TEXAS INSTRUMENTS

Improving Man's Effectiveness Through Electronics

DX 980
General Purpose Operating System

Programmer's Guide

MANUAL NO. 943005-9701
ORIGINAL ISSUE 1 OCTOBER 1974
REVISED AND REISSUED 15 MAY 1975

Change 1

1 August 1975

Digital Systems Division

LIST OF EFFECTIVE PAGES

INSERT LATEST CHANGED PAGES DESTROY SUPERSEDED PAGES

Note: The portion of the text affected by the changes is indicated by a vertical bar in the outer margins of the page.

DX980 General Purpose Operating System Programmer's Guide (943005-9701)

Original Issue . . . . 1 October 1974
Revised and Reissued . . . . 15 May 1975 (ECN 388288)
Change 1 . . . . 1 August 1975 (ECN 402621)

Total number of pages in this publication is 287 consisting of the following:

PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.
Cover	1	6-3	0	8-77 — 8-82B	1
Eff. Pages	1	6-4	1	8-83	0
v — xiv	1	6-5 — 6-8	0	8-84	1
1-1 — 1-10	0	7-1 — 7-16	1	App. A Div	0
2-1 — 2-21	0	8-1 — 8-3	1	A-1	1
2-22 — 2-34	1	8-4 — 8-8	0	A-2 — A-4	0
3-1 — 3-17	0	8-9	1	App. B Div	0
3-18 — 3-36	1	8-10 — 8-11	0	B-1 — B-5	0
4-1 — 4-3	0	8-12	1	B-6	1
4-4 — 4-5	1	8-13	0	B-7 — B-10	0
4-6 — 4-12	0	8-14	1	B-11	1
4-13 — 4-14	1	8-15	0	B-12 — B-18	0
4-15 — 4-22	0	8-16	1	App. C Div	0
5-1 — 5-2	0	8-17	0	C-1 — C-20	0
5-3	1	8-18 — 8-20	1	App. D Div	0
5-4 — 5-10	0	8-21	0	D-1 — D-72	0
5-11 — 5-18	1	8-22	1	D-73	1
5-19 — 5-24	0	8-23 — 8-64	0	D-74 — D-76	0
5-25 — 5-33B	1	8-65 — 8-66	1	Alphabetical Index Div	0
5-34 — 5-38	0	8-67 — 8-69	0		
6-1	0	8-70	1	Index-1	0
6-2	1	8-71 — 8-76	0	Index-2	1

LIST OF EFFECTIVE PAGES

INSERT LATEST CHANGED PAGES DESTROY SUPERSEDED PAGES

Note: The portion of the text affected by the changes is indicated by a vertical bar in the outer margins of the page.

Total number of pages in this publication is consisting of the following:

PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.
Index-3	0				
Index-4	1				
Index-5	0				
Index-6	1				
Index-7	0				
Index-8	1				
User's Resp . . .	1				
Bus. Reply	0				
Cover Blank . . .	0				
Back Cover . . .	1				

PREFACE

This manual provides information concerning the Texas Instruments DX980 General Purpose Operating System for use by programming personnel working with the system. The information is divided into eight sections and four appendixes as follows:

I. INTRODUCTION - This section describes the operating system and its components to provide an overview of its capabilities.

II. JOB CONTROL LANGUAGE - This section introduces the Job Control Language used in the system and details the required parameters and formats of JCL commands.

III. INPUT/OUTPUT STRUCTURE - This section explains the organization of I/O handling routines and lists the I/O Supervisor Calls.

IV. DISC FILE MANAGEMENT - This section describes the storage of data in disc files and how the operating system controls that operation.

V. SUPERVISOR CALLS - This section lists and describes the supervisor calls available for programmer use.

VI. BATCH PROCESSING SUBSYSTEM - This section explains the three component subsystems that handle batch processing for the system.

VII. INTERACTIVE TERMINAL SUBSYSTEM - This section describes the operation of the subsystem that provides interactive communication between the terminal users and the operating system.

VIII. UTILITIES - This section describes the characteristics of the utility programs available for use with the operating system.

A. ERROR MESSAGES - This appendix lists DX980 error messages and definitions.

B. RECOMMENDED JCL SEQUENCES - This appendix provides sample listings of JCL sequences for the operating system.

C. ADDING TO ITS - This appendix provides detailed descriptions of the Interactive Terminal Subsystem as an aid to adding new application programs to run under the subsystem.

D. ADDING NON-STANDARD DEVICES TO DX980 - This appendix describes the procedure for designing a device service routine to service a device not normally supplied with the operating system.

An alphabetical index of key phrases also appears at the back of this manual. In addition to this manual the user should also have access to the following manuals that are referenced as applicable within the text of this manual:

DX980 General Purpose Operating System, System Operator's Guide, Part Number 943004-9701

Model 980 Computer, Basic System Use and Operation, Part Number 961961-9710

Model 980 Computer, Terminal User's Guide, Engineering Data, Part Number 943010-9701

Model 980 Computer, Terminal User's Guide, Model 733 ASR/KSR Data Terminal, Part Number 943009-9701

Model 980 Computer, Terminal User's Guide, Model 912 Video Display Terminal, Part Number 943014-9701

Model 960 Computer and Model 980 Computer Debug User's Guide and Operating Instructions, Part Number 942760-9701

Model 980 Computer FORTRAN, Part Number 944800-9701

Model 980 Computer Assembly Language Input/Output, Part Number 961961-9734

Model 980 Computer Assembly Language Programmer's Reference Manual, Part Number 943013-9701

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

APPENDIXES

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

SECTION I INTRODUCTION

1.1 GENERAL

DX980 is a general purpose operating system that supports the Texas Instruments 980 Series minicomputers with major features that include the following:

- Interrupt driven device handlers
- Multiprogramming with biased addressing and protected memory bounds
- Flexible job management with complete resource allocation
- Comprehensive data management
- Dynamic memory management
- Priority scheduling of jobs and of tasks within jobs
- Subsystems to support batch and interactive terminal processing

These system attