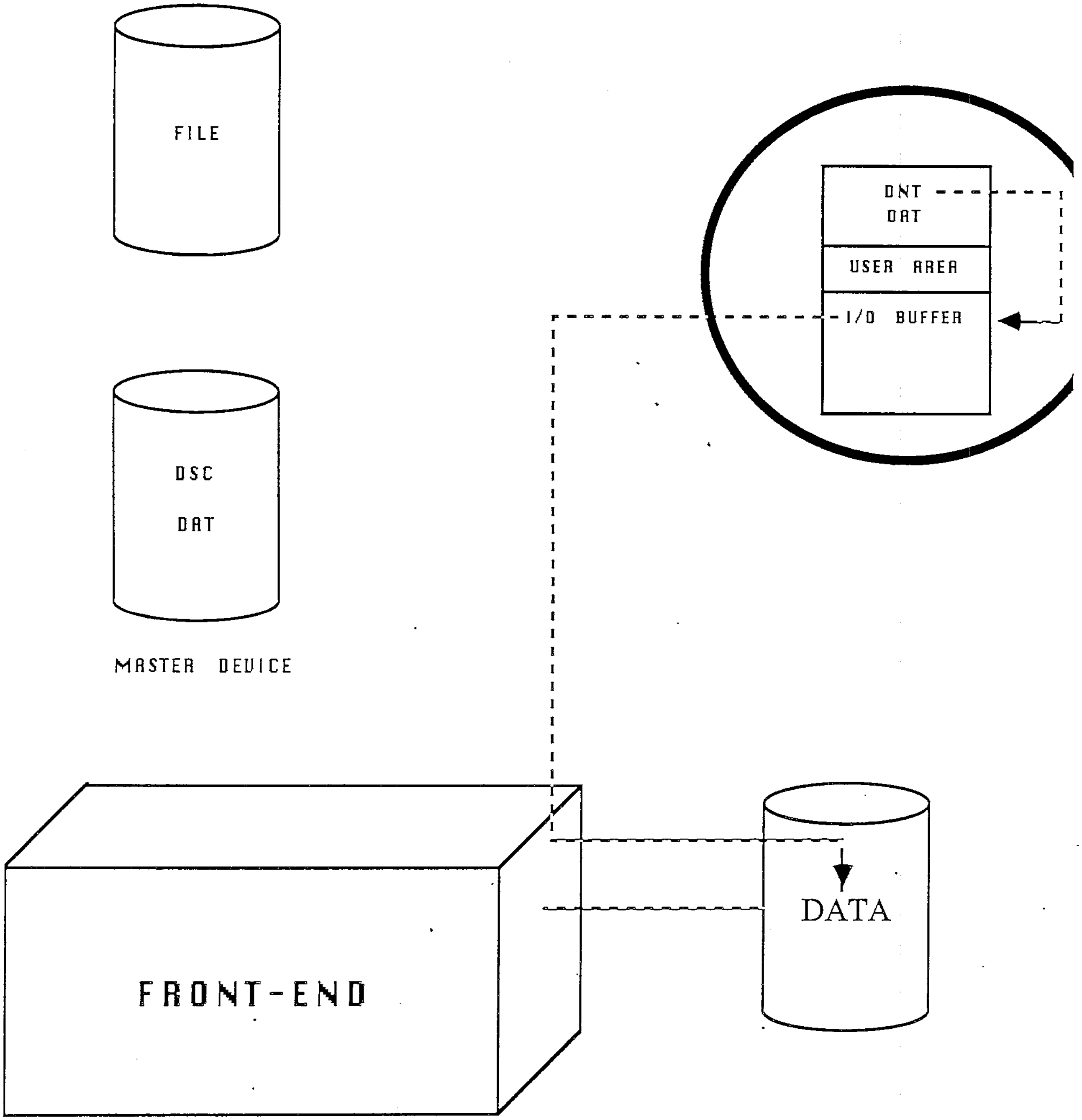
```
DISPOSE,DN=dn,SDN=sdn,DC=dc,DF=df,MF=mf,SF=sf,
          ID=uid,TID=tid,ED=ed,RT=rt,R=rd,W=wt,M=mn,
          TEXT=text,WAIT,NOWAIT,DEFER,NRLS.
```

The only required parameter is the dataset name of 1-7 alphanumeric characters.

Examples:	Comments:
DISPOSE,DN=SWCE1,MF=AP,DC=PR,SF=A.	Outputs dataset SWCE1 to front-end AP's printer. SF is a special form code used by the front-end.
DISPOSE,DN=SWCE1,MF=AP,DC=MT,^ TEXT=DSD:0:NR.	Ouputs dataset SWCE1 to mag tape on front-end AP. The text field is code for the front-end.

DISPOSE

DISPOSE,DN=LOCAL,DC=ST,MF=M4,TEXT='DSN '.



SUBMIT Statement

Directs a dataset to the CRAY input queue as a job. The first file must be a JCL file.

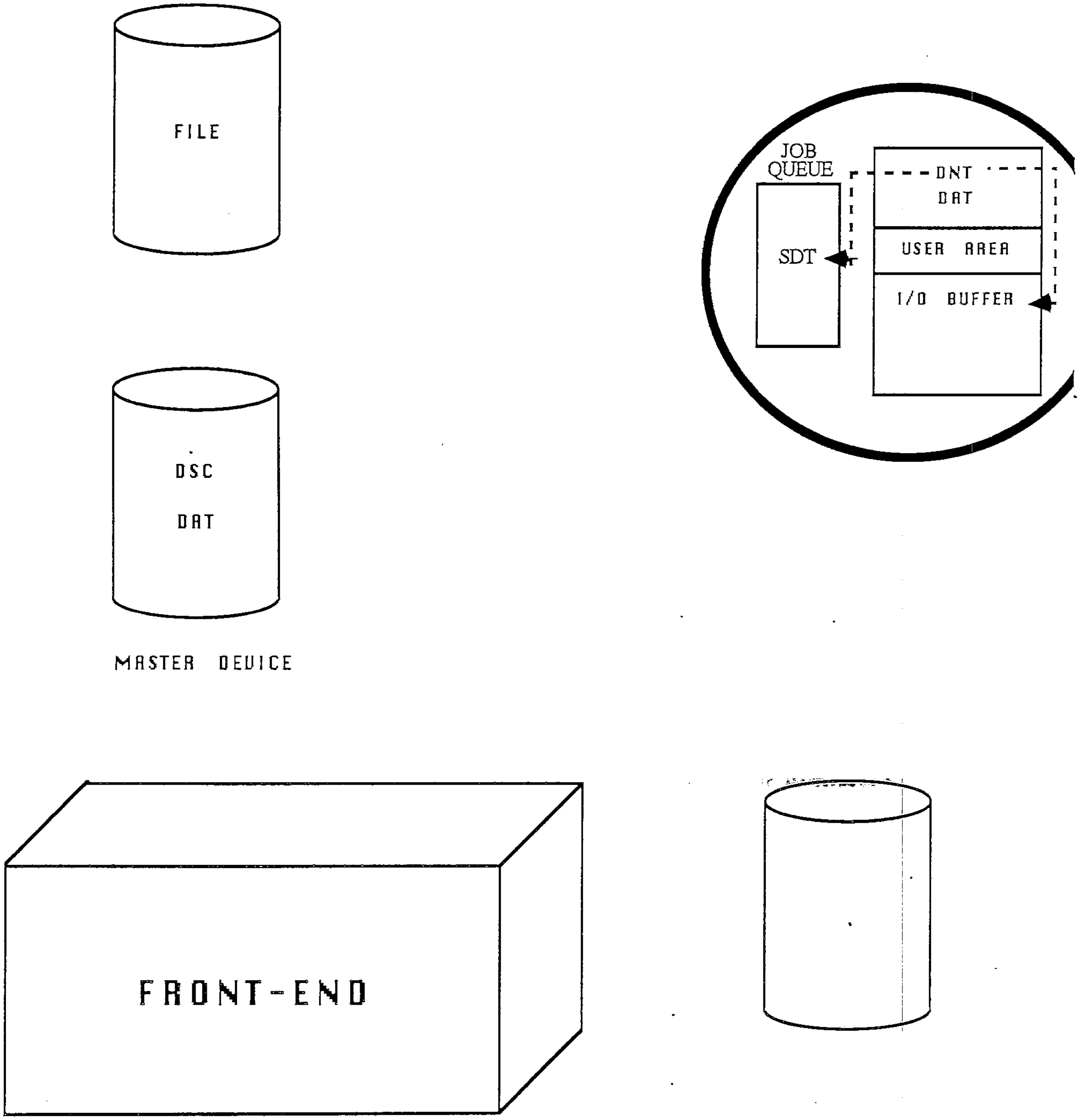
```
SUBMIT,DN=dn,SID=mf,DID=mf,TID=tid,DEFER,NRLS.
```

The only required parameter is the dataset name of 1-7 alphanumeric characters.

Examples:	Comment:
SUBMIT,DN=TLOAD,NRLS.	Directs job TLOAD to the CRAY job queue. It will remain local to this job after submission (NRLS).

SUBMIT

SUBMIT,DN=MINE,NRLS.



FILE

DSC

DAT

MASTER DEVICE

JOB QUEUE

SDT

DNT

DAT

USER AREA

I/O BUFFER

FRONT-END

CONTROL STATEMENT BLOCKS

Control statements that are grouped in a file comprise a control statement block.

Control statement blocks provide:

Conditional control statement processing.

A sequence of control statements is processed only if the specified condition is met.

IF defines the beginning of the sequence block.

ENDIF defines the ending of the sequence block.

ELSE is used to define an alternate condition.

ELSEIF defines an alternate condition to test.

Iterative statement processing.

A sequence of control statements is processed repetitively until the specified condition is met.

LOOP defines the beginning of the iterative block.

ENDLOOP defines the end of the iterative block.

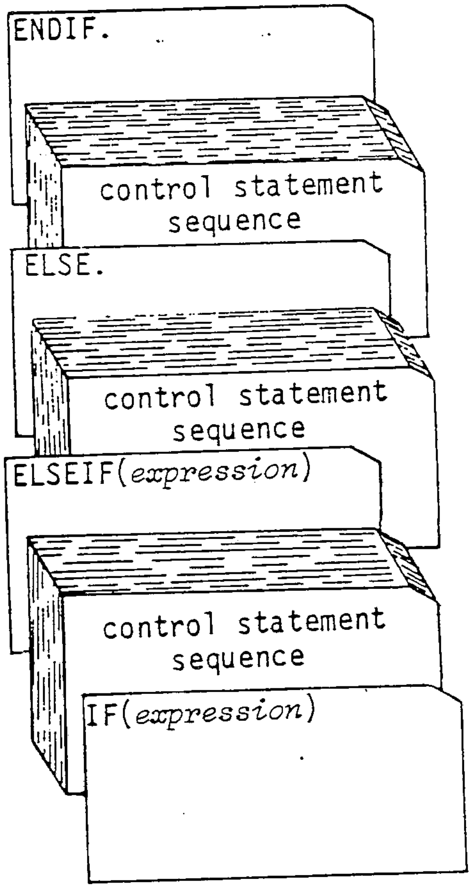
EXITLOOP defines the conditions under which the iterative block is to end.

Procedure definitions (PROCs).

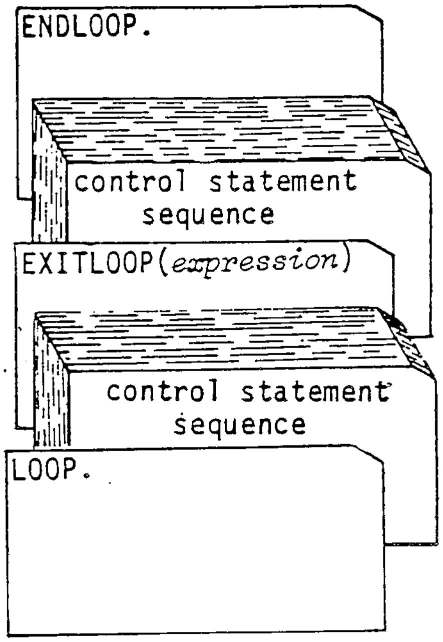
A sequence of control statements and/or data that has been saved for processing at a later time.

A subprogram or subroutine of control statement blocks.

Parameter substitution and passing is available.


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

Conditional block structure including ELSEIF and ELSE



Iterative block structure

PROCEDURES

A series of control statements in a library called for processing at a later time.

A simple PROC consists only of control statements and must be invoked through the use of the CALL statement.

A well-defined PROC consists of a prototype definition statement, control statement body, and optional data.

WELL-DEFINED PROCS

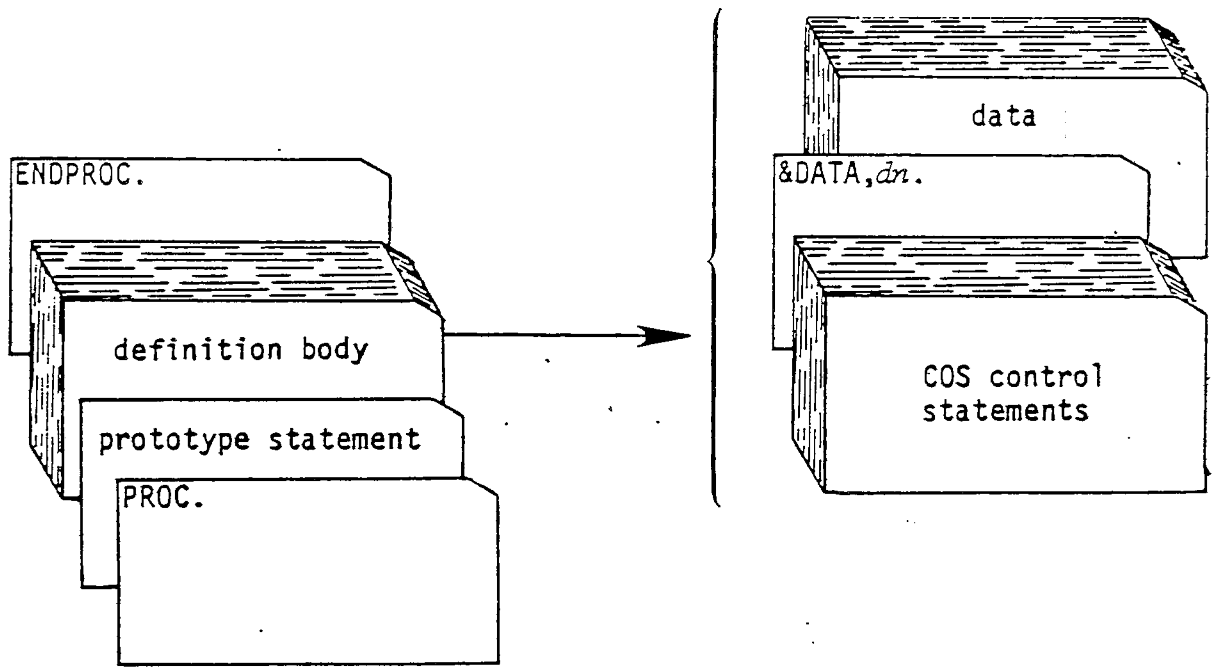
Provides the capability of replacing values within the procedure body by values supplied from the procedure call (Invocation).

A **PROC** control statement is the first statement in an in-line procedure.

A prototype statement is the next statement in an in-line procedure.

A definition body and optional data are the next statements in an in-line procedure.

An **ENDPROC** statement is the final statement in an in-line procedure.



Procedure definition deck structure

PROCEDURE DEFINITION

The **PROC** statement defines the beginning of an in-line procedure definition block.

PROC.

The 'prototype' statement is next and specifies the name of the PROC and parameter specifications.

NAME,p_1,p_2,.....,p_n.

NAME is the name of the PROC and can be 1-8 alphanumeric characters long. P_1...p_n are the parameter specifications. Either positional or keyword specifications are allowed. (See SR-0011).

The 'definition' body is next and consists of a series of CRAY control statements and optional substitution parameters.

The **ENDPROC** statement is the final statement and indicates the end of an in-line procedure.

ENDPROC.

PROCEDURE EXAMPLE

EXAMPLE 1.

This PROC will delete a permanent dataset.

```
1. PROC.
2. ERASE,DSX,ED=:,ID=DON1:DON2.
3. ACCESS,DN=MYDON,PDN=&DSX,ED=&ED,ID=&ID,UQ.
4. DELETE,DN=MYDON.
5. RELEASE,DN=MYDON.
6. ENDPROC.
```

<u>Line 1</u>

- Defines the beginning of an in-line procedure definition block.

<u>Line 2</u>

- Is the prototype statement which specifies 'ERASE' as the PROC name.

- Also defines three parameters, two of which are in keyword.

Format:

DSX - Parameter must be supplied by the user when the PROC is invoked. A required positional parameter.

ED=: - Provides no default values, but allows the user to specify a value.

ID=DON1:DON2
- Provides DON1 as the default value if 'ID' is omitted from the calling statement (DVALUE).
- Provides DON2 as the default value if 'ID' is present without a value (KVALUE).

<u>Line 3</u>

- Is a part of the definition body.

- DN=MYDON is a parameter which is used in the access control statement.

- PDN=&DSX is a substitution parameter that a user is <u>required</u> to supply.

- ED=&ED is a substitution parameter that a user <u>may</u> supply.

- ID=&ID is a substitution parameter that a user <u>may</u> supply.

- UQ opens the dataset exclusively.

<u>Line 4,5</u> - Cray control statements to delete and then release the local dataset MYDON.

<u>Line 6</u> - Indicates the end of an in-line procedure.

MAGNETIC TAPE DATASETS

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

A magnetic tape can be unlabeled (NL), ANSI standard labeled (AL), or IBM standard lableled (SL) and can be recorded or read at either 1600 or 6250 bits per inch (bpi).

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

The COS tape system uses Buffer Memory (in the IOS subsystem) as a tape blocked buffering area so that the job's I/O buffer need not be as large as the tape block. This technique can result in significant memory savings whenever large tape blocks are being processed and in increased transfer rates whenever smaller blocks are being processed. The advantage in having a large COS buffer is a reduction in the overhead in the tape subsystem.

With Release 1.13 positioning support for tape datasets is possible. Users can position a tape dataset at any block on any volume, obtain the current position information for a tape dataset, and enable recovery of tape jobs after a system interruption.

Also, a MOD parameter has been added to the ACCESS control statement for use with on-line tapes. When MOD is specified on an access of a tape dataset, any data written to the datset is appended to the data already contained in the dataset rather than being written from the beginning of the dataset.

A tape dataset is created by an ACCESS statement with the NEW parameter.

The ASSIGN statement can be used to create the dataset characteristics such as buffer size and precedes the ACCESS,NEW,DN=_______ statements.

TAPE JCL EXAMPLE

```
JOB,JN=EXAMPLE,*6250=2.
ACCOUNT,AC=account#.
ACCESS,DN=INTAPE,DT=*6250,VOL=1000:1001.
ASSIGN,DN=INTAPE,A=FT20.
ACCESS,DN=OUTTAPE,PDN=EXAMPLETAPE,LB=SL,DF=IC,CS=SL,DT=*6250,^
          XDT=83365,VOL=2000:2001,NEW.
ASSIGN,DN=OUTTAPE,A=FT21.
CFT.
LDR.
/EOF

        PROGRAM EXAMPLE
                .
                .
                .
        READ (20,xx) . . .
                .
                .
                .
        WRITE (21,xx) . . .
                .
                .
                .
/EOF
```

In this example job a FORTRAN program reads a magnetic tape dataset on unit 20 and writes a magnetic tape dataset on unit 21.

The input tape has the following characteristics:

Non-lableled
Transparent format
ASCII character set
6250 bpi
Volume identifiers: 1000 and 1001
COS blocked format

The output tape has the following characteristics:

IBM standard label
Permanent dataset name = EXAMPLETAPE
Interchange format
EBCDIC character set
6250 bpi
Expiration date = 83365
Volume identifiers: 1000 and 2001
This dataset is to be created (by use of the NEW parameter)

The permanent dataset name corresponds to the file identifier in the tape label.

JCL QUIZ

1. What statement does every job require?

2. What dataset format will COPYD statements process?

3. What is dataset staging?

4. What statement tells you what datasets exist on mass storage?

5. What statement is used to deallocate a local dataset?

6. What is a permanent dataset?

7. What JCL statements are needed to delete a permanent dataset?

8. How is a text dataset represented in COS?

9. What is the first file of every job submitted to a Cray?

10. When would you use PDSLOAD?

Cray's Text Editor- TEDI

3

MODULE OBJECTIVES

Upon completion of the TEDI and Interactive module, and with the aid of all furnished reference material, the learner should be able to:

1. Use an Interactive Station
2. Create a CAL program with TEDI
3. Execute a CAL program interactively
4. Modify the source code with TEDI
5. Save, access, and modify a text dataset with TEDI

USING TEDI

This section explains the basic usage of TEDI. TEDI is an interactive line editor on a Cray computer system operating under control of COS. TEDI can be used to edit computer programs, data, documentation, or any other text files.

TEDI's most commonly used commands are demonstrated through the creation and modification of a CAL program. Read through this section once first to get the big picture of how TEDI is used, then complete it step by step.

The first step is to log onto the terminal. This section describes use of an AMDAHL as the interactive station. The user IDs and passwords are those available to the Software Training Department at Mendota Heights and will be different thatn those used at your site. If this module is used within Mendota Heights the IDs and passwords will be the same as used in this module. Keep in mind that you will probably use TEDI on the IOP station at your site.

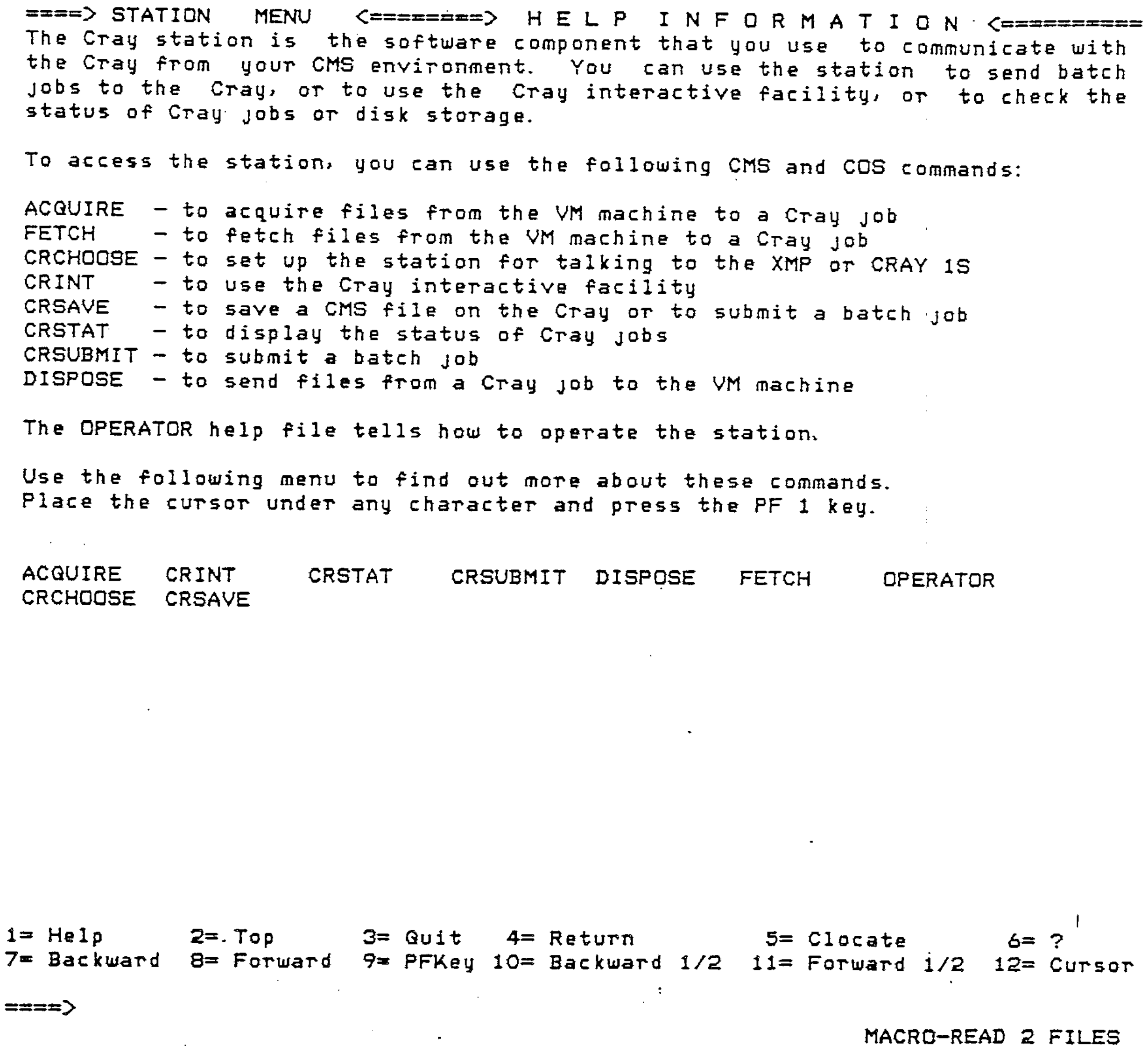
To begin, make sure you have the AMDAHL logon screen on your terminal, then depress the "ENTER" key on the console. This key will be denoted by a ↲ throughout this module. The console displays 'CP read' in the lower right corner of the screen.

Response	L TNGxx ↲	Where xx is your training ID.
Display	ENTER PASSWORD:	
Response	TNGxx ↲	Your training password.
Display	LOGMSG	(see example below)
Response	CRINT ↲ (IAC for IOP Station) (IAS for DB AOS Station)	This gets you the interactive station.
Display	ENTER "/LOGON" ↲	Similar to the JOB statement.
Response	/LOGON ↲	

This last response should get you the interactive station under directory TNG.

```
L U1502
ENTER PASSWORD:

LOGMSG - 10:29:57 CDT TUESDAY 08/28/84
*=======================================*
* FOR COMPLETE LOG MESSAGE TYPE:  Q LOG *
*                                       *
* LAST LOG UPDATE:VM STATION UNAVAILABLE *
*                                       *
* LAST UPDATE TIME: 08/28/84 10:30      *
*                                       *
* ENTER "CRAYNEWS" FOR CRAY INFORMATION. *
*=======================================*
LOGON AT 11:41:46 CDT TUESDAY 08/28/84
VM/SP REL3  03/27/84
CRINT
U (19C) R/O
R; T=0.50/0.57 11:41:53
ENTER "/LOGON"
/LOGON
```

AMDAHL VM STATION COMMANDS



```
====> STATION   MENU   <========> H E L P  I N F O R M A T I O N <==========
The Cray station is  the software component that you use  to communicate with
the Cray from  your CMS environment.  You  can use the station  to send batch
jobs to the  Cray, or to use the  Cray interactive facility, or  to check the
status of Cray jobs or disk storage.

To access the station, you can use the following CMS and COS commands:

ACQUIRE  - to acquire files from the VM machine to a Cray job
FETCH    - to fetch files from the VM machine to a Cray job
CRCHOOSE - to set up the station for talking to the XMP or CRAY 1S
CRINT    - to use the Cray interactive facility
CRSAVE   - to save a CMS file on the Cray or to submit a batch job
CRSTAT   - to display the status of Cray jobs
CRSUBMIT - to submit a batch job
DISPOSE  - to send files from a Cray job to the VM machine

The OPERATOR help file tells how to operate the station.

Use the following menu to find out more about these commands.
Place the cursor under any character and press the PF 1 key.


ACQUIRE   CRINT     CRSTAT    CRSUBMIT DISPOSE   FETCH     OPERATOR
CRCHOOSE  CRSAVE









1= Help       2= Top       3= Quit    4= Return          5= Clocate         6= ?
7= Backward   8= Forward   9= PFKey  10= Backward 1/2   11= Forward 1/2   12= Cursor

====>
                                                               MACRO-READ 2 FILES
```

Once in interactive mode the screen displays the interactive prompt '!'.

```
               !

Response       ACCOUNT,AC=account#,US=TNG,UPW=TNG. ↲

               !

Response       TEDI,DN=TEDI1. ↲          TEDI1 is the dataset you are going to create and
                                         edit under the TEDI utility.
```

A similar message to the following should appear on your screen followed by the TEDI prompt '*'.

```
               TE017    NEW DATASET.
               TEDI1     0 LINES.
               *
```

You are finally executing the TEDI utility, so now you can type in a program.

The following exercise command steps will create a CAL program to square a value in memory location 'NUMBER' and store the result in memory location 'ANSWER'. As you use TEDI commands refer to SG-0055, Section 4, for further information. Key in the example.

Do not use the tab key; two spaces are a TEDI tab character.

CREATING A CAL PROGRAM

EXERCISE 1

Step 1.

```
Display     *
            STS  10  20  35                 Double spacing will now tab.
```

Step 2.

```
Display     *

Command     AL ↲                      Adds lines to a dataset (TEDI1).
            &                         The & prompts for the line insertion.

Responses   &          IDENT          SQUARE ↲
            &          START          HERE ↲
            &HERE      =              * ↲
            &          A2             NUMBER,0     GET ↲
            &          A3             NUMBER,0     OPERANDS ↲
            &          A1             A2*A3        SQUARE # ↲
            &          ANSWER,0       A1 ↲
            &          ENDP ↲
            &NUMBER    CON            5 ↲
            &ANSWER    BSS            1 ↲
            &          END ↲
            &. ↲                   The period terminates the AL command and will
                                   display the contents of TEDI with line numbers.
```

Step 3.

```
Display     *

Command     W ↲                        Writes your program to dataset TEDI1.

Display     TEDI1        12 LINES      The number of lines in dataset TEDI1.

            *
```

You now have a local dataset named TEDI1 that you may SAVE, ASSEMBLE, etc. Remember you are in interactive mode, so many possibilities for this dataset exist.

In order to manipulate the dataset TEDI1 further you may exit the TEDI utility. Remember, you have the * prompt on the screen.

Step 4.

Respond	END ↲	To terminate TEDI and update the dataset if not updated previously. At this point you already have updated TEDI1 through the W command.

You should now be back into the interactive station. A '!' prompt should appear on the screen. It takes concentration to remember what is actually taking place on the screen. The easiest method is to remember the prompts (* for TEDI and ! for interactive station).

Your interactive station looks like datasets $IN and $OUT to COS so knowledge of JCL and COS is imperative when submitting further commands.

To SAVE dataset TEDI1 as a permanent dataset type:

```
!SAVE,DN=TEDI1,ID=TNGxx. ↲
```

To ASSEMBLE dataset TEDI1 type:

```
!CAL,I=TEDI1. ↲
```

After ASSEMBLY, to EXECUTE TEDI1 type:

```
!LDR. ↲
!RELEASE,DN=TEDI1. ↲
!ACCESS,DN=TEDI1,ID=TNGxx,UQ. ↲
```

You are now ready for Exercise 2 where you will change lines of source code.

MODIFYING TEDI

EXERCISE 2

STEP 1.

Respond	!TEDI,DN=TEDI1. ↲	
Display	*	Now in TEDI.
Command	BL1 ↲	To add comments before line 1.
	&	
Insert	*THIS PROGRAM SQUARES A NUMBER ↲	
Display	&	
	. ↲	The period terminates the BL command.

STEP 2.

Display	*	
	T* ↲	View TEDI1 with new line numbers.
Command	RL6 ↲	You will replace line 6.
	& A4 Number,0	Replacement line. ↲
	&. ↲	Terminates the RL command.
	*	
Command	X7 ↲	Allows you to change parts of a line.
	:	You must now space the cursor in line 7 (A1 A2*A3) to beneath the 3 in A3. Once you have done this type:
	4 ↲	
	*	
	W ↲	Rewrites TEDI with all changes included. NOTE: You may use the W wherever you deem necessary, <u>but</u> the rewrite should be done periodically during an editing session to prevent loss of current changes due to system failure. If TEDI1 has been made a <u>permanent</u> dataset on the Cray you will be asked permission to update the dataset. That is, if you have accessed it UQ.
	*	
	T* ↲	View TEDI1 with the new changes.

Interactive mode enables the terminal to both send and receive from the Cray as datasets in $IN and $OUT. Each interactive command ($IN) is sent to the Cray and executed. A response will appear on terminal ($OUT) as if it were the message in the user's $LOG. As an example, suppose you wanted to ASSEMBLE and EXECUTE dataset TEDI1 which is a permanent dataset on the Cray. Type in:

!ACCESS,DN=TEDI1. ↵	Accessing TEDI1.
!CAL,DN=TEDI1. ↵	Assembling TEDI1.
!LDR. ↵	Executing TEDI1.

The next three JCL statements transmit a dump of TEDI1 back to Software Training's printer.

!DUMPJOB. ↵	Dumps entire TEDI1 job.
!DUMP,O=FREND. ↵	Dumps words 0-200_8 to FREND.
!DISPOSE,DN=FREND,MF=V3,DEFER,DC=PR,TID=RSCS,TEXT='TAG=TNGA'. ↵	Prints FREND on Software Training printer.

ADDING COS SYSTEM FUNCTIONS

EXERCISE 3

We will assume TEDI1 is still in the form it was after Exercise 2. If your TEDI1 is not, modify the program to alter it back to this form.

We will add the JCL statements necessary to create a job dataset suitable for the Cray job input queue.

Step 1.

*	TEDI prompt reminder.
BL ↲	You will add JCL statements to the program.
&JOB,JN=TNGxx. ↲	Job card.
&ACCOUNT,AC=account#,US=TNG,UPW=TNG. ↲	
&CAL. ↲	To assemble TEDI1.
&LDR. ↲	Execute program.
&EXIT. ↲	
&DUMP,O=FREND. ↲	
&DISPOSE,DN=FREND,MF=V3,DEFER,DC=PR,TID=RSCS,TEXT='TAG=TNGA'. ↲	
&EXIT. ↲	
&. ↲	Terminates the BL command.
*	
T* ↲	Examine TEDI1 to ensure completeness.

Step 2.

*END ↲	Command to update TEDI1.
!	Back to interactive mode.
SUBMIT,DN=TEDI1. ↲	Submit job to Cray.
!	Back to interactive mode.
/LOGOFF Q ↲	To leave interactive and return to the station operating system.
LOGOFF ↲	To log off the Amdahl.

TEDI QUIZ

1. What character indicates you are in interactive mode on the Cray?

2. What character prompts for line insertion in TEDI?

3. What character indicates TEDI is awaiting a command?

4. What command terminates TEDI and updates your dataset?

5. What is the command to insert a line before line 6 in a TEDI dataset?

6. What character terminates a TEDI command?

7. What parameter must be included on the ACCESS statement in order to make changes to the accessed dataset?

8. What command is issued to ask to be put on the Cray interactively?

9. What is an advantage to using TEDI?

10. What is a disadvantage to using TEDI?

TEDI QUIZ

1. What character indicates you are in interactive mode on the Cray?

2. What character prompts for line insertion in TEDI?

3. What character indicates TEDI is awaiting a command?

4. What command terminates TEDI and updates your dataset?

5. What is the command to insert a line before line 6 in a TEDI dataset?

6. What character terminates a TEDI command?

7. What parameter must be included on the ACCESS statement in order to make changes to the accessed dataset?

8. What command is issued to ask to be put on the Cray interactively?

9. What is an advantage to using TEDI?

10. What is a disadvantage to using TEDI?

Cray Assembly Language

4

MODULE OBJECTIVES

Upon completion of the Cray Assembler Language module, and with the aid of all furnished reference material, the learner should be able to:

1. Read simple CAL programs
2. Write a CAL program
3. Assemble a CAL program
4. Debug a CAL program
5. Create a binary load module
6. Run an object program from a library
7. Explain the difference between relative and absolute binary datasets
8. Read a loader map
9. Create an executable binary dataset
10. Run an executable binary dataset

A POWERFUL ASSEMBLER

CAL is a powerful translator with high level language features.

```
CAL,CPU=type,I=idn,L=ldn,B=bdn,E=edn,ABORT,DEBUG,
    LIST=name,S=sdn,T=bst,X=xdn.
```

CAL source statements are:

Symbolic machine instructions or **pseudo instructions**

Symbolic Machine Instructions:

Represent functions of Cray CPU architecture
Translate one for one
- one symbolic machine instruction translates to one binary machine instruction

Pseudo Instructions:

Allow programmer control of assembly process
Generally do not generate code
Provide features which include:
- control of the content of the assembler listing
- data defined and loaded with program
- source code and data can be assigned to specific areas in memory

CAL source statements can also include:

Macro code:

A sequence of code defined in the source program and assembled in the object program when the assembler calls it

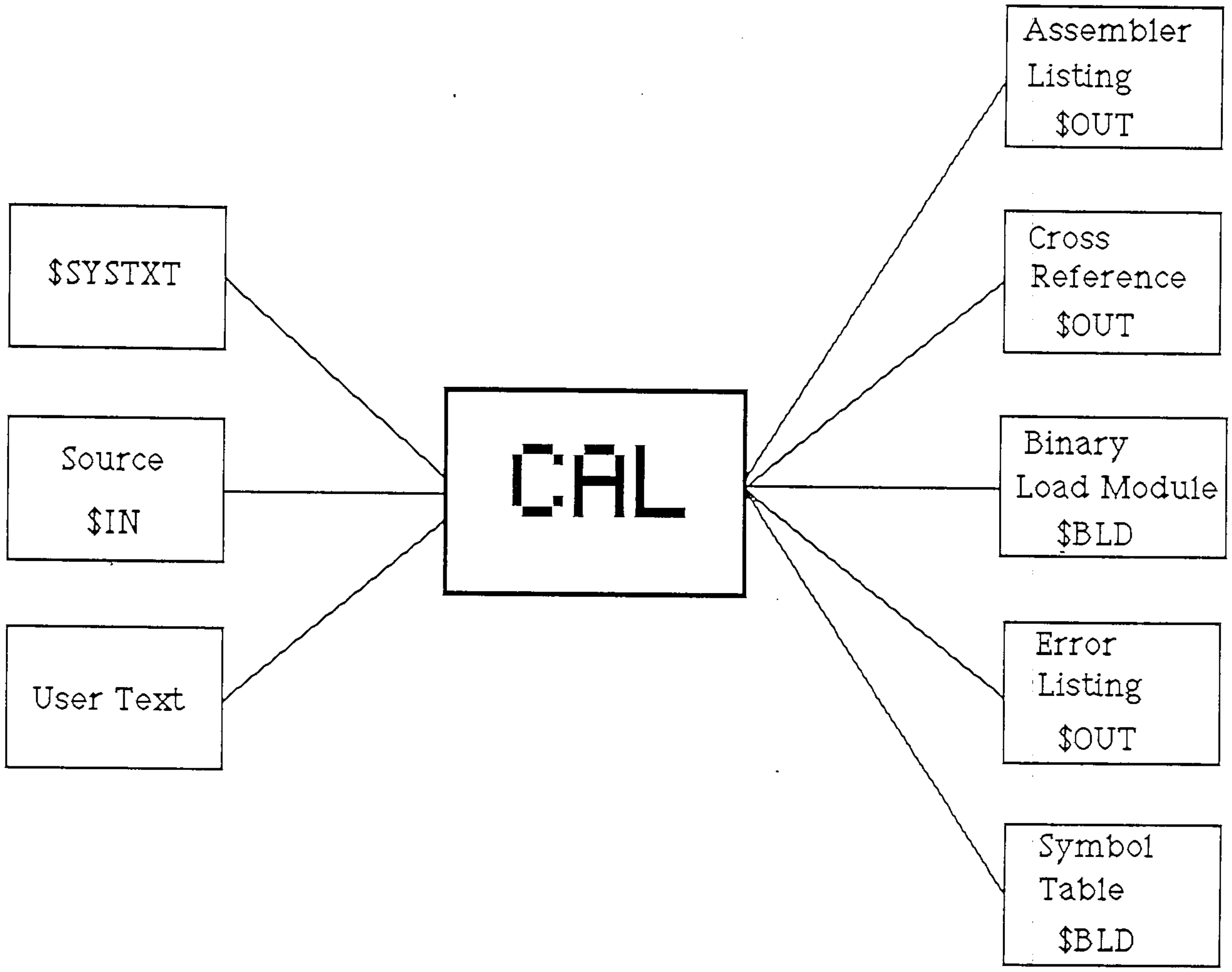
The assembler may produce many binary machine instructions to complete a macro operation.

Opdef code:

Recognized by syntax pattern and uses nonverb-structured syntax

System text:

Defines macros and opdefs to the assembler


$SYSTXT
Source
$IN
User Text
CAL
Assembler
Listing
$OUT
Cross
Reference
$OUT
Binary
Load Module
$BLD
Error
Listing
$OUT
Symbol
Table
$BLD

SOURCE LINE FIELDS

Location field:

Must begin in columns 1 or 2.

No entry is assumed if columns 1 and 2 are blank.

Terminated by a blank.

Optionally contains up to an 8-character symbol which must begin with (A-Z,$,%,@).

Result field:

First non-blank character following the location field.

Terminated by a blank.

Must begin in columns 3-34.

No entry is assumed if columns 3-34 are all blanks.

Operand field:

First non-blank character following the result field.

Terminated by a blank.

Must begin at column 34 or before.

If the result field extends beyond column 32, the operand field must follow one blank separator.

Comment field:

First non-blank character following the operand field.

If the operand field is empty then it must start at column 35 or beyond.

NOTE: An asterisk (*) is column 1 indicates to the assembler that what follows in that line will be a comment.

```
  15534c  031550                     A5          A5-1                                      LC5921XA.207
       d  030446                     A4          A4+A6                                     LC5921XA.208
  15535a  011 00015532b              JAN         BOOT40          If more entries to process LC5921XA.209
                                *                                                          N081130F.276
                                *    Start all defined CPUs if able to.                    N081130F.277
                                *                                                          N081130F.278
       c  0401 00000001              S1          C@CPQUAN        Number of CPUs assembled  N081130F.279
  15536a  0402 00000003              S2          C@CPTYPE        Mainframe type            N081130F.280
       c  <macro>                    $IF         (S1,GT,S7=1),AND,(S2,EQ,S7=@CRAYXMP)      N081130F.281
  15540d  022201                       A2          1             Initial CPU to start      N081130F.282
                                *                                                          N081130F.283
                                *      Loop attempting to start CPUs 1 through C@CPQUAN-1. N081130F.284
                                *      If any processor does not start, assume the rest    N081130F.285
                                *      are not available.                                  N081130F.286
                                *                                                          N081130F.287
  15541a  <macro>                      $LOOP       A2,LT,A7=C@CPQUAN                       N081130F.288
                                *                                                          N081130F.289
                                *        Build an initial exchange package for this CPU.   N081130F.290
                                *                                                          N081130F.291
  15542a  022600                         A6          INITXP                                N081130F.292
       b  022720                         A7          LE@XP                                 N081130F.293
       c  007 00026351b                  R           CLEAR       Clear the exchange package area N081130F.294
  15543a  022600                         A6          INITXP                                N081130F.295
       b  0401 0015741a                  S1          STCPU       Start address             N081130F.296
       d  042277                         S2          1           Mode to monitor mode      N081130F.297
                                         ERRIF       S@XPMM,NE,S@XPM+N@XPM-1               N081130F.298
  15544a  1203 00002040                  S3          XLA,0       Limit address             N081130F.299
       c  007 00026613a                  R           SETXP       Set up the exchange package N081130F.300
  15545a  071302                         S3          A2                                    N081130F.301
       b  1303 00000013                  W@XPS3+INITXP,0  S3     Processor number being started N081130F.302
                                *                                                          N081130F.303
                                *        Indicate the request to start in the CPUs PWS entry. N081130F.304
                                *                                                          N081130F.305
       d  0207 00003620                  A7          B@PWS+LH@PWS  FWA + Header length     N081130F.306
  15546b  0206 00000120                  A6          LE@PWS      Entry length              N081130F.307
       d  032626                         A6          A2*A6       Entry number              N081130F.308
  15547a  030667                         A6          A6+A7       Address of PWS for this CPU N081130F.309
       b  <opdef>                        PUT,1       S6&S7,PWINIT,A6  Set request-to-start flag N081130F.310
  15551a  <macro>                        SETIP       PN=A2,SCR=A3,ERROR=$STOP084           N081130F.311
                                *                                                          N081130F.312
                                *        Check to see if this one started.                 N081130F.313
                                *                                                          N081130F.314
  15553d  0203 00001750                  A3          D'1000      Arbitrary number of tries N081130F.315
                                *                                before declaring a CPU dead. N081130F.316
  15554b  <macro>                        $LOOP       A3,GT,A7=0                            N081130F.317
  15555b  <opdef>                          GET,S1      S6&S7,PWEXEC,A6  Get CPU-started-execution flagN081130F.318
  15556b  031330                           A3          A3-1                                N081130F.319
       c  <macro>                          $EXITLP     S1,NONZERO  If CPU did start        N081130F.320
  15557b  <macro>                        $ENDLOOP                                          N081130F.321
       d  <macro>                        $EXITLP     S1,ZERO     If this CPU did not start N081130F.322
  15560c  030220                         A2          A2+1        Next CPU number           N081130F.323
       d  <macro>                      $ENDLOOP                                            N081130F.324
  15561b  <macro>                    $ENDIF                                                N081130F.325
                                *                                                          LC5922GE.41
                                *    Set up System Task Table (STT) for task 0.            LC5922GE.42
                                *                                                          LC5922GE.43
       b  0206 00003762              A6          B@STT           STT header address        LC5922GE.44
```

NAMES AND SYMBOLS

Names and symbols used in a CAL program module look alike. They have the same syntax rules, but are used differently.

Syntax:

One to eight characters.
Letters must be caps and no hidden characters are allowed in the spaces.
Characters other than the first may be 0-9.

Names:

Identify:
- Program modules.
- Blocks.
- Sequences of pseudo instructions.

Do not have values or attributes.
Do not conflict with each other in different contexts.

Symbols:

Are used in symbolic machine instructions such as:
- Jump addressing.
- Memory addressing.
- Expression (EXP) evaluation.

Are used in pseudo instructions:
- Symbol definitions.

Have values and attributes.
Must be unique.

SYMBOL ATTRIBUTES

Word address - 22 bit value.
Parcel address - 24 bit value (upper 22 bits and word address).
Value - 64 bit value.

Relocatable - symbol addresses in a relocatable assembly.
EXT - symbols defined by EXT pseudo .
Absolute - symbols in an absolute assembly.

Common - symbols defined in a common block.
Redefinable - symbols defined by certain pseudos which may be defined more than once in a program module.

```
JOB,JN=U1502A.
ACCOUNT,AC=265124,US=TNG,UPW=TNG.
*************************************************************************
*
*     THIS JOB DEMONSTRATES THE DIFFERENCE BETWEEN NAMES AND SYMBOLS
*
*
*************************************************************************
CAL.
LDR,MAP=ON.
/EOF
           IDENT     NAME
           START     HERE
*
ZERO       =         0
*
HERE       =         *
*
           A1        ZERO           INDEX
           A2        10             MAX LOOP
           A4        0              ACCUMULATOR
*
LOOP       A3        ADD,A1         LOAD
           A1        A1+1           INCREMENT TABLE INDEX
           A0        A2-A1          CHECK FOR COMPLETION
           A4        A3+A4          ACCUMULATIVE SUM
           JAN       LOOP
*
           RESULT,0  A4             STORE
           SNAP      (A4)
*
           ENDP
*
ADD        CON       1,2,3,4,5,6,7,8,9,10
*
RESULT     BSS       1
*
           END
/EOF
```

PROGRAM CONTROL

Defines the limits of a program module.

Defines the type of assembly.

IDENT - Required; marks the beginning of a program module.

END - Required; marks the end of a program module.

ENDP - Required; marks the end of a program.

ABS - Designates absolute rather than relocatable assembly.

COMMENT - Enters a comment, generally a copyright, in the program descriptor table.

BASE - Declares the base for numeric data; diagnostics use M.

SYMBOL DEFINITION

= - Equates a symbol to a value; not redefinable.

SET - Sets a symbol to a value; redefinable.

MICSIZE - Equates a symbol to the value of the number of characters in a micro string.

DATA DEFINITION

The following pseudos allow preloaded data to be designated as integer floating point or character notation. They are the only pseudos generating object binary.

CON - Generates one full word of binary data; forces a word boundary.

BSSZ - Generates words of zeros.

DATA - Generates words of numeric or character data; does not force a word boundary.

BSS - Reserves words of memory.

```
                                                     IDENT     NAME
                                                     START     HERE
                                          *
                                   0      ZERO       =         0
                                          *
                                   0a+    HERE       =         *
                                          *
              0a  <opdef>                            A1        ZERO           INDEX
               b  <opdef>                            A2        10             MAX LOOP
               c  <opdef>                            A4        0              ACCUMULATOR
                                          *
               d  1013 00000010+          LOOP       A3        ADD,A1         LOAD
              1b  030110                             A1        A1+1           INCREMENT TABLE INDEX
               c  031021                             A0        A2-A1          CHECK FOR COMPLETION
               d  030434                             A4        A3+A4          ACCUMULATIVE SUM
              2a  011 00000000d+                     JAN       LOOP
                                          *
               c  1104 00000022+                     RESULT,   A4             STORE
              3a  <macro>                            SNAP      (A4)
                                          *
              6c  <macro>                            ENDP
                                          *
             10   0000000000000000000001  ADD        CON       1,2,3,4,5,6,7,8,9,10
                  0000000000000000000002
                  0000000000000000000003
                  0000000000000000000004
                  0000000000000000000005
                  0000000000000000000006
                  0000000000000000000007
                  0000000000000000000010
                  0000000000000000000011
                  0000000000000000000012
                                          *
             22                       1   RESULT     BSS       1
                                          *
                                                     END
```

CROSS REFERENCE

blank Symbol value is used at this point.

D Symbol defined at this reference; that is, it appears in the location field of an instruction or is defined by a SET, =, or EXT pseudo instruction.

E Declares the symbol as an entry name.

F Symbol used in an expression on an IFE, IFA, or ERRIF conditional pseudo instruction.

R Symbol used in an address expression in a memory read instruction or as a B or T register symbol in an instruction which reads the B or T register.

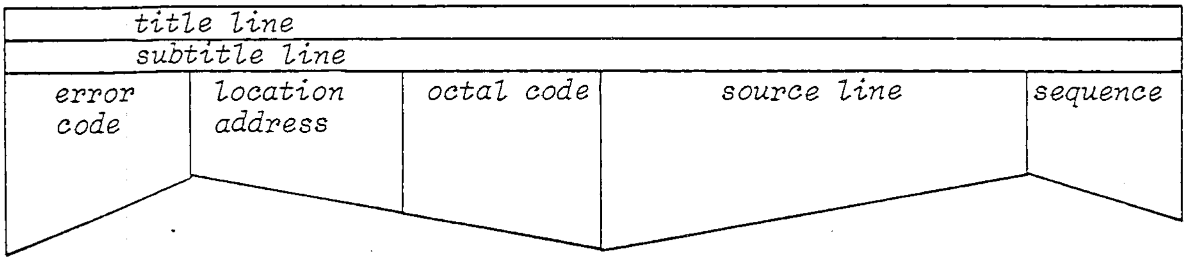
S Symbol used in an address expression in a memory store instruction or as a B or T register symbol in an instruction which stores a new value in the B or T register.

Example of page header:

1	66	76	96	105	115
title	*cpu type*	CAL *version*	*date*	*time*	Page *n*
subtitle	unused	Block:*bname*	Qualifier:*qualname*		(*n*)

SOURCE STATEMENT LISTING

The listing for the source statements of a CAL program is organized into five columns of information.



title line				
subtitle line				
error code	*location address*	*octal code*	*source line*	*sequence*

```
NAME                                                           CRAY XMP   CAL X.15(11/16/84)  11/24/84 15:35:44  Page 2
                                                                                                                   (2)

                 1357a+ $$LOAD$$                   1:19       1:19 F     1:19 D
                 1315a+ $$SAVE$$                   1:19       1:19 F     1:19 D
                    0X  $WFA                       1:19 D     1:19
                    0X  $WFF                       1:19 D     1:19
                    0X  $WFI                       1:19 D     1:19
                    0X  $WFV                       1:19 D     1:19
                    0   %$MULTI    $SYSTXT         1:19 F     1:19
                    1   %$NEWSEQ   $SYSTXT         1:19 F     1:19
                    0   %$STACK    $SYSTXT         1:19 F     1:19
                    6   %ARPTR                     1:19 D     1:19 F     1:19
                    0   %STKPTR                    1:19 D
                   10+  ADD                        1:12 R     1:23 D
                    0   F$ADV      $SYSTXT         1:21
                    0a+ HERE                       1: 2 E     1: 6 D
                    0d+ LOOP                       1:12 D     1:16
                   30   N@ARN      $SYSTXT         1:19
                    1   N@ARVAL    $SYSTXT         1:19
                   65+  QZH44HZQ                   1:19 S     1:19 R     1:19 F     1:19 D     1:19
                   22+  RESULT                     1:18 S     1:34 D
                   50   S@ARN      $SYSTXT         1:19
                    1   S@ARVAL    $SYSTXT         1:19
                    4   SM@SVREG   $SYSTXT         1:19 F     1:19 R
                    0   ZERO                       1: 4 D     1: 8 F     1: 8

### Snap (A4) at 0000203b;  B0 = 0000000a

A4:
  00000067


      15:35:43       0.0000   CSP             CRAY X-MP SERIAL-201/40    CRI - MENDOTA HEIGHTS, MINN. 11/24/84
      15:35:43       0.0000   CSP
      15:35:43       0.0000   CSP             CRAY OS - EDITION 218 OF XMs            COS X.15  ASSEMBLY DATE 11/19/84
      15:35:43       0.0000   CSP
      15:35:43       0.0000   CSP
      15:35:43       0.0000   CSP        JOB,JN=U1502A.
      15:35:43       0.0009   CSP        ACCOUNT,AC=,US=,UPW=.
      15:35:43       0.0054   EXP        ***********************************************************************
      15:35:43       0.0054   EXP        *
      15:35:43       0.0054   EXP        *     THIS JOB DEMONSTRATES THE DIFFERENCE BETWEEN NAMES AND SYMBOLS
      15:35:43       0.0054   EXP        *
      15:35:43       0.0054   EXP        *
      15:35:43       0.0054   EXP        ***********************************************************************
      15:35:43       0.0058   CSP        CAL.
      15:35:44       0.0060   USER       CA001 - [CAL] CAL VERSION X.15 (11/16/84) - CRAY XMP
      15:35:44       0.0138   USER       CA034 - [CAL] OPDEF LONGALD  REDEFINED IN BINARY TEXT $SYSTXT
      15:35:45       0.7548   USER       CA002 - [CAL] ASSEMBLY TIME:      0.7489 CPU SECONDS
      15:35:45       0.7548   USER       CA003 - [CAL] MEMORY WORDS: 81813 + I/O BUFFERS:  6348
      15:35:45       0.7553   CSP        LDR.
      15:35:46       0.8791   USER       LD000 - BEGIN EXECUTION
      15:35:46       0.8794   CSP        END OF JOB
      15:35:46       0.8794   CSP
      15:35:46       0.8794   CSP
      15:35:47       0.8794   USER           JOB NAME -                                U1502A
      15:35:47       0.8795   USER           USER NUMBER -                             TNG
      15:35:47       0.8795   USER           TIME EXECUTING IN CPU -                   0000:00:00.8795
      15:35:47       0.8795   USER           TIME WAITING TO EXECUTE -                 0000:00:00.7168
      15:35:47       0.8795   USER           TIME WAITING FOR I/O -                    0000:00:02.7711
      15:35:47       0.8795   USER           TIME WAITING IN INPUT QUEUE -             0000:00:00.0219
      15:35:47       0.8795   USER           MEMORY * CPU TIME (MWDS*SEC) -                     0.07390
      15:35:47       0.8796   USER           MEMORY * I/O WAIT TIME (MWDS*SEC) -                0.14242
      15:35:47       0.8796   USER           MINIMUM JOB SIZE (WORDS) -                     25600
      15:35:47       0.8796   USER           MAXIMUM JOB SIZE (WORDS) -                     89088
      15:35:47       0.8796   USER           MINIMUM FL (WORDS) -                           22016
      15:35:47       0.8796   USER           MAXIMUM FL (WORDS) -                           85504
      15:35:47       0.8796   USER           MINIMUM JTA (WORDS) -                           3072
      15:35:47       0.8796   USER           MAXIMUM JTA (WORDS) -                           3584
      15:35:47       0.8796   USER           DISK SECTORS MOVED -                             496
      15:35:47       0.8796   USER           USER I/O REQUESTS -                              118
      15:35:47       0.8796   USER           OPEN CALLS -                                      25
      15:35:47       0.8796   USER           CLOSE CALLS -                                     24
      15:35:47       0.8797   USER           MEMORY RESIDENT DATASETS -                         0
      15:35:47       0.8797   USER           TEMPORARY DATASET SECTORS USED -                  27
      15:35:47       0.8797   USER           PERMANENT DATASET SECTORS ACCESSED -              99
      15:35:47       0.8797   USER           PERMANENT DATASET SECTORS SAVED -                  0
      15:35:47       0.8797   USER           SECTORS RECEIVED FROM FRONT END -                  0
      15:35:47       0.8797   USER           SECTORS QUEUED TO FRONT END -                      0
```

SNAP MACRO

Outputs contents of registers:

LOCATION	RESULT	OPERAND
	SNAP	(LIST),UNIT=...,AF=,BF=;SF=,TF=,VF=,VL=

	DEFAULT	
LIST		List of registers to be snapped.
UNIT	$OUT	Output unit.
AF	(8(3X,08))	Format of A registers.
BF	(8(3X,08))	Format of B registers.
SF	(4025)	Format of S registers.
TF	(4025)	Format of T registers.
VF	(4025)	Format of V registers.
VL	VL	V register elements to be snapped.

EXAMPLE:

```
          SNAP    (A,S)
          SNAP    (T),TF=(3F20.10)
          SNAP    (VL),VL=20
          SNAP    (B10-B20)
```

```
  THIS IS AN ADD OF OP1 & OP2
0*** Snap (A,B,S) at 0000206b;  B0 = 0000204a
0A0 through A7:
   77777066   00000005   00000006   00000013   00057066   00060000   00000000   00000000
0B0 through B77:
   00001020   00000000   00000000   00000000   00000000   00000000   00000000   00000000
   00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
   00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
   00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
   00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
   00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
   00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
   00000000   00000000   00051230   00000000   00000000   00000000   00000000   77777066
0S0 through S7:
   0000200000000000000000   0000200000000000000000   0000000000000000057065   0000000000000100057066
   1040000000000000057065   0000000000000000000000   1000000000000000000000   0000000000000000000000

     RESULT =        11
  THAT WAS THE RESULT
```

DUMP MACRO

Dump contents of memory.

All registers are saved and restored.

LOCATION	RESULT	OPERAND
	DUMP	(LIST),UNIT=...

LIST List of memory ranges.

F..L (dumps from first word address to last word address)

F(N) (dumps N words starting at first word address)

F (dumps the first word)

UNIT Output unit. Default is $OUT.

EXAMPLE:

```
X     CON     . . .
      .
      .
      .
Y     DATA    . . .
A     BSS     . . .
      DUMP    (X..Y+1)            Dumps contents of X through
                                  Y plus one word.

      DUMP    (X(10),Y(20))       Dumps the first 10 words of both
                                  X and Y.

      DUMP    (X..Y,A)            Dumps contents of X to first
                                  word in Y, plus the contents of
                                  A.

      DUMP    (R.A2..R.A5)        Dumps from the address stored
                                  in A2 to the address stored in
                                  A5.

      DUMP    (@R.A6(100))        Dumps 100 words starting with
                                  the address pointed to by the
                                  contents of A6.
```

```
 THIS IS AN ADD OF OP1 & OP2
0*** Dumping (0'0..0'300) at 0000206b
     User memory, 00000000 through 00000277:

 000000: 000000000351040000000 000003000000000000000 000000000000000001000 000000000000000000000    :D
 000004: 000000000000000000000 000000000000000000000 000000000000000000000 000000000000000000000
                                         - same -
 000100: 052461152300622020000 036000000351040006200 000002000520000051200 000015071770000051146 U1502A  <  :D d    T  R       R
 000104: 000001000000000000000 114450700065000000000 041517246201301343046 000000000000000000000          (  @   COS X.14
 000110: 000000000000000000000 000000000000000000000 000000000000000000000 037440000000000000000                         ?
 000114: 000000000000000000000 000000200000000000000 000000000000000000002 000000000000000000000
 000120: 000000000000000000000 000000000000000000000 000000000000000000000 000000000000000000000
                                         - same -
 000164: 000000000000000000000 000000000000000000000 030063136310601363406 030461164300671643247                 03/20/8411:07:5
 000170: 000000000000000000000 000000000000000000000 000000000000000000000 000000000000000000000
 000174: 000000000000000000000 030063136310601363406 030461164300671643247 000000000000000000000         03/20/8411:07:58
 000200: 043202200400005565111 040000000040100010010 000270101000005322230 020600001440160015566 F @@ RJ@       @      $ ]
 000204: 024601400400005250020 000253141452206000025 003404440000034062400 110000006014160000165 )   @     0           (
 000210: 100000006014520010000 001603017020070102230 022400101000036522010 000267103000006771610     *         $ %      @  ]
 000214: 000001471000000051630 000003472000000100700 025074500400005302060 000346034000542002460                 *< @  ]     ,@)
 000220: 020600001634160002603 024601017022002000027 020200001310446122060 000350034003335502460 ]     , )    @      $ ]       )
 000224: 040100001335000000001 004000016076216000030 130200001350476212710 000001260641322612020 @@  @               '  @  XhZX
 000230: 000101261101324605215 017000005175420421700 001237541400005673040 000274542400005733060  AX Z Th  .  b              @
 000234: 000276543400005771000 000300440400006031020 000302441400006071040 000304442400006131060            @               @
 000240: 000306215000414000035 007000235520514032030 000273502000005712050 000275503000005752070   F ]     N )          @
 000244: 000277400000006010010 000301401000006050030 000303402000006110050 000305403000006152020        @               @
 000250: 000272403400006160361 040000000000100000000 000000000000000000000 000000000000000000000         @
 000254: 000000000000000000001 020040250441112462011 051440202470402024210 020117214201172403044           THIS IS AN ADD OF OP1
 000260: 023040236500621002004 020040100200401002004 020040250441012502012 040523100521102122012 & OP2             THAT WAS THE
 000264: 042523252461241002004 020040100200401002004 020040100200401002004 022117252520000000000 ESULT                   $OUT
 000270: 000000000000007777775 000000000000000000000 000000000000000000000 000000000000000000000
 000274: 000000000000000000000 000000000000000000000 000000000000000000000 000000000000000000000

    RESULT =          11
 THAT WAS THE RESULT
```

OPEN MACRO

Prepares a dataset for processing.

Makes a DNT entry.

Creates a DSP and LFT and an I/O buffer at high end of a job's memory if needed.

OPEN generates a two-word Open Dataset Name Table (ODN) the first time the macro is encountered.

LOCATION	RESULT	OPERAND
	OPEN	DN,PD

DN - Dataset name.

PD - Processing direction:
I if dataset opened for input.
O if dataset opened for output.

CLOSE MACRO

Terminates I/O processing on a dataset.

Writes Record Control Words (RCWs).

Flushes buffers if:
1. It is opened for output.
2. No end of data written.
3. Sequential.
4. DSP managed by COS.
5. Block dataset.
6. Not memory resident.

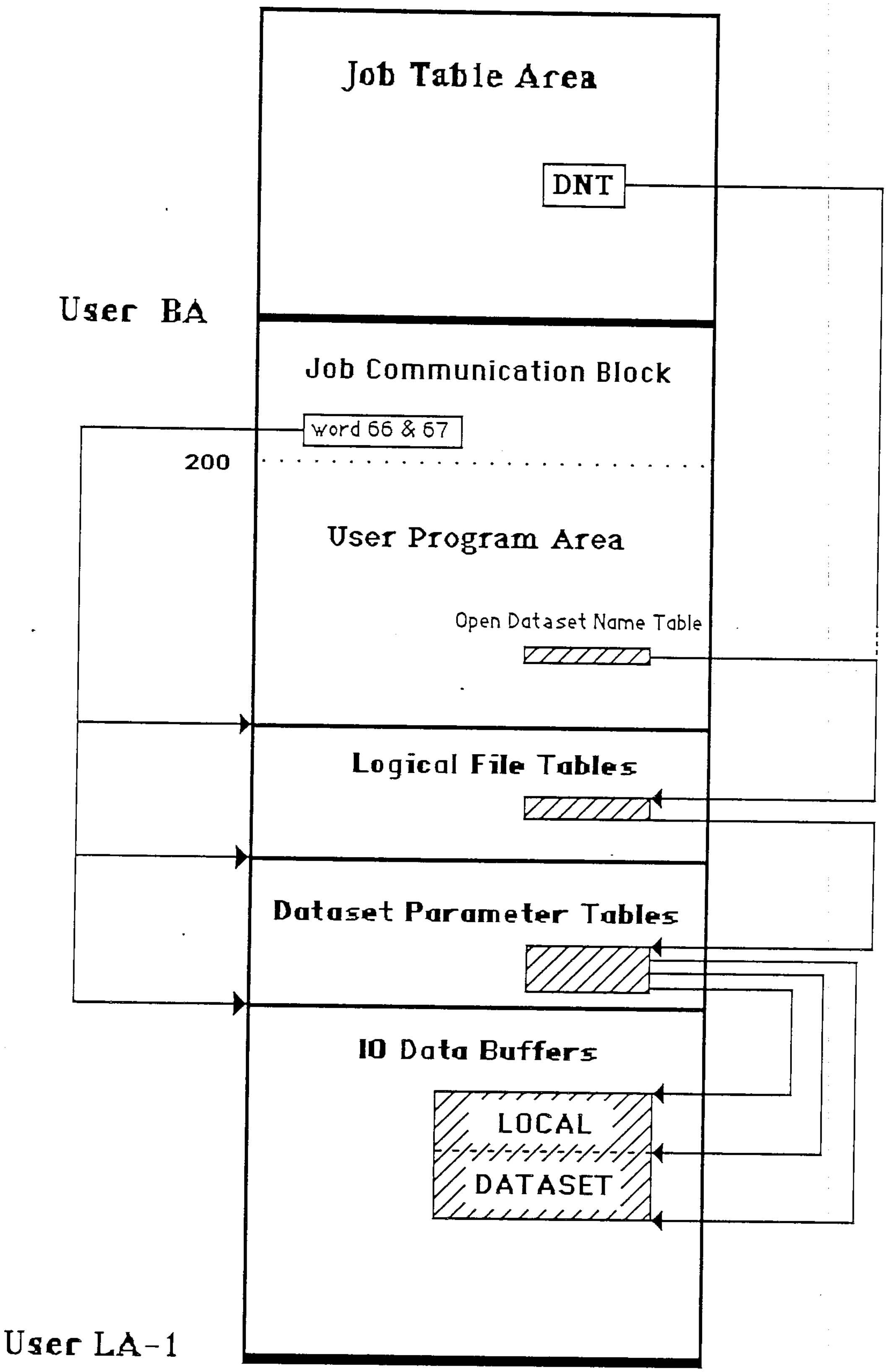
Releases buffer, LFT, and DSP table area.

Updates DNT.

LOCATION	RESULT	OPERAND
	CLOSE	DN

DN - Dataset name.

User Job Area



WRITE MACROS

LOCATION	RESULT	OPERAND	COMMENT
	WRITE	DN,UDA,CT	WRITE WORDS
	WRITEP	DN,UDA,CT	WRITE WORDS PARTIAL
	WRITEC	DN,UDA,CT	WRITE CHARACTERS
	WRITECP	DN,UDA,CT	WRITE CHARACTER PARTIAL

On partial writes an EOR is not written.

DN - Dataset name.

UDA - FWA of user data area or A,B, or S register (not A1) containing FWA.

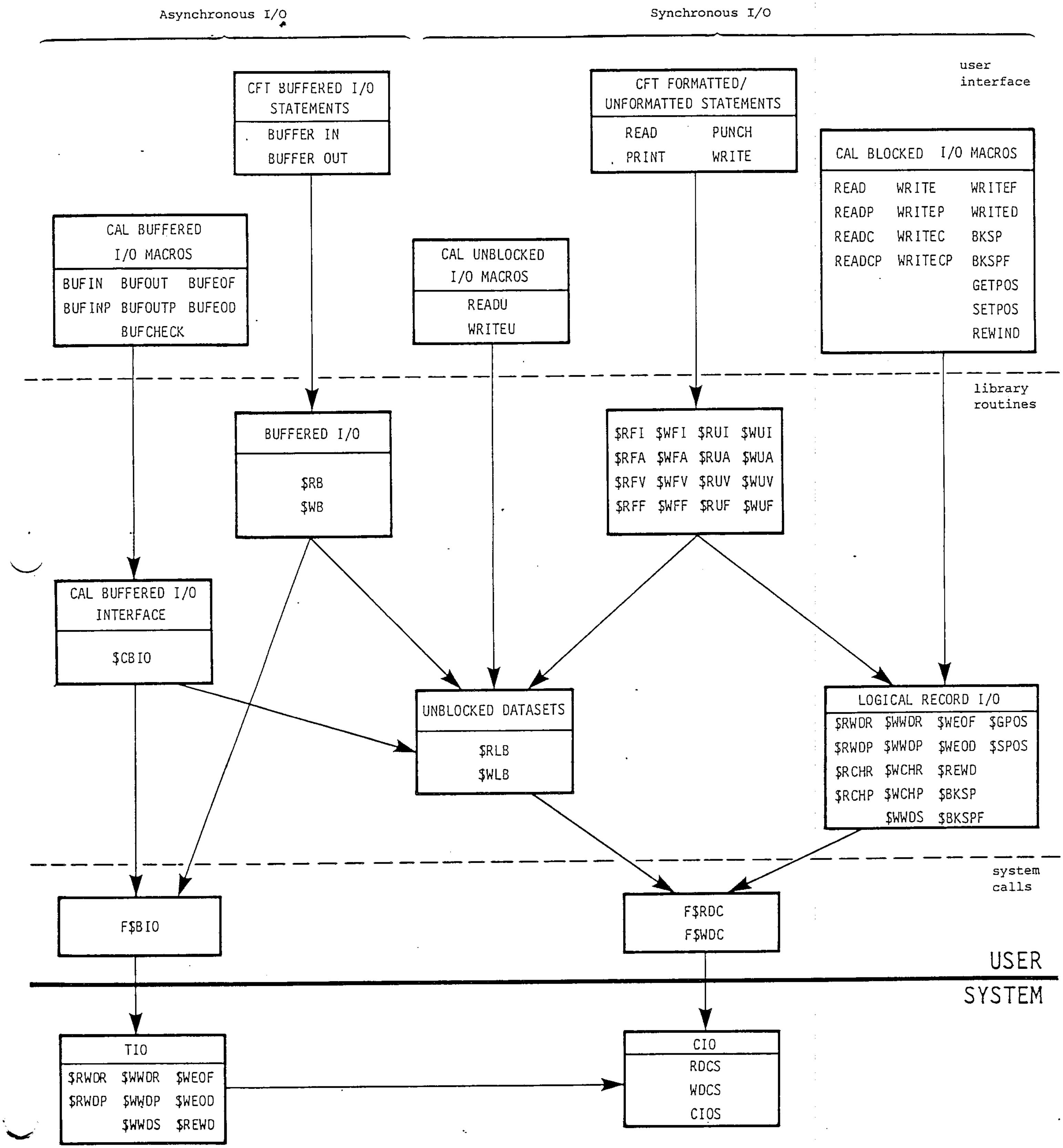
CT - Word or character count or A,B, or S register (not A1 or A2) containing count.

Return conditions:

A1 - DSP address.

A2 - FWA of user data area.

A3 - Requested word or character count.


Asynchronous I/O
Synchronous I/O
user interface
CFT BUFFERED I/O STATEMENTS
BUFFER IN
BUFFER OUT
CFT FORMATTED/ UNFORMATTED STATEMENTS
READ PUNCH
PRINT WRITE
CAL BLOCKED I/O MACROS
READ WRITE WRITEF
READP WRITEP WRITED
READC WRITEC BKSP
READCP WRITECP BKSPF
GETPOS
SETPOS
REWIND
CAL BUFFERED I/O MACROS
BUFIN BUFOUT BUFEOF
BUFINP BUFOUTP BUFEOD
BUFCHECK
CAL UNBLOCKED I/O MACROS
READU
WRITEU
library routines
BUFFERED I/O
$RB
$WB
$RFI $WFI $RUI $WUI
$RFA $WFA $RUA $WUA
$RFV $WFV $RUV $WUV
$RFF $WFF $RUF $WUF
CAL BUFFERED I/O INTERFACE
$CBIO
UNBLOCKED DATASETS
$RLB
$WLB
LOGICAL RECORD I/O
$RWDR $WWDR $WEOF $GPOS
$RWDP $WWDP $WEOD $SPOS
$RCHR $WCHR $REWD
$RCHP $WCHP $BKSP
$WWDS $BKSPF
system calls
F$BIO
F$RDC
F$WDC
USER
SYSTEM
TIO
$RWDR $WWDR $WEOF
$RWDP $WWDP $WEOD
$WWDS $REWD
CIO
RDCS
WDCS
CIOS

READ MACROS

LOCATION	RESULT	OPERAND	COMMENT
	READ	DN,UDA,CT	READ WORDS
	READP	DN,UDA;CT	READ WORDS PARTIAL
	READC	DN,UDA,CT	READ CHARACTERS
	READCP	DN,UDA,CT	READ CHARACTER PARTIAL

On partial reads the dataset is positioned after the last word or character read. Otherwise the dataset is positioned after the EOR.

DN - Dataset name.

UDA - FWA of user data area or A,B, or S register (not A1) containing FWA.

CT - Word or character count or A,B, or S register (not A1 or A2) containing count.

Return conditions:

A1 - DSP address.

A2 - FWA of user data area (UDA).

A3 - Requested word or character count.

A4 - Actual LWA+1 of data transferred (should be A2+A3).

S0 - if <0, EOR.
- if =0, null record EOF, EOD.
- if >0, count exhausted before EOR.

S6 - Contents of RCW if S0=0.

```
*23456789 123456789 1234567899 123456789
         IDENT     COPY
         START     HERE
HERE     =         *
         OPEN      INDATA
         OPEN      $OUT
         WRITE     $OUT,MESSAGE,4         WRITE HEADER MESSAGE
*
LOOP     =         *
         READ      INDATA,BUFFER,10       READ A RECORD
         JSZ       ENDFILE                S=0 IF END
         A7        A4-A2                  NUMBER OF WORDS READ
         WRITE     $OUT,BUFFER,A7         WRITE RECORD
         J         LOOP
*
ENDFILE  =         *
         CLOSE     INDATA
*
         ENDP
*
MESSAGE  DATA      '1  THIS IS A LIST OF INDATA      '
*
BUFFER   BSS       12                      READ/WRITE BUFFER
*
         END
```

FORTRAN-LIKE I/O MACROS

LOCATION	RESULT	OPERANDS
	FREAD	FMT,(LIST),SV=...,UNIT=...,ERR=...,END=...
	FWRITE	FMT,(LIST),SV=...,UNIT=...
	UREAD	UNIT,(LIST),SV=...,ERR=...,END=...
	UFWRITE	UNIT,(LIST),SV=...

FMT - Address of a format of character string enclosed in double parentheses.

(LIST) - List of addresses.

SV - Save flag (save register contents); default is no.

UNIT - Local dataset name.

ERR - Branch address if error occurs.

END - Branch address if EOF occurs.

EXAMPLE:

```
A          BSS        20
FMTA       DATA       '(2F5,3,I10)'
X          BSS        1
Y          BSS        1
Z          BSS        1
           .
           .
           .
           FREAD      FMTA,(X,Y,Z)
           .
           .
           .
           FWRITE     ((15,2F10,2)),(Z,X,Y),SV
           .
           .
           .
           UREAD      DATA,(A,10,2))
```

```
JOB,JN=U1502A.
ACCOUNT,AC=265124,US=TNG,UPW=TNG.
********************************************************************************
*
*     THIS JOB ADDS TWO NUMBERS AND WRITES THE RESULT TO $OUT
*
*******************************************************************************
CAL.
LDR,MAP=ON.
/EOF
         IDENT     FWRITE
         START     HERE
*
HERE     =         *
         OPEN      $OUT
         WRITE     $OUT,MSG1,5            WRITE HEADER MESSAGE
*
         A1        OP1,0
         A2        OP2,0
         A3        A1+A2
         SNAP      (A)
         RESULT,0  A3
         DUMP      (O'200..O'300)
*
         FWRITE    (('     RESULT =  ''I8)),(RESULT)
         WRITE     $OUT,MSG2,5            WRITE TRAILER
         CLOSE     $OUT
*
         ENDP
*
OP1      CON       5
OP2      CON       6
RESULT   BSS       1
MSG1     DATA      '  THIS IS AN ADD OF OP1 & OP2     '
MSG2     DATA      '  THAT WAS THE RESULT             '
*
         END
/EOF
```

COS RELOCATABLE LOADER

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 the loading of a program in relocatable format. Absolute load modules can also be loaded.

LDR CONTROL STATEMENT

Format:

LDR,DN=dn,LIB=ldn,NOLIB=ldn,LLD,AB=adn,MAP=op,
SID='string',T=tra,NX,DEB=l,C=com,OVL=dir,CNS,
NA,USA,L=ldn,SET=val,E=n,I=sdir,NOECHO,NORED,
SECURE,GRANT=sc_1:sc_2:...:sc_n:,BC=bc,PAD=pad.

LOADER LINKAGE PSEUDOS

Linking object program modules into a single executable program module.

MODULE - Defines contents of module type field.

ENTRY - Specifies symbols, defines as addresses or values, so they can be used by other program modules linked by the loader.

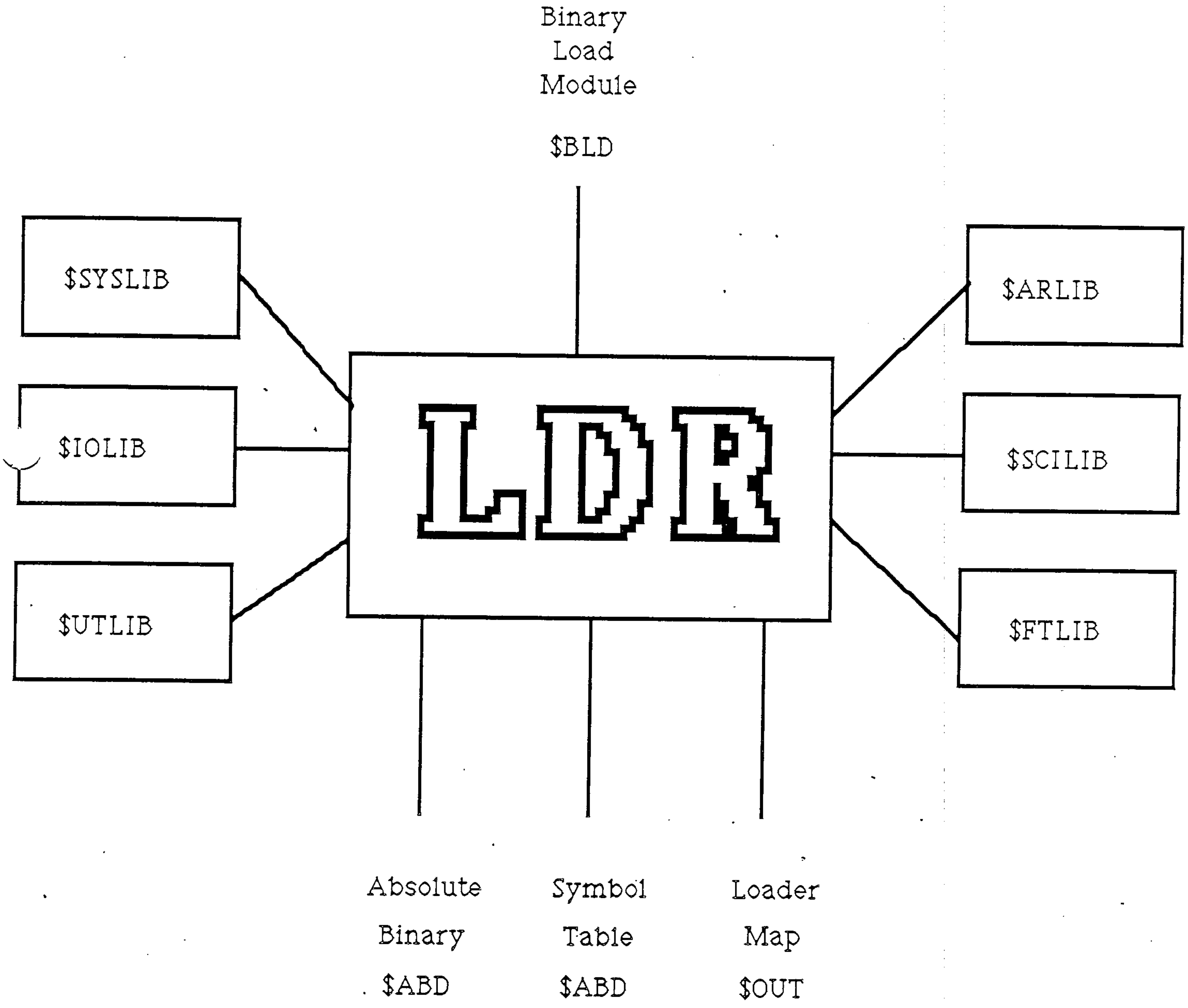
EXT - Specifies linkage to subroutines defined as entry symbols in other program modules.

START - Specifies symbolic address where execution begins.

NEWSEQ - Notifies loader of the use of new CFT calling sequence.

STACK - Notifies loader that stack structure is in effect.

Loader


Binary
Load
Module
$BLD
$SYSLIB
$IOLIB
$UTLIB
LDR
$ARLIB
$SCILIB
$FTLIB
Absolute
Binary
$ABD
Symbol
Table
$ABD
Loader
Map
$OUT

RELOCATABLE LOADER EXAMPLES

Example 1 - LDR.

```
LDR.
```

The simplest form. All parameters are defaulted.

Example 2 - DN.

```
LDR,DN=DONPROG.
```

The program module will be loaded from dataset DONPROG. Other parameters are defaulted.

Example 3 - LIB.

```
LDR,DN=DONPROG,LIB=DONSLIB.
```

The loader will search dataset DONSLIB in addition to the system default libraries ($FTLIB,$SCILIB,etc.) for the loading and linking of externals.

Example 4 - NOLIB.

```
LDR,DN=DONPROG,LIB=DONSLIB,NOLIB=$SCILIB.
```

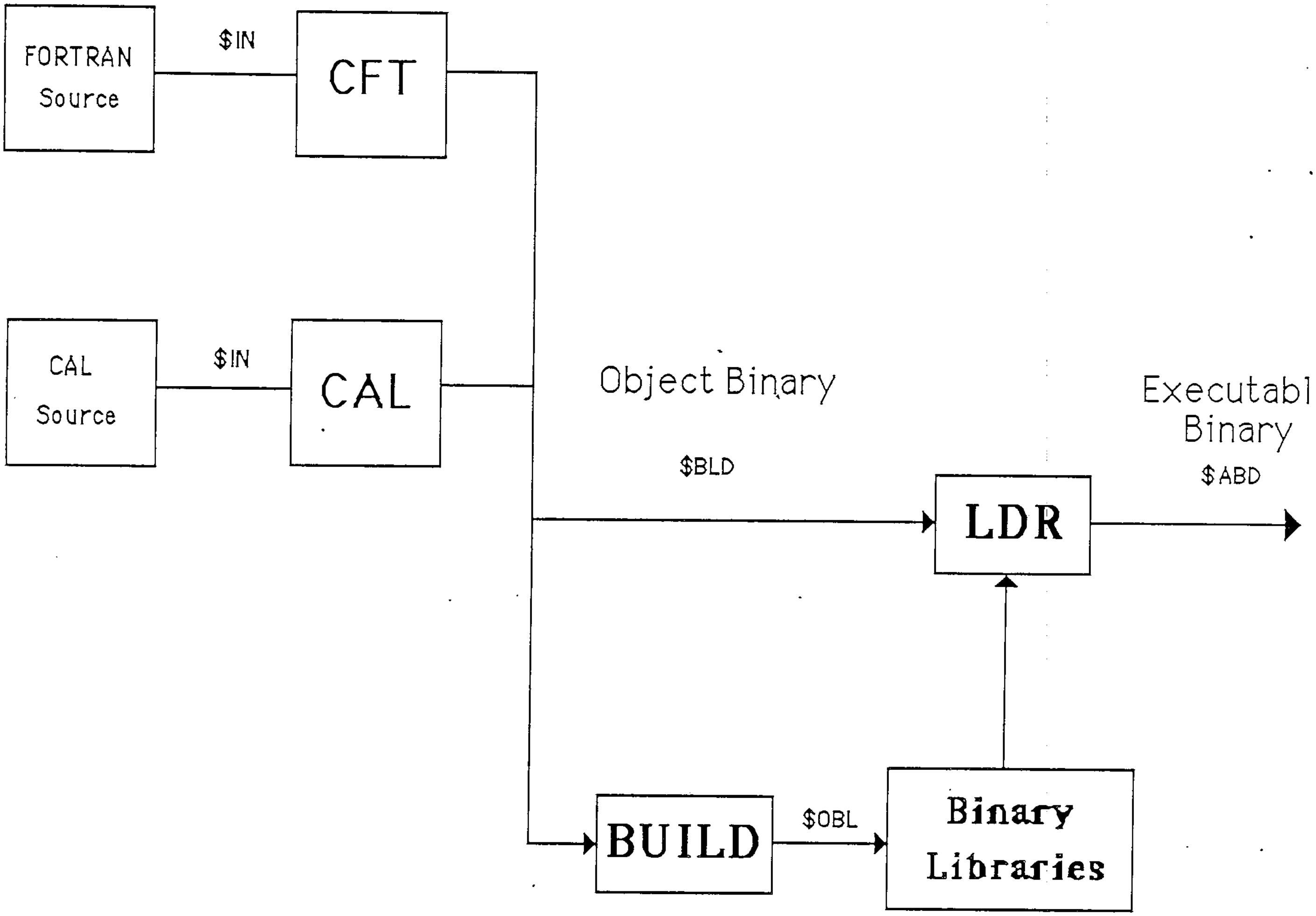
The loader will exclude dataset $SCILIB in its search for loading and linking externals.

Example 5 - LLD.

```
LDR,DN=DONPROG,LIB=DONSLIB,NOLIB=$SCILIB,LLD.
```

Any libraries that are possibly accessed during this load ($FTLIB,$SYSLIB,DONSLIB, etc.) are not released following the load. This means the buffer area(s), DNT(s), etc. remain in the program area.

OBJECT BINARY VS EXECUTABLE BINARY



Example 6 - AB.

```
LDR,DN=DONPROG,LIB=DONSLIB,NOLIB=$SCILIB,LLD,
     AB=DONS.
```

The loader will construct a memory image (all externals linked) of the program DONPROG on dataset DONS. DONS could then be made permanent on the Cray and executed at a later time by simply using the dataset name, as in:

```
JOB,JN=NOW.
ACCESS,DN=DONS.
DONS.
EXIT.
```

Basically, CFT, LDR, CAL, etc. are constructed in the same manner.

Example 7 - MAP.

```
...LLD,AB=DONS,MAP=ON.
```

A map of the loaded program is produced on $OUT. Refer to SR-0011 for a LOAD MAP example.

Example 8 - NX.

```
...MAP=ON,SID,T=HERE,NX.
```

When NX is used there is no execution of the loaded program. However, all externals will be loaded to ensure a complete program as in:

```
JOB,JN=NOW.
CFT.
LDR,AB=DONS,NX.
SAVE,DN=DONS.
EXIT.
```

This program creates and saves object program 'DONS' with all the linkages to subroutines and externals, but <u>does</u> <u>not</u> execute the program.

LOADER MAP

```
          RELOCATABLE LOAD

     LOAD TRANSFER IS TO     HERE       AT   (      200a )
      DATASET      BLOCK        ADDRESS     LENGTH       DATE       OS REV      PROCSSR     VER.       COMMENT

                   *SYSTEM            0        200
      $BLD         FWRITE           200        124   06/14/84   COS 1.13   CAL 1.13   06/06/84
      $IOLIB       $CDCO            324        404   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $IBMO            730        337   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $IOERP          1267       1527   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $WFD            3016       2027   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $WUT            5045       1643   06/06/84   COS X.14   CAL 1.13   06/06/84
      $UTLIB       $BTD            6740        102   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $BTO            7100         76   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $CDCPACK        7176         75   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $CDCTRAN        7273        777   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $DEALLOC       10272        113   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $IBMPACK       10405        146   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $IBMTRAN       10553       1023   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $NCON          11576        173   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $NOCV          11771        452   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $SCHED         12443       6217   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $UTERP         20662        242   06/06/84   COS X.14   CAL 1.13   06/06/84
      $SYSLIB      $DSNDSP        21124         17   06/06/84   COS X.14   CAL 1.13   06/06/84
                   GPOS           21143        127   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $GTDSP         21272        111   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $INSASCI       21403         70   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $PBN           21473        162   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $PRCW          21655        123   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $RCW           22000        730   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $REWD          22730        157   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $SLERP         23107       2533   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $SLFT          25642         72   06/06/84   COS X.14   CAL 1.13   06/06/84
                   SPOS           25734        401   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $TRBK          26335       1606   06/06/84   COS X.14   CAL 1.13   06/06/84
                   TRBKLVL%       30143         57   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $UEOFTCL       30222         13   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $WCH           30235        340   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $WRTUTIL       30575        346   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $WWD           31143        600   06/06/84   COS X.14   CAL 1.13   06/06/84
      $ARLIB       $ARERP         31743        124   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $DASS          32100        134   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $DDSS          32240         63   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $DMSS          32340        104   06/06/84   COS X.14   CAL 1.13   06/06/84
                   $LDIVSS        32500        104   06/06/84   COS X.14   CAL 1.13   06/06/84
      $FTLIB       $UTIL          32604        166   06/06/84   COS X.14   CAL 1.13   06/06/84

                 BLOCK NAME   ENTRIES        ENTRY VALUE     ABSOLUTE REFERENCES

                    FWRITE    HERE              200a
                    $CDCO     $CDCO             362a        5616d
                    $IBMO     $IBMO             767a        5607d
                    $IOERP    IOERP%           1366a        3757d       4366b       5173d       6663a
                              NLERP%           1432a
                    $WFD      $WFI             3613a         212c


   *** LOAD IMAGE STATISTICS ***
   ABSOLUTE BINARY LENGTH:     13818(10),       32772(8) WORDS
   PROGRAM IMAGE: FWA =          200(8),     LWA =       33172(8)
    THIS IS AN ADD OF OP1 & OP2
       RESULT =          11
    THAT WAS THE RESULT
```

CAL PROGRAMMING QUIZ

1. How is a source dataset represented and is it blocked or unblocked?

2. Name 6 binary libraries and a routine in each library?

_______________ _______________________

_______________ _______________________

_______________ _______________________

_______________ _______________________

_______________ _______________________

_______________ _______________________

3. What is the main purpose of the loader?

4. How and when is a loader map used?

5. Where do local datasets reside?

6. What CAL statement will dump a job's memory and format it to $OUT?

7. Explain why you would use a macro?

8. What's the difference between a source listing and an assembly listing?

9. Where do you look to find a symbol's value?

10. What loader command parameter prevents execution of the object program?

Program Libraries and UPDATE

5

MODULE OBJECTIVES

Upon completion of the Program Library Module, and with the aid of all furnished reference material, the learner should be able to:

1. Create a diagnostic program library
2. Modify a diagnostic program library
3. List the decks of a diagnostic program library
4. Create a binary library

LIBRARIES

Procedure Library

Created by the JCL PROC definition

Library statement makes the PROC available

Defined JCL stream call

Program Library

Created and maintained by UPDATE

Program source code

Composed of decks

System has 25 common libraries

Object Library

Created and maintained by BUILD

Binary program file and directory file

System has 6 common libraries

Described by itemized statement

```
AUDIT            COS 1.14
PDN              ID          ED      PDN           ID          ED

$APTEXT          V114BF1     1       $ARLIB        V114BF1     1
$DBHELP          V114BF1     1       $FTLIB        V114BF1     1
$IOLIB           V114BF1     1       $PSCLIB       V114BF1     1
$SCILIB          V114BF1     1       $SID          V114BF1     1
$SYSLIB          V114BF1     1       $SYSTXT       V114BF1     1
$UTLIB           V114BF1     1       $UTLTXT       V114BF1     1
ACCOUNT          V114BF1     1       ACCTDEF       V114BF1     1
ADSTAPE          V114BF1     1       APML          V114BF1     1
ARLIBPL          V114BF1     1       AUDIT         V114BF1     1
AUDPL            V114BF1     1       AUTODIR       V114BF1     1
BIND             V114BF1     1       BUILD         V114BF1     1
CAL              V114BF1     1       CALPL         V114BF1     1
CFT              V114BF1     1       CFTPL         V114BF1     1
CHARGES          V114BF1     1       COMPARE       V114BF1     1
COPYD            V114BF1     1       COPYF         V114BF1     1
COPYR            V114BF1     1       COPYU         V114BF1     1
COSPL            V114BF1     1       COSTXT        V114BF1     1
CSIM             V114BF1     1       CSIMPL        V114BF1     1
DEBUG            V114BF1     1       DSDUMP        V114BF1     1
DUMP             V114BF1     1       EXTRACT       V114BF1     1
FDUMP            V114BF1     1       FLODUMP       V114BF1     1
FTREF            V114BF1     1       GENPL         V114BF1     1
IOLIBPL          V114BF1     1       IOPPL         V114BF1     1
ITEMIZE          V114BF1     1       JCSDEF        V114BF1     1
LDR              V114BF1     1       LDRPL         V114BF1     1
MODSEQ           V114BF1     1       MODSET        V114BF1     1
PASCAL           V114BF1     1       PASCLPL       V114BF1     1
PDSDUMP          V114BF1     1       PDSLOAD       V114BF1     1
PRVDEF           V114BF1     1       SCILBPL       V114BF1     1
SEGLDR           V114BF1     1       SEGRLS        V114BF1     1
SETOWN           V114BF1     1       SIDPL         V114BF1     1
SKIPD            V114BF1     1       SKIPF         V114BF1     1
SKIPR            V114BF1     1       SKIPU         V114BF1     1
SKOL             V114BF1     1       SKOLPL        V114BF1     1
SKOLREF          V114BF1     1       SKOLTXT       V114BF1     1
SPAWN            V114BF1     1       STATS         V114BF1     1
STEP             V114BF1     1       SYSLBPL       V114BF1     1
SYSREF           V114BF1     1       TEDI          V114BF1     1
TEDIPL           V114BF1     1       TOOLPL        V114BF1     1
UNB              V114BF1     1       UPDATE        V114BF1     1
UPDPL            V114BF1     1       UTILPL        V114BF1     1
UTLIBPL          V114BF1     1       WRITEDS       V114BF1     1

        84 DATASETS,    17164 BLOCKS,            8787968 WORDS
```

UPDATE UTILITY

The UPDATE utility is a Cray utility that provides the user with a method of maintaining source programs on datasets called program libraries (PL's) rather than on punched cards.

It allows the user to CREATE, MODIFY, EDIT, and UPDATE source language programs on the Cray.

A program library (PL) consists of specially formatted image decks, each separated by an EOF record.

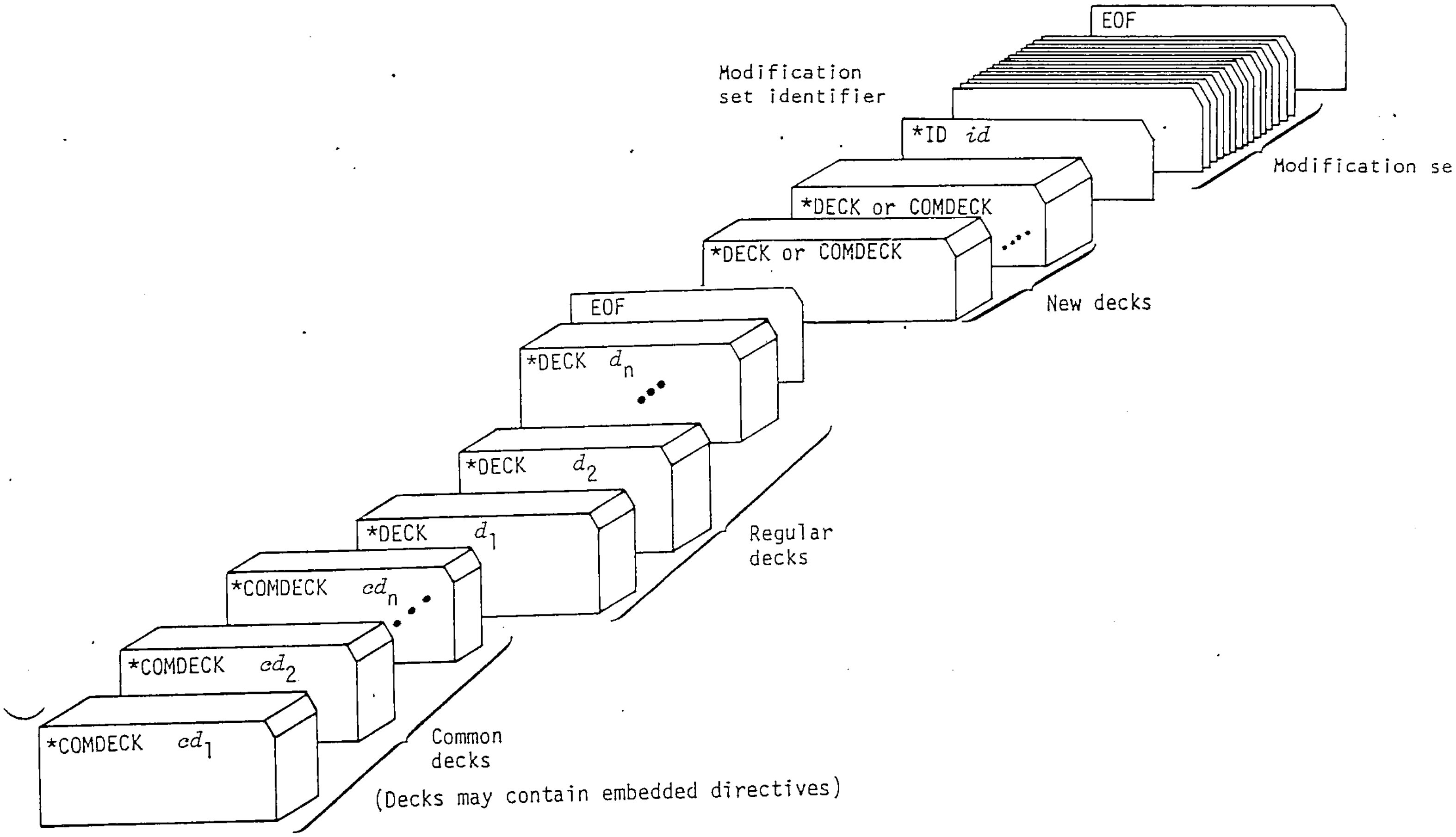
There are two deck types - regular and common.

A regular deck is sequentially placed in the PL and remains in only one location.

A common deck is sequentially placed in the PL but can be called in other locations of the PL (similar to a macro in assembly language or a statement function in FORTRAN).

PL's are used in the generation and modification of the Cray operating system - COSPL, IOPPL, and GENPL.

PL's used in the generation and modification of Cray diagnostics are CRAYPL, XMPPL, DOCPL, IOPPL, C200PL, X200PL, and I200PL.



Typical source deck input sequence

UPDATE STATEMENT

UPDATE is a program library line editor.

```
UPDATE,P=pdn,I=idn,C=cdn,N=ndn,L=ldn,E=end,S=sdn,DW=dw,
     DC=dc,*=m,l=c,Q=dk:dk,F,NA,NR,IN,ID,ED,CD.
```

Examples:

Creation of a PL

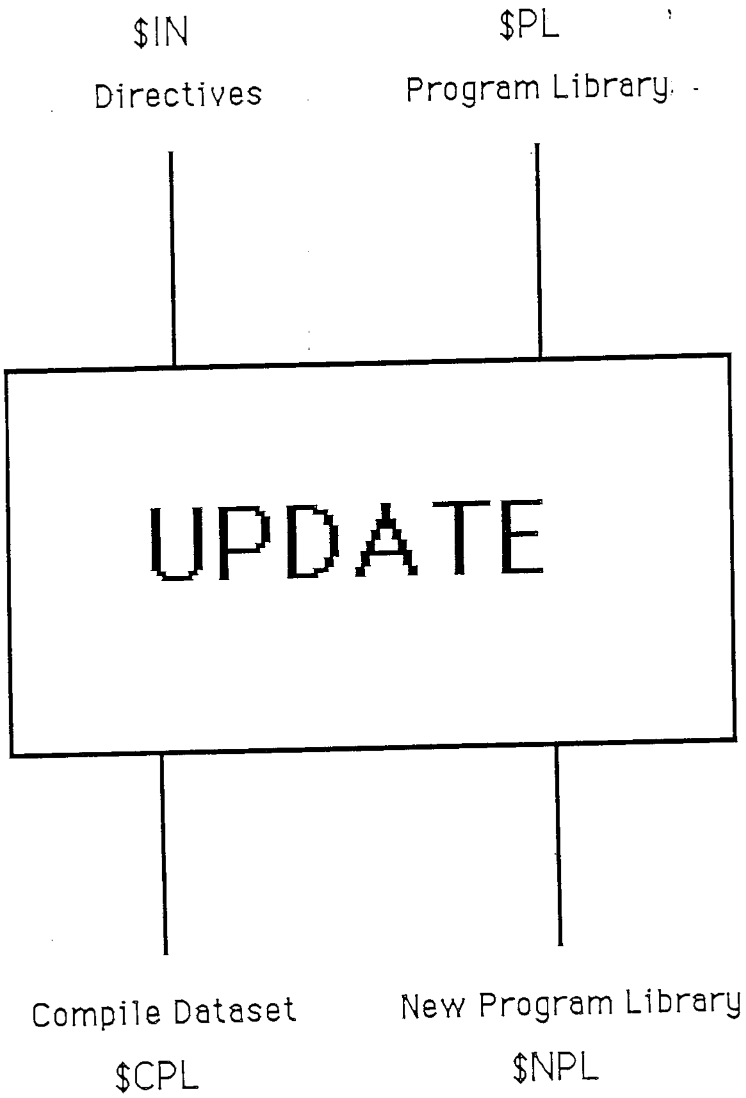
```
UPDATE,I=SOURCE:SOURCE1,P=0,N=LIBRARY.
```

Modify a deck in a PL

```
UPDATE,P=PLDPL,N=NEWPL,ID.
```

Compile a deck in a PL

```
UPDATE,P=LIBRARY,I=0,Q=*ARA..Z.
```


$IN
Directives
$PL
Program Library
UPDATE
Compile Dataset
$CPL
New Program Library
$NPL

UPDATE DIRECTIVES

Update directives include:

Commands to modify common decks

Commands to modify program decks

Commands to compile a deck

Commands to maintain a deck

The directives default dataset is $IN.

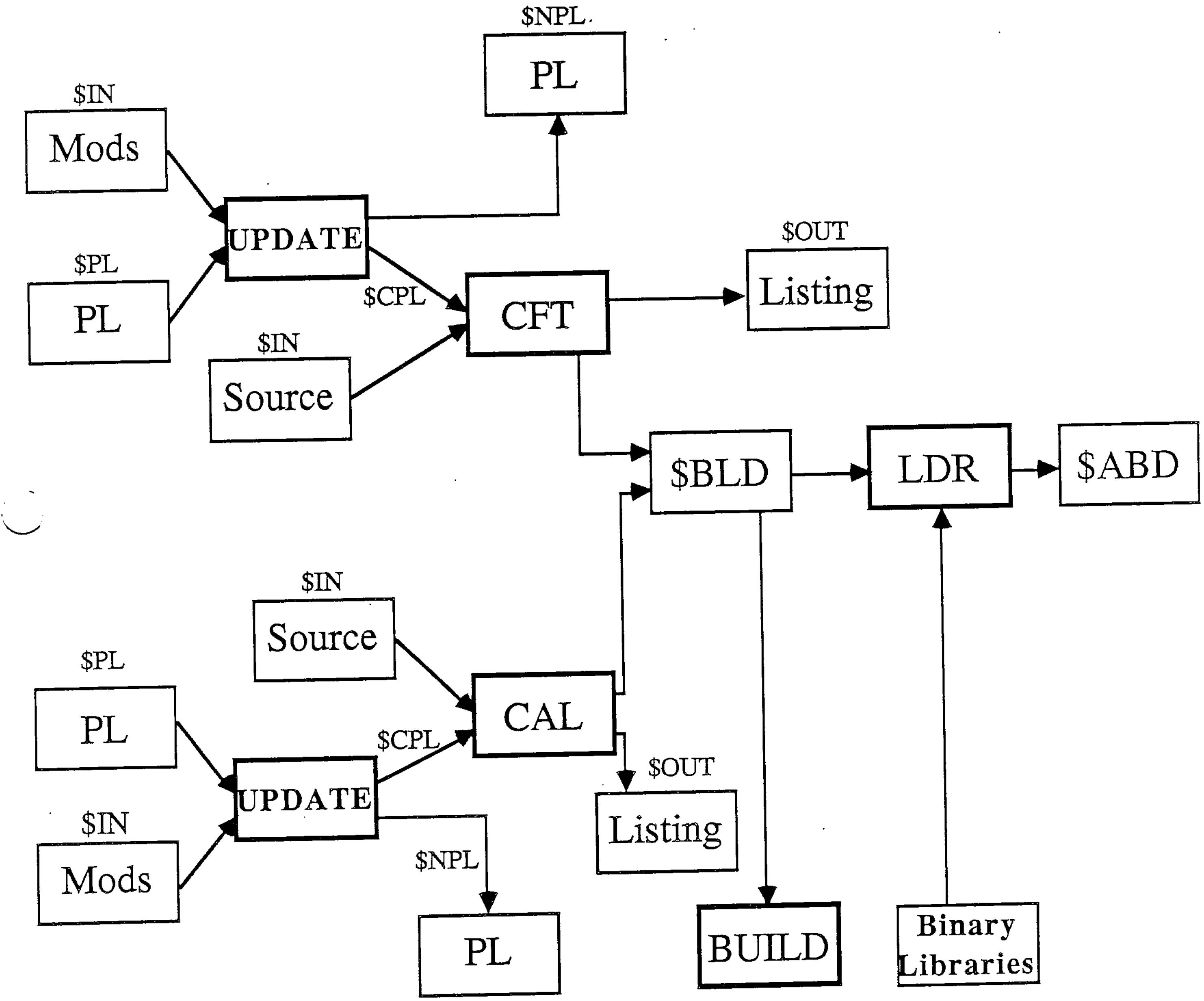
UPDATE DIRECTIVES

BEFORE	Insert before a card number
CALL	Call a common deck
COMDECK	Insert a common deck
COMPILE	Compile a deck
CWEOF	Conditional WEOF
DECK	Insert a new deck
DECLARE	Declare a deck for
DELETE	Deactivate cards
EDIT	Remove inactive cards
IDENT	Modification ID
INSERT	Insert after a card number
LIST	Resume listing to E dataset
NOLIST	Stop listing to E dataset
MOVEDK	Move deck
PURGEDK	Remove deck
READ	Read alternate input
SEQ	Write sequence numbers
NOSEQ	Stop writing sequence numbers
WEOF	WEOF on a compile dataset
YANK	Deactivates a deck
UNYANK	Reactivates a deck

```
                                               CRAY XMP  CAL X.15(01/25/85)  01/29/85 17:11:28  Page 1
COPY                                                                                              (1)

                                              IDENT    COPY                                     COPY.2
                                              START    HERE                                     COPY.3
                             0a+     HERE     =        *                                        COPY.4
 0a   <macro>                                 OPEN     INDATA                                   COPY.5
 1d   <macro>                                 OPEN     $OUT                                     COPY.6
 3c   <macro>                                 WRITE    $OUT,MESSAGE,4      WRITE HEADER MESSAGE COPY.7
                                     *                                                          COPY.8
 6a   <macro>                                 READ     INDATA,BUFFER,10    READ FIRST RECORD    CHANGES2.1
10c   031742                                  A7       A4-A2                                    CHANGES2.2
  d   <macro>                                 WRITE    $OUT,BUFFER,A7      WRITE FIRST RECORD   CHANGES2.3
                             13b+    LOOP     =        *                                        COPY.9
13b   <macro>                                 READ     INDATA,BUFFER,10    READ A RECORD        COPY.10
15d   014 00000017a+                          JSZ      ENDFILE             S=0 IF END           COPY.11
16b   031742                                  A7       A4-A2               NUMBER OF WORDS READ COPY.12
  c   006 00000013b+                          J        LOOP                                     COPY.14
                                     *                                                          COPY.15
                             17a+    ENDFILE  =        *                                        COPY.16
17a   <macro>                                 WRITE    $OUT,BUFFER,A7                           CHANGES2.4
                                     *                                                          COPY.17
21c   <macro>                                 CLOSE    INDATA                                   COPY.18
                                     *                                                          COPY.19
46c   <macro>                                 ENDP                                              COPY.20
                                     *                                                          COPY.21
50    0304401005211022251440 MESSAGE  DATA     '1  THIS IS A LIST OF INDATA      '               COPY.22
51    0445231004044023044523
52    0520402364304022247104
53    0405242022004010020040
                                     *                                                          COPY.23
54                             14    BUFFER   BSS      12                  READ/WRITE BUFFER    COPY.24
                                     *                                                          COPY.25
                                              END                                               COPY.26
```

```
U1502A                                                            UPDATE X.13 84063                 03/15/84        11:29:59      PAGE    1
*** IDENTIFIERS IN PROGRAM LIBRARY CRAYPL    ***                                                    PLDATE 03/15/84 LASTID MENUS

**MTE8       **ADRTC       *CAET        *BET         *BTDMP       *CHE         *CHR         *DDR0        *DDR1        *CHT         *EDB
 *EJT         *EXD         *EXJ         *FPE         *JBK         *MCR         *MMI         *MPC         *MSLR        *MVWR        *MVX
 *PCI         *PDP         *RTC         *SCAT        *SR3         *TDI0        *TDI1        *TDI3        *TDM         *TDMP        *TMX
 *TPM         *VCH         *VCP         *VDMP        *MSEC        *RUN         *MIT         *CIT         *CSSCI1      *IBR         *SSFR
 *CMD         *DKSE        *DSKZ        *DKRX        *DSKR        *ADRT        *MSL8        *RUN8        *PADI        *PADO        *TDI2
 *TMX8        *CCM         *CCI         *ZZ          *WW          *INFC        *CT          *MSL         *ACE370      *DX9         *EXDS
 *BTDMPS      *VDMPS       *TDIOS       *TDI3S       *TDI1S       *TDI2S       *IRUN        *IBPOS       *IBP1S       *IBP2S       *IBP3S
 *TDMS        *CMST        *A130        *IRUN8      **DKSEC      **DSKZC      **DKRXC      **DAMLC       *DBTR        *DBRR        *DFLW
 *DFC        **DFLWC      **DFCC       **DBTRC       *BEU         *SRIO        *CHNS        *CHU         *FCS         *FCU         *SECD
  AC0163A      AC0176A      AC0177A      AC0198A      BC201A       BI5127A      BC5112A      BC5112B      BC5112C      BC5112D      BC5112E
  BC0176A      BC5137A      BC5134C      BC5112F      BC5112G      BC5112H      BC5112I      BC5112J      BC5112K      BC5112L      BC5112M
  BC5112N      BC5147A      BC5144A      BC5148A      BC5150A      BC5150B      BC5150C      BC5150D      BC5150E      BC5153A      BC5154A
  BC5161A      BC5150F      BC5150G      BC5150H      BC5150I      BC0217A      BC5170A      BC5171A      BC217B      *HTEST       **OTA
**GRN         *MIX         *BLA         *STAN        *AMT         *ARA         *ARB         *ARBA        *ARI         *ARM         *BATS
 *BRB         *BTRT        *BTV         *CBG         *CLEAR       *FUTA        *IFT         *JAB         *JCB         *SFA         *SFM
 *SFR         *SIS         *SMU         *SR1         *SR2         *SRA         *SRB         *SRBA        *SRL         *SRS         *SVC
 *TLT         *TRB         *VLT         *VPT         *VRR         *VRS         *VRB         *VRL         *VRN         *SRR         *ADD
 *VPOP        *VRA         *BJK         *VCTST       *AHT         *SAR         *SCN         *IBP1        *IBP2        *IBP3        *LADDER
 *BIT         *IBP0       **MTA        **MTI        **MTD         *MTAS        *MTIS        *MTDS       **UNIT       **DDCOMM     **DDSFNS
 *DDTEST      *DELAY       *ERRINF      *ICVTOS      *TESTSW      *SETBIT      *DATACHK     *BREAKER     *SORT        *FLAC        *CFLW
**MTIO       **MTEE       **ITA        **ITI        **ITIO       **ITEE        *ITAS        *ITIS        *ITEES       *MTEES       *MTIOS
 *ITIOS      **SXXXK       *S101K       *S104K       *S102K       *S202K       *S201K       *S204K      **OMTI        *TMXS        *DISK
 *CSSD        *ASSD        *ASSDL      **MSG0       **MSG1       **MSG2       **MSG3       **MSG4       **GLOBS      **PARAMS     **MTRCOMS
**MTS         *MTSS        *CST         *BASE        *ETH1        *QUICK1      *COMP1       *QUIK1E      *COMP1E      *ETH1E       *M101M
 *M102M       *M202M       *M204M        BC5172A      BC0217C      BC5175A      BC0201B      BC5182A      BC5182B      BC5241A      BC5254A
  BC217D       DC0283A      BC189A       DC1082A      CI0294A      DC0281A      DC0323A      DC0323B      DC0323C      EC0188A      EC0291A
  EC0683A      EC0148A      EC0334A      EC0683       EC0339A      EC0341A      EC0343A      ECKB001      EC0344A      EC0346A      EJB005
  EJB006       EJB007       EJB008       EC0BLAA      EJB009       EJB011       EJB012       EJB013       EJB014       EJB015       EJB016
  EJB017       ECKB002      ECKB003      EC0GAPA      EJB018       EC0352A      EC0352B     *SORTC        EC0353A      EC0351A     *BREAKERC
  EC0350A      EC0349A      ECKB005      ECKB004      ECKB006      ECKB007      EJB0015      EC0360A      EJB0016      EJB0018      ECKB008
 *IBR1        *IBR3        *IBR2        *IBR0        *IBRX        *DELY        *MENUS

***     200 DECKS       34 COMMON DECKS      103 CORRECTION SET IDENTIFIERS
```

PROGRAM FLOW



BUILD UTILITY

The BUILD utility is a Cray utility that provides the user with a method of maintaining object programs (modules) on datasets called library datasets.

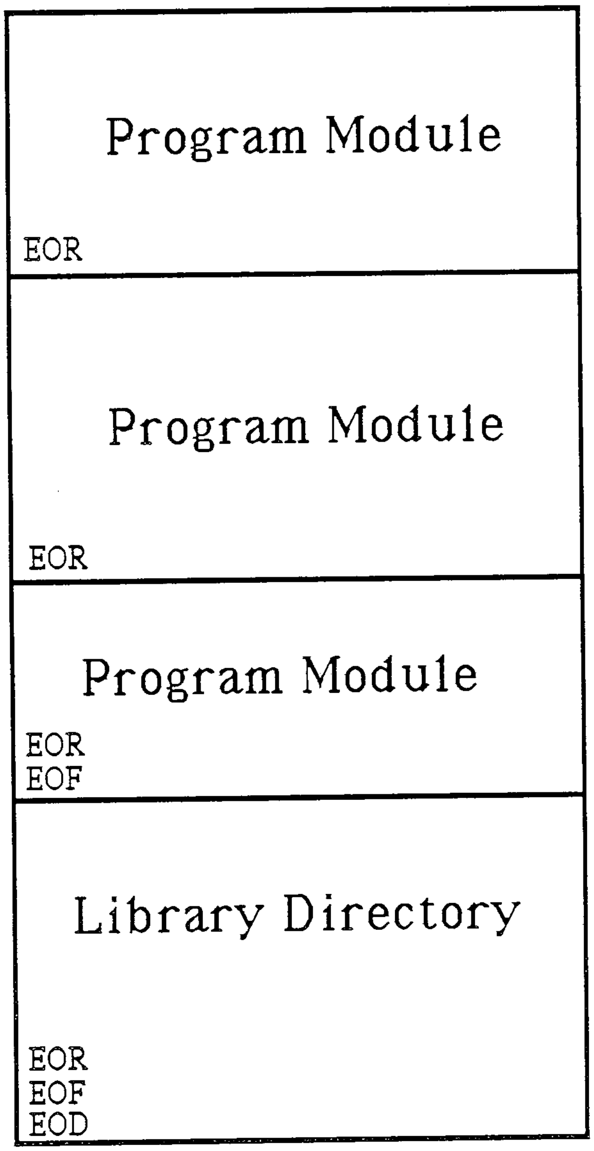
BUILD allows the user to CREATE, MODIFY, and LIST object language programs on the Cray.

A library dataset consists of two files:

> File one contains the program modules.
>
> File two contains a directory that allows the loader to rapidly locate and access the program modules.

(See the SM-0045 publication for a breakdown of tables involved with the loader.

Binary Library



BUILD EXAMPLE 1

1. The JOB card is the first statement in the JCL file.

2. The next statement compiles the first file of $IN.

3. The ACCESS statement renames the permanent dataset NLIB to the local dataset name OLIB and has the system copy the DSC information into the user JTA.

4. The next statement is BUILD. Its parameters are broken down as follows:

OBL=OLIB,	Input for the <u>old</u> library.
NBL=NEWLIB,	The <u>new</u> library name. This dataset will be used as the IBK dataset for the next modification run.
SORT,	The modules will be listed alphabetically.

5. The BUILD utility operates on the second file in $IN and searches for a directive.

6. The 'FROM OLIB' directive causes BUILD to search dataset OLIB for the module named in the next directive.

7. The 'COPY ASUB' directive causes BUILD to select the specified module from the input dataset (OLIB) and copy it to the output dataset (NEWLIB).

8. The 'FROM $BLD.COPY BSUB' directive is actually two directives on one line. Periods or semicolons separate directives on the same line.

9. The first directive on the line 'FROM $BLD' causes BUILD to search dataset $BLD for modules named in following directives. Remember the preceeding FROM used OLIB (item 6 above).

10. The second directive on the line 'COPY BSUB' causes BUILD to select the specified module from the input dataset ($BLD) and copy it to the output dataset (NEWLIB).

11. BUILD reads the /EOF and terminates.

12. The next statement deletes dataset OLIB since there is no further use for it. Two editions are not needed.

13. The next statement makes NEWLIB permanent on the Cray under the new name NLIB.

BUILD EXAMPLE 2

1. The new version of dataset NLIB (from BUILD example 1 above) is accessed before invoking LDR so the library is local to the job. NLIB could have been entered in the SDR to avoid accessing NLIB each time it is used.

2. The CAL statement assembles the program in file four of $IN and has the object code written to $BLD.

3. The program makes calls to subroutines ASUB and BSUB. Do you know where they are?

4. The next statement is 'LDR,LIB=NLIB' which names the previous BUILD library dataset (NLIB) as the library to search for unsatisfied externals and then execute the program.

BUILD PROGRAM EXAMPLE 1

```
JOB,JN=BUILD1.
CFT.
ACCESS,DN=OLIB,PDN=NLIB,UQ.
BUILD,OBL=OLIB,NBL=NEWLIB,SORT.
DELETE,DN=OLIB.
SAVE,DN=NEWLIB,PDN=NLIB.
EXIT.
/EOF
        SUBROUTINE BSUB(I)
        PRINT 10, I*I
10      FORMAT ('***THIS IS I SQUARED***',I5)
        END
/EOF
FROM OLIB
COPY ASUB
FROM $BLD.COPY BSUB
/EOF
```

BUILD PROGRAM EXAMPLE 2

```
JOB,JN=BUILD2.
ACCESS,DN=NLIB.
CAL.
LDR,LIB=NLIB.
EXIT.
/EOF
        IDENT           DON
        START           HERE
HERE    =               *
        CALL            ASUB,(DATA)
        CALL            BSUB,(DATA)
        ENDP
        CON
        DATA            10
        END
/EOF
```

PROGRAM LIBRARY QUIZ

1. What is the advantage of object libraries?

2. What is contained in a program library?

3. What utility maintains program libraries (such as inserting, deleting, and modifying decks)?

4. What statement parameter lists the decks of a PL?

5. What is the advantage of using UPDATE and a PL?

6. Write the necessary JCL statements to get a listing of program library ARA.

7. What maintains an object library by inserting and deleting modules?

8. Name three default datasets of UPDATE.

9. Name the two types of decks in a program library.

10. What are the UPDATE commands called and where in the program are they located?

Programming Exercises

6

Exercise 1 **Terminal Orientation and JCL**

Skill: Program In Job Control Language using the front-end editor

Task: Write a batch job that accesses and copies the permanent dataset SWCE1 with an ID=TNGSWCE and R=TNG to $OUT. Read what the Text dataset says.

NOTE: Start naming your programs logically, such as EX1 for this exercise, EX2 for Exercise 2, etc.

Resources:

SWCE Workbook
SR-0011
SPF Editor Reference Materials
User ID and ACCOUNT information
Terminal and station logged on to a Cray

Tools: ACCESS, COPYD.

Related Reading:

SR-0011	page 9-5
	page 12-1

Intended Lesson Results: To know how to use the front-end editor to write, submit and view the output of a Cray Batch Job.

Exercise 2 Permanent Datasets

Skill: Save, Access and Audit permanent datasets

Tasks: **NOTE:** Write each of these parts to Exercise 2 as separate programs. Use the name of the exercise part as the program name.

EX2A. Write a batch job that copies $IN to a local dataset and saves that local dataset with your ID=TNG___. Make at least 10 records in $IN.

EX2B. Write a second batch job to audit the dataset catalog with your ID and verify which disk drive the dataset is on.

EX2C. Write a batch job that accesses your permanent dataset and copies to $OUT and verify it is your dataset.

Resources:

SWCE Workbook
SR-0011
Station Terminal
SPF Editor reference material

Tools: COPYD, SAVE, ACCESS, AUDIT

Related Reading:

SR-0011	page 9-1
	page 9-5
	page 11-8
	page 12-2

Intended Lesson Results: To be able to create, locate and access a permanent dataset on Cray mass storage.

Exercise 3 Local Datasets

Skill: Manipulate COS local Datasets

Tasks:

EX3A. Write a batch job that copies the three input files listed below to a new local dataset called NUMBERS. Then copy individual records from NUMBERS to $OUT in the following order:

All records of file 2
Records 3,4,5 from file 1
The last record of file 3

The job output is to have the records shown in that order.

NOTE: Type the data starting in column 2 or your output will chop off the first character. File separators (/EOF) start in column 1, however.

FILE 1	FILE 2	FILE 3
1	11	21
2	12	22
3	13	23
4	14	24
5	15	25
6	16	26
7	17	27
8	18	28
9	19	29
10	20	30

EX3B. Add the necessary JCL to your program to dump the local dataset you are positioning (NUMBERS) in with DSDUMP. This will be used in class to examine blocked dataset structure.

Intended Lesson Results: To be able to move the dataset pointer in a local dataset, process individual records and files of a dataset, and have an understanding of text and blocked dataset structure in memory.

Exercise 4

TEDI

Skill: Write a CAL program using TEDI and an Interactive Station

Task:

EX4. Using TEDI and an interactive station, write a CAL program to square a number, modify the source program adding the JCL to assemble and execute the program. Submit it from the interactive station.

Resources:

SWCE Workbook
SG-0055
SR-0000
SR-0011
Interactive Station Terminal - VM or Unix or AOS

Tools: CAL, LDR, TEDI

Related Reading:

SR-0011	pages 14-1 to 14-9
SR-0000	pages 2-1 to 2-15
	page 4-2
	page 5-1
SG-0055	pages 1-1 to 1-5
	pages 2-1 to 2-8
	pages 3-1 to 3-5
SWCE1 Workbook	Section 3

Intended Lesson Results: To be able to write a program using the Cray interactive station and the Cray interactive text editor.

Exercise 5

CAL Programming

Skill: Program and Read Cray Assembly Language

Tasks:

EX5A. Write a CAL program to add two numbers, Use the DUMPJOB and DUMP JCL statements to find the answer. Dump the Job Table Area and words 200 to 300. Also use MAP=ON in the LDR statement.

EX5B. Use a SNAP macro of the A and S registers to look for the answer.

EX5C. Use the DUMP macro to examine the user memory area from 200 to 300.

EX5D. Use an FWRITE macro to put the result in $OUT.

Resources:

SWCE Workbook
SR-0000
SR-0012
SR-0011
CAL Card
SPF editor reference material

Related Reading:

SR-0000	Chapter 4
	pages 13-1 to 13-5
SR-0012	page 2-21
	page 2-28
	page 2-32
	page 3-33

Intended Lesson Results: To be able to write a CAL program that assembles without errors and to use various output macros to get the results of the CAL program in $OUT.

Exercise 6 Dataset Staging

Skill: Fetching CAL source programs from a front-end station

Task: The IBM dataset U1502.SWCE.CNTL(COPY) is a CAL source program. Fetch it from the front end, assemble it and execute it using the dataset you created in exercise 2. COPY is a CAL program that copies a local dataset named INDATA to $OUT.

Resources:

SWCE Workbook
SR-0011
SR-0000
SR-0012
Station Terminal
Text field for fetch statement
SPF Editor reference materials

Tools: ACCESS, FETCH, CAL, LDR

Related Reading:

SR-0000	Appendices
SR-0011	page 10-11
SR-0012	page 3-1
	page 3-5

Intended Lesson Results: To be able to get datasets to and from a front-end station's mass storage and peripheral devices - Tape and Printer.

Exercise 7

Executable Binaries

Skill: Create and use an executable binary program

Task:

EX7A. Access the dataset you created in exercise 2, access the executable binary dataset COPY with an ID=TNGSWCE and execute COPY on the local dataset INDATA.
COPY copies INDATA to $OUT.

EX7B. Create an executable binary of COPY using the source program from the front end and save it with your ID=TNG___.

EX7C. Access and execute the binary you just saved with an ID=____ on INDATA and verify that it works.

Resources:

SWCE Workbook
SR-0011
SR-0000
Station Terminal
SPF Editor reference material

Tools: ACCESS, FETCH, CAL, LDR

Related Reading:

SR-0011	page 14-4 to 14-5
SR-0000	page 3-1 to 3-5

Intended Lesson Results: To be able to know the difference between a source program, an assembler listing, a binary load module and an executable binary

Exercise 8 Program Libraries

Skill: Modify a Program Library Deck using UPDATE

Tasks:

EX8A. Compile the deck COPY from COPYPL. COPYPL has an ID=TNGSWCE. Assemble and execute COPY on your permanent dataset from exercise 2.

EX8B. Use UPDATE to modify deck COPY of COPYPL to change the banner message.

EX8C. Use UPDATE to add a trailer message to the end of INDATA.

EX8D. Get the deck IDs from X200PL.

EX8E. Write the JCL necessary to dispose a listing of SRA to the Calcomp printer.

Resources:

SWCE Workbook
SR-0000
SR-0013
SR-0000
SR-0012
SR-0011
Station Terminal
SPF Editor reference material

Tools: UPDATE, CAL, LDR, DISPOSE, ACCESS

Related Reading:

SR-0013	pages 1-1 to 1-12
	page 2-1
	page 3-1 to 3-4
	page 3-6,3-8,3-10,3-12,3-14
	Examples in section 4

Intended Lesson Results:

To be able to get a listing of all decks in a PL and to be able to use UPDATE directives to modify and compile a source program from the program library.

Exercise 9 Diagnostic Generation PROCs

Skill: Write a Procedure for generating a diagnostic listing.

Tasks:

EX9A. Write a batch job that defines and saves a procedure that will compile a deck from a diagnostic program library and dispose its listing to the IOP station printer.

EX9B. Write a job that uses the procedure you have saved to dispose a diagnostic listing to the IOP station printer.

Resources:

SWCE Workbook
CAL Reference card
SR-0011
SR-0000
SR-0013
Terminal and station logged on to the Cray
SPF Editor reference material

Related Reading:

SR-0011	page 16-21 to 16-29 page 9-5 page 10-5 page 14-1
SR-0013	page 2-1 pages 2-4 to 2-6 Chapter 4 UPDATE Examples

SPF Editor reference material

Intended Lesson Results:

To be able to create your own JCL procedures or interpret a procedure already written such as from GENPL for diagnostic generation.

APPENDIX A

GLOSSARY

GLOSSARY OF CRAY TERMINOLOGY

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.

Activation Record (AR): The element of a TASKSTACK associated with a subroutine call from within the task. An activation block/record contains: traceback addresses and local variable storage locations.

Address: (1) An identification, as represented by a name, label, or number, for a register, location in storage, or any other data source or destination such as the location of a station in a communication network. (2) Any part of an instruction that specifies the location of an operand for the instruction.

Allocate: To reserve an amount of some resource in a computing system for a specific purpose (usually refers to a data storage medium).

Alphabetic: A character set including, $, %, @, as well as the 26 upper case letters A through Z.

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

Asynchronous: A mode of operation in which performance of operations is not dependent on the completion of all previous operations. Asynchronous I/O using Buffer In and Buffer Out instructions allows process to continue without waiting for I/O to complete. (The use of Unit and Length functions set synchronization points thus changing the mode of operation back to synchronous.) The firing up (starting) of a task makes the mode of operation of a program or job asynchronous since processing will continue without waiting for the completion of the task. (The use of Events, Locks and TSKWAIT subroutines set synchronization points and may thus change mode of operation of a multitask process back to synchronous.)

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 is set by the system and must be a multiple of 208.

Blank common: 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.

Blank Common Block: A common block into which data cannot be initialized at load time. Any number of program modules may declare a blank common block and each may declare a block of a different size. The loader allocates storage to the blank common block after all other blocks have been processed.

$BLD: A dataset on which object language modules are placed by a compiler or CAL unless the user designates some other dataset.

Block: (1) A tape block is a collection of characters written or read as a unit. Blocks are separated by an interblock gap and may be from 1 through 1,048,576 bytes. A tape block and a physical record are synonymous on magnetic tape. (2) In COS 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 COS is 512 words (one sector on disk). In Cray manuals, the terms tape block and 512-word block are consistently used to distinguish between the two uses.

Block control word: A word occurring at the beginning of each block in the COS blocked format that identifies the sequential position of the block in the dataset.

BOT: Beginning of tape; the position of the beginning-of-tape reflective marker.

BOV: Beginning of volume. See BOT.

BPI: Bits per inch. COS supports the 1600 and 6250 bpi recording densities.

Buffer: A storage device used to compensate for the difference in rate of flow of data, or time of occurrence of events, when transmitting data from one device to another. It is blocks of memory used by the system to transmit data from one place to another.

Buffer memory: A 64-bit memory in the I/O Subsystem common to all I/O Processors.

Busy Wait: A wait state of a process during which the process issues legal instructions and is apparently doing normal, useful work while waiting for something to happen.

CAL: Cray Assembler Acronym.

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.

Channel: A path along which signals can be sent.

Character: A logical unit composed of bits representing alphabetic, numeric, and special symbols. COS software processes 8-bit character in the ASCII character set.

Chime: A series of pipelined instructions. The execution time for the chime is dominated by the execution time of first instruction in the sequence. Overlapping of execution times of subsequent instructions in the chime diminishes their cost.

COBEGIN: A sequence of independent program segments.

Common block: A block that can be declared by more than one program module during a load operation. More than one program module can specify data for a common block but if a conflict occurs, information from later programs is loaded over previously loaded information. A program may declare no common blocks or as many as 125 common blocks. The two types of common blocks are labeled and blank.

Conditional control statement block: Defines the conditions under which a group of control statements are to be processed. The statements which define the block and conditions are: IF, ELSE, ELSEIF, EXITIF, and ENDIF.

Control statement: The format, consisting of a verb and its parameters, used to control the operating system and access its products. Directives are used to control products.

Control statement input file: A dataset containing valid control statements as its first file.

Counting Semaphore: A mechanism that allows a fixed number of synchronization or monitoring actions to be accounted for before the mechanism is reset.

Critical Region: A part of a sequential program operating on shared data in such a way that it must have exclusive access to the shared data during its execution.

$CS: A primary control statement input file.

Dataset: A quantity of information maintained on mass storage by the Cray Operating System. Each dataset is identified by a symbolic name call a dataset name. Datasets are of two types: temporary and permanent. A temporary dataset is available only to the job that created it. A permanent dataset is available to the system and to other jobs and is maintained across system deadstarts.

Dataset name: A verb that is the name of a dataset.

Deadlock: A state resulting in the inability of processing to continue due to an unresolvable conflict. Waiting for something to happen that cannot happen results in a deadlock.

Deadly Embrace: A special case of a deadlock involving two interactive processes. Process A is waiting for process B to do something and process B is waiting for process A to do something; neither can break its own wait cycle.

Deadstart: The process by which an inactive CRAY is brought up to an operational condition ready to process jobs.

Deterministic: A property of a process which allows any future state of the process to be predicted exactly. Repeated executions of a deterministic software process will always produce the same results in the same amount of time.

Directive: A command used to control a product, such as UPDATE.

Diagnostic: (1) Pertaining to the detection and isolation of a malfunction or a mistake. (2) A message printed when its corresponding job is terminated or the dataset is released.

Disposition code: A code used in I/O processing to indicate the disposition to be made of a dataset when its corresponding job is terminated or the dataset is released.

DOALL: A loop with independent iterations.

DOPIPE: A software pipeline of program segments or iterations of a loop that are not fully independent.

Dynamic Load Balancing: A technique for distributing work evenly among parallel tasks by having the task dynamically determine the work it will do by means of run time decisions.

End-of-data-delimiter: Indicates the end of a dataset. In COS blocked format, this is a record control word with a 178 in the mode field.

End-of-file delimiter: Indicates the end of a file. (1) In COS blocked format, this is the record control word with a 168 in the mode field. (2) On magnetic tape, this is a tapemark.

End-of-record delimiter: Indicates the end of a record. (1) In COS blocked format, this is a record control word with a 108 in the mode field. (2) In an ASCII punched deck, this is indicated by the end of each card.

EOD: End-of-data on tape. The definition of EOD is a function of whether the tape is labeled or nonlabeled and of the type of operation being performed (input or output). When reading a labeled tape, EOD is returned to the user when an EOF1 trailer label is encountered. When reading a nonlabeled tape, EOD is returned when a tapemark is read on the last volume in the volume list for a particular dataset. When writing a labeled or non labeled tape, EOD processing is initiated by a write EOD, rewind, close, or release request.

EOI: End-of-information; see EOD.

EOT: End-of-tape; a status, set only on a write operation indicating sensing of the end of the tape reflective marker.

EOV: End-of-volume. On output, EOV occurs when end-of-tape status is returned on a write operation. This status occurs when the EOT reflective marker is sensed by the tape device. For input of a labeled tape dataset, EOV occurs when an EOV1 trailer label is read; for input of a nonlabeled dataset, EOV is returned when a tapemark is encountered and the volume list is not exhausted.

Event: A signal indicating an action of significance to other tasks of the same job. One means of coordinating multiple tasks is through the signaling of (posting) and waiting for (testing) an event. (*EVENTMARK)

Exchange: A mechanism for facilitating the contact switch between tasks (i.e., software processes).

Exchange Package: A 16-word block of data in an area of memory that is reserved for exchange packages. This block of data contains the contents of all of the necessary registers and conditions or mode flags which are associated with a particular program. Each program residing in memory will have an associated exchange package (refer to the CRAY-1 Hardware Reference Manual).

Execution Point: The instruction of the code associated with a task that is pointed to by the P register in an exchange package. Every task has an execution point.

Expression (JCL parameter expression): A series of characters grouped into operands and operators which are computed as one value during parameter evaluation; should be delimited by parentheses.

File: A collection of records in a dataset. In COS blocked format, a file is terminated by a record control word with 168 in the mode field.

Fork and Join: Transition points where the nature of a process changes from serial (sequential) to parallel (FORK) and from parallel to serial (JOIN).

Formal parameter specifications: Parameters in a procedure definition which identify the character strings within the procedure body that can be substituted during the procedure's evaluation.

Front-end processor: A computer connected to a Cray mainframe channel. The front-end processor supplies data and jobs to the Cray computer and processes or distributes the output from the jobs. Front-end systems are also referred to as stations in Cray publications.

Global Variable: A variable whose value is accessible throughout a program.

HEAP: A data structure providing for allocation and deallocation of variable size blocks of storage.

HLM: High limit of memory, the highest memory address available to the user for program and data area.

IDLE: The state of the computer when all jobs are completed and it is waiting for something to do.

$IN: A dataset containing the job control language statements as well as the source input and data for compiler and assemblers, unless the user designates some other dataset.

In-line procedure: A procedure defined in a control statement file..

Instruction Stream: A series of instructions to be pointed to and executed sequentially as a block of code. An instruction stream may be defined to be a task if it can be executed in parallel with another instruction stream. Instruction streams do not have their own exchange package but tasks do.

I/O Subsystem: Part of a CRAY-1S Series Model S/1200 through S/4400 consisting of two to four I/O processors and one-half, one, four, or eight million words of shared Buffer Memory. The optional tape subsystem is composed of at least one block multiplexer channel, one tape controller, and two tape units. The tape units supported are IBM-compatible 9-track, 200 ips, 1600/6250 bpi devices.

Iterative control statement block: Defines the repeated execution of a series of statements if a condition is satisfied.

JCL block control statement: A statement in the control statement file that is part of a group of control statements called a block which specifies an action to be taken by COS; the three types of blocks are: procedure definition, conditional, and iterative.

JOB: (1) An arbitrarily defined parcel of work submitted to a computing system. (2) A collection of tasks submitted to the system and treated by the system as an entity. A job is presented to the system as a formatted dataset. With respect to a job, the system is parametrically controlled by the content of the job dataset.

Job Communication Block: The first 2008 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 the inter-phase job communciation.

Job input dataset: A dataset named $IN on which the 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).

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 is automatically staged to a front-end computer for processing.

Job Step: A unit of work within a job, such as source language compilation or object program execution. Instructions to be executed and data associated with a particular control statement are parts of a job step.

JTA: User job table area. The area directly above BA. The system uses this area for job and dataset information, such as XP,DNT's, DAT's, B-T-V's, etc.

Keyword parameter: A string of 1 to 8 alphanumeric characters that consists of a keyword followed by one or more values; identified by its form rather than by its position in the control statement.

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

Library: A dataset composed of sequentailly organized records and files. The last file of the library contains a library directory. The rest of the files and records, known as entries, can consist of processed procedure definitions and/or related modules. The directory gives a listing of entry names with their associated characteristics.

Library Scheduler: A library routine that assumes primary responsibility for managing and scheduling library tasks to be connected to logical CPU's.

Library Task (Micro Task): This is a Cray term referring to tasks that are controlled by multitasking library routine and by the library scheduler (also a library routine, $SCHED).

Limit address: The upper address of a memory field. This address is maintained in the limit address (LA) register.

Literal: A symbol which names, describes, or defines itself and not something else that it might represent.

Literal constant: A string of one through eight characters delimited with apostrophes whose ordinal numbers are in the range 0408 through 1768; value of a character constant corresponds to the ASCII character codes positioned with a 64-bit word; alignment indicated can be left or right adjusted and zero-filled of left-adjusted and space-filled; apostrophes reamin as part of value.

Literal string: A string delimited with apostrophes which are normally not treated as part of the value, except with JCL block control statements which treat the apostrophes as part of the string value.

Load Balancing: A process used to insure that the amount of work done by all processors involved in a job is approximately equal (i.e., the work is split evenly among parallel tasks).

Local dataset: A temporary or permanent dataset that has pointers in a users Job table area (JTA).

Local Variable: A variable whose value is known only to the program module in which it is defined. It exists only during the execution of that module and may not be accessed or modified by other modules.

Lock: A mechanism to provide unique access to data by a segment of code. A process examines a lock before proceeding with a segment of code that requires unique access to some data (critical region). If one process is accessing the data all others must wait before entering corresponding critical regions.

$LOG: See logfile.

Logfile: During the processing of the job, a special dataset named $LOG is maintained. At job termination, this dataset is appended to the $OUT file for the job. The job logfile serves as a time-ordered record of the activities of the job -- all control statements processed by the job, significant information such as dataset usage, all operator interactions with a job, and errors detected during processing of the job.

Logical CPU: A scheduling unit. In COS this is associated with an exchange package. (*VIRTUAL PROCESSOR)

Loosely Coupled: A lower level of synchronization and communication required by software processes in a multitasking or multiprocessing environment.

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 The utilization of more than one processor to logically or functionally divide and to execute various segments in parallel.

Monitor: Controlling access to a critical region.

Multiprocessing: The utilization of more than one processor to logically or functionally divide processes and to execute various segments in parallel Multiprocessing may be simulated by one processor in a multiprogramming environment.

Multiprogramming: A technique for handling numerous routines or programs simultaneously by interleaving their execution: i.e., permitting more than one program to share machine resources (COS 1.11 is a multiprogramming operating system using jobs as the unit of user work).

Multitasking: A technique in which several separate but interrelated tasks operate under a single program or job identity. It may or may not be a form of multiprocessing.

Nesting: Including a block of statements of one kind into a larger block of statements of the same kind, such as an iterative block within a larger iterative block.

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

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 requirement for instructions exceeds the available memory.

Parallel: Objects (tasks, job steps, elements of an array) considered simultaneously (or nearly so) rather than in sequence or in some special order.

Parcel: A 16-bit portion of a CRAY 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.

Physical CPU: A processor (a real hardware entity).

Pipelining: The beginning of an operation before a previous one has been completed. Pipelining is accomplished on the Cray through the use of fully segmented functional units that allow several sets of operands to be at various stages of processing in the same functional unit at the same time.

Positional paramenter: A parameter that must appear in a precise position relative to the separators in the control statement.

Posting: The entering of a unit of information in a location to be examined for messages. An event is said to be posted when it is signaling some occurrence having taken place. Event posting is done through a call to a library routine in the Cray system.

Priority: The sequence in which various tasks and jobs will be processed. Priority 15 jobs will begin before priority 1 jobs.

$PROC: A dataset to which in-line procedure definitions are written.

Procedure: A named sequence of control statements and/or data that is saved in a library for processing at a later time when activated by a call to its name by a calling statement; provides the capability of replacing values within the procedure with other values.

Procedure definition: The definition of a procedure that is saved in a library to be called for processing at a later time.

Program library: (PL) The base dataset used by the UPDATE utility. This dataset consists of one or more specially formatted card image decks, each separated by an end-of-file.

Ready: A state of a task in which it has fulfilled all conditions for its execution and is queued for scheduling of a logical CPU (*PENDING)

Reentrant: The property of a program module that allows one copy of it to be used by more than one job or task. A mechanism is supplied by which the routines environment is preserved, i.e., working storage and control indicators are assigned independent storage location each time the routine is called. Only reentrant code can recursive call itself.

Relative address: An address defined by its relationship to the base address register such that the base address has a relative address of 0.

Roll-In: The act of reading a job into memory that had been previously rolled.

Roll-Out: The act of writing a complete job area to the disk.

Relocatable module: This is the basic program unit produced by a compiler or assembler. A relocatable module consists of several loader tables that define blocks, their contents, and address relocation information.

Scheduling Unit: An entity that can be scheduled as an independent unit by a multiprogramming operating system (eg., tasks, jobs).

Scope of a Variable: That portion of code for which the variable is defined and in which it can be referenced. In FORTRAN the portion of code is the program, subprogram or statement.

Sector: A physical area on disk equivalent to 512 64-bit words. In COS blocked format, a block is also 512 contiguous words with a block control word as the first word of the block. Therefore the internal block size for the CRAY is equivalent to one CRAY disk sector.

Serially Reusable: The property of an instruction stream that allows one copy of it to be used by more than one job or task but only one at a time. The second task wishing to enter a serially reentrant code must wait if another user has entered first and not yet exited. The routines environment must be restored to its initial condition after each use. This is referred to as single threading of the code.

Shared Data: Data which may be referenced and modified by the program modules that share it.

Single Threading: Supporting only one user at a time. See Serially Reusable.

Spin Wait: A special case of Busy Wait in which the process repeats the same set of instructions, usually including condition checking, while waiting for something to happen.

STACK: A data structure providing a dynamic sequential data list having special provisions for access from one end or the other. A last in, first out (push down, pop up) stack is accessed from just one end.

Staging: The moving of data to/from the CRAY.

Starvation: A state of deprivation of a task in which it never gets a chance to execute.

Static Load Balancing: A technique for distributing work evenly among parallel tasks y assigning equal amounts of processing to each when designing the task structure.

Suspended: A state of a task in which it cannot be executed (i.e., it doesn't have possession of a logical CPU).

Synchronous: A mode of operation in which the performance of an operation does not begin until all previous operations are complete. The normal execution of FORTRAN code including I/O statements is synchronous. Calls to subroutine and function references in FORTRAN could be viewed as synchronous operations. Synchronous I/O hardware channels operate under the restriction that the ready (for output) or the resume (for input) signal is held on during data transfer.

System dataset name: The name of a system-defined dataset in the System Directory Table (SDR); consists of an alphabetic character which can be followed by one through fourteen alphanumeric characters.

System logfile: A permanent dataset named $SYSTEMLOG.

System Task: This is a Cray term that refers to the tasks that make up part of the Cray Operating System (COS).

Task: A subjob or subprogram. A unique process that may have code and data areas in common with other tasks of the same job. A task is treated as a scheduling unit in a multitasking environment.

Task Control Array: The area in user assigned memory, but not accessible to the user job, containing all the information associated with an active task (one that has to be started but has not yet encountered the stop or return). The contents include: the tasks exchange package, pointers to the TASKSTACK and subroutines containing task code.

Task Information Block (TIB): An area in the base of TASKSTACK that contains information about the stack.

TASKSTACK: A puch down, pop up stack created upon the activation (firing up) of a task. The elements of the TASKSTACK are activated record. One activation block is created (placed on top) each time a subroutine is called and popped off when STOP or RETURN is executed.

Temporary: Short term; for immediate use only; not made permanent by saving it for long term future retrieval.

Temporary dataset: A dataset which is not permanent and is available only to the job that created it.

Tightly Coupled: A higher degree of synchronization and communication (binding) required by software processes in a multitasking environment. Tasks may handle their own communication with other tasks of the same job. *Term defined in the "Industrial Real-Time FORTRAN" Stand (IPW/EWICS TC1,2.2/80).

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

User logfile: A dataset named $LOG created for a job when it is initiated by the Job Scheduler.

User Task: This is a Cray term that refers to the tasks as they are known to the operating system COS. Each has an exchange pack and an entry in the Task Execution Table (TXT). The Library Schedule may switch Library Tasks connected to a single User Task.

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 a Cray computer is 64 bits.

APPENDIX B

PUBLICATIONS LIST

SYSTEM ACRONYMS

JCL

SR-0011	CRAY-OS (COS) Reference Manual
SR-0012	Macros and Opdefs Reference Manual
SR-0039	CRAY-OS Message Manual
SM-0040	COS EXEC/STP/CSP Internal Reference Manual
SM-0043	COS Operational Procedures Reference Manual
SM-0044	COS Operational Aids Reference Manual
SM-0045	COS Table Descriptions Reference Manual
SR-0009	FORTRAN (CFT) Reference Manual
SM-0017	FORTRAN (CFT) Internal Reference Manual
SR-0014	Library Reference Manual
SR-0060	Pascal Reference Manual
SM-0061	Pascal Internal Reference Manual
SR-0000	CAL Assembler Version 1 Reference Manual
SM-0036	APML Assembler Reference Manual

SR-0013	UPDATE Reference Manual
SG-0055	Text Editor (TEDI) User's Guide
SG-0056	Symbolic Interactive Debugger (SID) Reference Manual
SR-0066	Segment Loader (SEGLDR) Reference Manual
SM-0041	COS Product Set Internal Reference Manual
SG-0051	IOS Operator's Guide
SM-0046	IOS Software Internal Reference Manual
SM-0007	IOS Table Descriptions Internal Reference Manual

		AREA
MEP	MESSAGE PROCESSOR TASK	S
MST	MEMORY SEGMENT TABLE	S
ODN	OPEN DATASET NAME TABLE	U
OVM	OVERLAY MANAGER TASK	S
PDD	PERMANENT DATASET DEFINITION TABLE	U
PDI	PERMANENT DATASET INFORMATION TABLE	S
PDM	PERMANENT DATASET MANAGER TASK	S
PDS	PERMANENT DATASET TABLE	S
POOLTBL	POOL TABLE	S
PUT	PHYSICAL UNIT TABLE	E
R011	DISK DRIVER	E
RJI	ROLLED JOB INDEX TABLE	S
RQT	REQUEST TABLE	S
RTI	REAL TIME CLOCK INTERRUPT	E
SCB	STREAM CONTROL BYTE	S
SCP	STATION CALL PROCESSOR TASK	S
SDT	SYSTEM DATASET TABLE	S
SPM	SYSTEM PERFORMANCE MONITOR TASK	S
STG	STAGER TASK	S
STP	SYSTEM TASK PROCESSOR	S
STT	SYSTEM TASK TABLE	E
TBPT	TASK BREAKPOINT TABLE	S
TIO	TASK I/O ROUTINES	S
TQM	TAPE QUEUE MANAGER TASK	S
XP	EXCHANGE PACKAGE	-
QDT	QUEUED DATASET TABLE	S
Z	STARTUP TASK	S
$SYSLOG	CRAY HISTORY LOG	
$LOG	USER HISTORY LOG	
$BLD	OBJECT CODE FROM CFT, CAL	
$IN	JOB DATASET INPUT	
$CS	JCL FILE	
$OUT	JOB PRINT OUTPUT	

		AREA
AUT	ACTIVE USER TABLE	S
BCW	BLOCK CONTROL WORD	U/S
CBT	CHANNEL BUFFER TABLE	E
CHT	CHANNEL PROCESSOR TABLE	E
CI	CHAIN ITEM	S
CIO	CIRCULAR I/O ROUTINE	S
CMCC	COMMUNICATION MODULE CHAIN CONTROL	S
CMOD	COMMUNICATION MODULE	S
CSD	CLASS STRUCTURE DEFINITION	S
CSP	CONTROL STATEMENT PROCESSOR	U
DAT	DATASET ALLOCATION TABLE	U/S
DCT	DEVICE CHANNEL TABLE	S
DDL	DATASET DEFINITION LIST	U
DEC	DISK ERROR CORRECTION TASK	S
DET	DEVICE ERROR TABLE	S
DNT	DATASET NAME TABLE	U/S
DRT	DEVICE RESERVATION TABLE	S
DSC	DATASET CATALOG	DISK
DSP	DATASET PARAMETER AREA	US
DVL	DEVICE LABEL	DISK
DQM	DISK QUEUE MANAGER TASK	S
EQT	EQUIPMENT TABLE	S
ERR	ERROR EXIT	U/S
EX	NORMAL EXIT	U/S/E
EXEC	SYSTEM EXECUTIVE ROUTINE	E
EXP	EXCHANGE PROCESSOR TASK	S
IBT	INTERACTIVE BUFFER TABLE	S
FED	FRONT-END DRIVER	E
JCB	JOB COMMUNICATION BLOCK	U
JCL	JOB CONTROL STATEMENT LANGUAGE	U
JCM	JOB CLASS MANAGER TASK	S
JSH	JOB SCHEDULER TASK	S
JSQ	JOB SEQUENCE NUMBER	S
JTA	JOB TABLE AREA	U
JXT	JOB EXECUTION TABLE	S
LCP	LINK CONTROL PACKAGE	S
LCT	LINK CONFIGURATION TABLE	U
LFT	LOGICAL FILE TABLE	U
LIT	LINK INTERFACE TABLE	S
LOG	MESSAGE TASK	S
LST	LINK INTERFACE STREAM TABLE	S
LTX	LINK INTERFACE EXTENSION TABLES	S

JOB DEFINITION AND CONTROL

JOB - JOB IDENTIFICATION
MODE - SET OPERATING MODE
EXIT - EXIT PROCESSING
MEMORY - REQUEST MEMORY CHANGE
SWITCH - SET OR CLEAR SENSE SWITCH
* - COMMENT STATEMENT
NORERUN - CONTROL DETECTION OF NONRERUNNABLE FUNCTIONS
RERUN - UNCONDITIONALLY SET JOB RERUNNABILITY
IOAREA - CONTROL USER'S ACCESS TO I/O AREA
CALL - READ CONTROL STATEMENTS FROM ALTERNATE DATASET
RETURN - RETURN CONTROL TO CALLER
ACCOUNT - VALIDATE USER NUMBER AND ACCOUNT
CHARGES - JOB STEP ACCOUNTING
ROLLJOB - ROLL A USER JOB TO DISK
SET - CHANGE SYMBOL VALUE
ECHO - ENABLE OR SUPPRESS LOGFILE MESSAGES
LIBRARY - LIST AND/OR CHANGE LIBRARY SEARCHLIST
OPTION - SET USER-DEFINED OPTIONS

DATASET DEFINITION AND CONTROL

ASSIGN - ASSIGN DATASET CHARACTERISTICS
RELEASE - RELEASE DATASET

PERMANENT DATASET MANAGEMENT

SAVE - SAVE PERMANENT DATASET
ACCESS - ACCESS PERMANENT DATASET
ADJUST - ADJUST PERMANENT DATASET
MODIFY - MODIFY PERMANENT DATASET
DELETE - DELETE PERMANENT DATASET
PERMIT - EXPLICITLY CONTROL ACCESS TO DATASET

DATASET STAGING CONTROL

ACQUIRE - ACQUIRE PERMANENT DATASET
DISPOSE - DISPOSE DATASET
SUBMIT - SUBMIT DATASET
FETCH - FETCH LOCAL DATASET

PERMANENT DATASET UTILITIES

PDSDUMP - DUMP PERMANENT DATASET
PDSLOAD - LOAD PERMANENT DATASET
AUDIT - AUDIT PERMANENT DATASET

LOCAL DATASET UTILITIES

COPYR - COPY RECORDS
COPYF - COPY FILES
COPYD - COPY DATASET
SKIPR - SKIP RECORDS
SKIPF - SKIP FILES
SKIPD - SKIP DATASET
REWIND - REWIND DATASET
WRITEDS - WRITE RANDOM OR SEQUENTIAL DATASET

ANALYTICAL AIDS

DUMPJOB - CREATE $DUMP
DUMP - DUMP REGISTERS AND MEMORY
DEBUG - PRODUCE SYMBOLIC DUMP
DSDUMP - DUMP DATASET
COMPARE - COMPARE DATASETS
PRINT - WRITE VALUE OF EXPRESSION TO LOGFILE
FLODUMP - FLOW TRACE RECOVERY DUMP
SYSREF - GENERATE GLOBAL CROSS-REFERENCE LISTING
ITEMIZE - INSPECT LIBRARY DATASETS

CONTROL STATEMENT BLOCKS

IF - BEGIN CONDITIONAL BLOCK
ENDIF - END CONDITIONAL BLOCK
ELSE - DEFINE ALTERNATE CONDITION
ELSEIF - DEFINE ALTERNATE CONDITION
LOOP - BEGIN ITERATIVE BLOCK
ENDLOOP - END ITERATIVE BLOCK
EXITLOOP - END ITERATION

PROCEDURES

PROC - BEGIN PROCEDURE DEFINITION
PROTOTYPE STATEMENT - INTRODUCE A PROCEDURE
PROCEDURE DEFINITION BODY
&DATA - PROCEDURE DATA
ENDPROC - END PROCEDURE DEFINITION

PSEUDO INSTRUCTIONS

- Program control
  - IDENT - Identify program module
  - END - End program module
  - ABS - Assemble absolute binary
  - COMMENT - Define Program Descriptor Table comment
- Loader linkage
  - ENTRY - Specify entry symbols
  - EXT - Specify external symbols
  - MODULE - Define program module type for loader
  - START - Specify program entry
- Mode control
  - BASE - Declare base for numeric data
  - QUAL - Qualify symbols
- Block control
  - BLOCK - Local block assignment
  - COMMON - Common block assignment
  - ORG - Set *O counter
  - BSS - Block save
  - LOC - Set * counter
  - BITW - Set *W counter
  - BITP - Set *P counter
  - ALIGN - Align on an instruction buffer boundary
- Error control
  - ERROR - Unconditional error generation
  - ERRIF - Conditional error generation
- Listing control
  - LIST - List control
  - SPACE - List blank lines
  - EJECT - Begin new page
  - TITLE - Specify listing title
  - SUBTITLE - Specify listing subtitle
  - TEXT - Declare beginning of global text source
  - ENDTEXT - Terminate global text source
- Symbol definition
  - = - Equate symbol
  - SET - Set symbol
  - MICSIZE - Set redefinable symbol to micro size
- Data definition
  - CON - Generate constant
  - BSSZ - Generate zeroed block
  - DATA - Generate data words
  - VWD - Variable word definition
  - REP - Loader replication directive
- Conditional assembly
  - IFA - Test expression attribute for assembly condition
  - IFE - Test expressions for assembly condition
  - IFC - Test character strings for assembly condition
  - SKIP - Unconditionally skip statements
  - ENDIF - End conditional code sequence
  - ELSE - Toggle assembly condition
- Instruction definition
  - MACRO - Macro definition
  - OPDEF - Operation definition
  - LOCAL - Specify local symbols
  - ENDM - End macro or opdef definition
  - OPSYN - Synonymous operation
- Code duplication
  - DUP - Duplicate code
  - ECHO - Duplicate code with varying arguments
  - ENDDUP - End duplicated code
  - STOPDUP - Stop duplication
- Micro definition
  - MICRO - Micro definition
  - OCTMIC and DECMIC - Octal and decimal micros

SYSTEM ACTION REQUEST MACROS

- JOB CONTROL MACROS
  - ABORT - Abort program
  - CONTRPV - Continue from reprieve condition
  - CSECHO - Send statement image to the logfile
  - DELAY - Delay job processing
  - DUMPJOB - Dump job image
  - ENDP - End program
  - ENDRPV - End reprieve processing
  - IOAREA - Control user access to I/O area
  - JTIME - Request accumulated CPU time for job
  - MEMORY - Request memory
  - MESSAGE - Enter message in logfile
  - MODE - Set operating mode
  - NORERUN - Control detection of nonrerunnable functions
  - RECALL - Recall job upon I/O request completion
  - RERUN - Unconditionally set job rerunnability
  - ROLL - Roll a job
  - SETRPV - Set job step reprieve
  - SWITCH - Set or clear sense switch
- DATASET MANAGEMENT MACROS
  - CLOSE - Close dataset
  - DISPOSE - Dispose dataset
  - DSP - Create dataset parameter table
  - OPEN - Open dataset
  - RELEASE - Release dataset to system
  - SUBMIT - Submit job dataset
- TIME AND DATE REQUEST MACROS
  - DATE - Get current date
  - DTTS - Date and time to timestamp conversion
  - JDATE - Return Julian date
  - MTTS - Machine time to timestamp conversion
  - TIME - Get current time
  - TSDT - Timestamp to date and time conversion
  - TSMT - Timestamp to machine time conversion
- DEBUGGING AID MACROS
  - DUMP - Dump selected areas of memory
  - FREAD - Read data
  - FWRITE - Write data
  - INPUT - Read data
  - LOADREGS - Restore all registers
  - OUTPUT - Write data
  - SAVEREGS - Save all registers
  - SNAP - Take snapshot of selected registers
  - UFREAD - Unformatted read
  - UFWRITE- Unformatted write
- MISCELLANEOUS MACROS
  - GETMODE - Get mode setting
  - GETSWS - Get switch setting
  - INSFUN - Call installation-defined subfunction
  - SYSID - Request system identification

LOGICAL I/O MACROS

- SYNCHRONOUS READ/WRITE MACROS
  - READ/READP - Read words
  - READC/READCP - Read characters
  - WRITE/WRITEP - Write words
  - WRITEC/WRITECP - Write characters
  - WRITED - Write end of data
  - WRITEF - Write end of file
- ASYNCHRONOUS READ/WRITE MACROS
  - BUFCHECK - Check buffered I/O completion
  - BUFEOD - Write end of data on dataset
  - BUFEOF - Write end of file on dataset
  - BUFIN/BUFINP - Transfer data from dataset to user record
  - BUFOUT/BUFOUTP - Transfer data from user record area
- UNBLOCKED READ/WRITE MACROS
  - READU - Transfer data from dataset to user's area
  - WRITEU - Transfer data from user's area to dataset
- POSITIONING MACROS
  - ASETPOS - Asynchronously position dataset
  - BKSP - Backspace record
  - BKSPF - Backspace file
  - GETPOS - Get current dataset position
  - POSITION - Position tape
  - REWIND - Rewind dataset
  - SETPOS - Synchronously position dataset
  - SYNCH - Synchronize
  - TAPEPOS - Tape position information

PERMANENT DATASET MACROS

- PERMANENT DATASET DEFINITION MACROS
  - LDT - Create label definition table
  - PDD - Create permanent dataset definition table
  - ACCESS - Access permanent dataset
  - ADJUST - Adjust permanent dataset
  - DELETE - Delete permanent dataset
  - PERMIT - Explicitly permit dataset
  - SAVE - Save permanent dataset

CFT LINKAGE MACROS

- DESIGN OF THE ENTRY BLOCK MACROS
  - DEFARG - Define calling parameters
  - DEFB - Assign names to B registers
  - DEFT - Assign names to T registers
  - ALLOC - Allocate space for local temporary variables
  - MXCALLEN - Declare maximum calling list length
  - PROGRAM - Generate mainline CAL routine start point
  - ENTER - Generate CFT-callable entry point
- RETRIEVE PASSED-IN ARGUMENT LIST INFORMATION MACROS
  - ARGADD - Fetch argument address
  - NUMARG - Get the number of arguments passed in
- REFERENCE LOCAL TEMPORARY VARIABLE STORAGE MACROS
  - LOAD - Get value from memory into a register
  - STORE - Store the value from a register into memory
  - VARADD - Return the address of a memory location
- CALL EXTERNAL ROUTINES MACROS
  - CALL - Call a routine using call-by-address sequence
  - CALLV - Call a routine using call-by-value sequence
- EXIT SUBROUTINE MACRO
  - EXIT - Terminate subroutine and return to caller

TABLE AND SEMAPHORE MANIPULATION

- TABLE DEFINITION AND CONSTRUCTION MACROS
  - Normal Macros
    - BUILD - Construct a table structure
    - ENDTABLE - Designate the end of a table definition
    - FIELD - Define a field with current table structure
    - NEXTWORD - Advance a specified number of words
    - REDEFINE - Redefine a specified number of words
    - SUBFIELD - Identify fields within a larger field
    - TABLE - Define the overall table attributes
  - Complex macros
    - CENDTAB - End a complex table structure
    - CFIELD - Define a field in the current complex table
    - CNXTWORD - Advance a specific number of 64-bit words
    - CREDEF - Redefine specific number of 64-bit words
    - CSBFIELD - Define field entirely within another field
    - CTABLE - Define overall table attributes
- PARTIAL-WORD MANIPULATION OPDEFS
  - Normal Opdefs
    - GET - Fetch contents of a field
    - GETF - Fetch contents of a field
    - PUT - Store data from a register into a field
    - SET - Pack field value into a register
    - SGET - Fetch contents of a field
    - SPUT - Store data from a register into a field
  - Complex Opdefs
    - CGET - Fetch contents of a field into a register
    - CPUT - Store contents of a register into a field
- SEMAPHORE MANIPULATION MACROS
  - DEFSM - Define semaphore name
  - CLRSM - Unconditionally clear a semaphore, do not wait
  - GETSM - Get current status of semaphore bit
  - SETSM - Unconditionally set a semaphore, do not wait
  - TEST$SET - Test semaphore and wait if set, set if clear
- CAL EXTENTION MACROS AND OPDEFS
  - DIVIDE OPDEF - Provide a precoded divide routine
  - PVEC MACRO - Pass elements of vector register to scalar routine
  - $CYCLES MACRO- Generate timing-related symbols and constants
  - $DECMIC MACRO - Convert a positive integer to a micro string
  - RECIPCON MACRO - Generate floating-point reciprocals

COS DEPENDENT MACROS

- SYSTEM TASK OPDEFS
  - ERDEF - Generate error processing entries in the Exchange Processor
  - GETDA - Obtain first DAT page address
  - GETNDA - Obtain next DAT page address
- OVERLAY MANAGER TASK MACROS
  - CALLOVL - Request Overlay Manager Task to load
  - DEFINOVL - Generate a list of modules
  - DISABLE - Prevent use of current memory-resident copy
  - GOTOOVL - Request Overlay Manager Task to load
  - LOADOVL - Request an initial overlay load
  - OVERLAY - Define a module as a system overlay
  - OVLDEF - Define overlay name
  - RTNOVL - Signal completion of an overlay execution
- MESSAGE PROCESSOR MACRO
  - LOGMSGM - Construct the LGR control word

UPDATE DIRECTIVES

Update SR-0013

BEFORE - INSERT BEFORE DIRECTIVE
CALL - CALL COMMON DECK DIRECTIVE
COMDECK - COMMON DECK DIRECTIVE
COMPILE - COMPILE DIRECTIVE
CWEOF - CONDITIONAL WRITE END-OF-FILE DIRECTIVE
DECK - DECK DIRECTIVE
DECLARE - DECLARE DECK FOR MOD APPLICATION DIRECTIVE
DELETE - DELETE CARDS DIRECTIVE
EDIT - EDIT DECKS DIRECTIVE
IDENT - MODIFICATION SET IDENTIFICATION DIRECTIVE
INSERT - INSERT AFTER DIRECTIVE
LIST AND NOLIST - RESUME/STOP LISTING DIRECTIVES
MOVEDK - MOVE DECK DIRECTIVE .
PURGEDK - REMOVE DECK DIRECTIVE
READ - READ ALTERNATE INPUT DIRECTIVE
SEQ AND NOSEQ - START/STOP SEQUENCE NUMBER WRITING
WEOF - WRITE END-OF-FILE DIRECTIVE -
/ - COMMENT DIRECTIVE
YANK AND UNYANK - YANK/UNYANK DECKS AND MODIFICATION

OPERATIONAL AIDS

Operational Aids SM-0044

UNB - Converts binary load modules
ADSTAPE - Builds IOP deadstart datasets
EXTRACT - Extracts messages from the system logfile
FDUMP - Formats control memory dump
STATS - Gathers mainframe performance statistics
JSCDEF - Defines a job class structure
PRVDEF - Defines permanent dataset privileges
ACCTDEF - Defines accounting entries
MODSET - Merges modifications into a set
SPAWN - Submits multiple jobs
STEP - Tests job steps
MODSEQ - Resequences mods
BIND - Resolves APML externals
SETOWN - Sets permanent dataset ownership

EXTRACT DIRECTIVES

SELECT
INPUT
OUTPUT
LINES
FLUSH
NOHEADER
DUMP
RAWDUMP
LEFT8 and RIGHT8
END
SUMMARY

FDUMP DIRECTIVES

CLASS 1 DIRECTIVES

FILES
DMEM
COMP
XCOMP
DSYM
AUTO
DSDT
DXTR
CPU
IOP
SETBIAS
DSSD

CLASS 2 DIRECTIVES

TITLE
SPACE
FLUSH

TYPES OF KERNEL COMMANDS
- Initialization commands
  - CRAY
  - STATION
  - MASTER
  - CONFIG
- Concentrator commands
  - Communication with CRI front-end interface
    - CONC
    - ENDCONC
  - Communication with an NSC A130 adapter
    - NSC
    - NSCEND
  - Interactive communication with COS
    - IAIOP
    - IAIOP LOG
    - IAIOP POLL
    - IAIOP LOGOFF
    - IAIOP END
    - IACON
- Miscellaneous maintenence commands
  - LISTP
  - LISTO
  - UBTAPE
  - PRTAPE
  - ERRDMP
  - ERROR
  - TIME
  - CLOCK

STATION COMMAND DESCRIPTIONS
- Command formats
- Station command summary
  - CHANNEL - Turn channel on or off
  - CLASS - Turn job classes on or off
  - CLEAR - Clear screen
  - COMMENT - Command stream comment
  - CONFIGURE - Alters tape or disk device configuration
  - CONSOLE - Allocate additional station console
  - DATASET - Display dataset status
  - DELAY - Suspend command processing
  - DEVICE - Change read-only status on mass storage
  - DISK - Display disk statistics
  - DROP - Drop job
  - END - End station operation
  - ENTER - Change job scheduling parameters
  - ERROR - Display hardware error information
  - FLUSH - Copy data to backup dataset
  - JOB - Display job status
  - KILL - Kill job
  - LIMIT - Limit number of jobs active
  - LINK - Link status display
  - LOGOFF - Log off station
  - LOGON - Log on station
  - MESSAGE - Enter message into logfile
  - MONITOR - Monitor system parameters
  - OPERATOR - Change master operator station
  - POLL - Set control message exchange rate
  - RECOVER - Recover system
  - REFRESH - Set display refresh rate
  - REPLY - Reply to station request message
  - RERUN - Rerun job
  - RESUME - Resume job processing
  - ROUTE - Change station ID
  - SAVE - Stage permanent dataset
  - SCROLL - Use display for command/response scroll area
  - SET - Modify parameters
  - SHUTDOWN - Shut down the system
  - SNAP - Print display contents
  - STAGE - Halt or resume staging
  - STATCLASS - Display job class status
  - STATION - Display I/O Subsystem station status
  - STATUS - Display system status
  - STMSG - Display station messages
  - STORAGE - Display mass storage status
  - STP - Display System Task Processor statistics
  - STREAM - Change stream counts
  - SUBMIT - Stage job dataset
  - SUSPEND - Suspend job processing
  - SWITCH - Manipulate job sense switches
  - TAPE - Display tape device information
  - TJOB - Diplay tape job's status

PSEUDO INSTRUCTIONS

- Program control
  - IDENT - Identify program module
  - END - End program module
  - ABS - Assemble absolute binary
  - COMMENT - Define program descriptor table comment
  - GLOBAL - Declare global symbols
- Code control
  - BASEREG - Declare base operand register
  - SCRATCH - Declare APML scratch register
  - NEWPAGE - Force a new instruction page
- Loader linkage
  - ENTRY - Specify entry symbols
  - EXT - Specify external symbols
  - START - Specify program entry
- Mode control
  - BASE - Declare base for numeric data
  - QUAL - Qualify symbols
- Block control
  - BLOCK - Local block assignment
  - ORG - Set *O counter
  - BSS - Block save
  - LOC - Set * counter
  - BITW - Set *W counter
  - BITP - Set *P counter
- Error control
  - ERROR - Unconditional error generation
  - ERRIF - Conditional error generation
- Listing control
  - LIST - List control
  - SPACE - List blank lines
  - EJECT - Begin new page
  - TITLE - Specify listing title
  - SUBTITLE - Specify listing subtitle
  - TEXT - Begin global text
  - ENDTEXT - Terminate global text
- Symbol definition
  - EQUALS - Equate symbol
  - SET - Set symbol
  - CHANNEL - Channel symbol
  - MICSIZE - Set redefinable symbol to micro size
- Data definition
  - CON - Generate constant
  - BSSZ - Generate zeroed block
  - DATA - Generate data words
  - PDATA - Generate data parcels
  - VWD - Variable word definition
- Conditional assembly
  - IFA - Test expression attribute for assembly condition
  - IFE - Test expressions for assembly condition
  - IFC - Test character strings for assembly condition
  - SKIP - Unconditionally skip statements
  - ENDIF - End conditional code sequence
  - ELSE - Toggle assembly condition
- Instruction definition
  - MACRO - Macro definition
  - LOCAL - Specify local symbols
  - ENDM - End macro definition
  - OPSYN - Synonymous operation
- Code duplication
  - DUP - Duplicate code
  - ECHO - Duplicate code with varying arguments
  - ENDDUP - End duplicated code
  - STOPDUP - Stop duplication
- Micros
  - MICRO - Micro definition
  - OCTMIC and DECMIC - Octal and decimal micros
  - Predefined micros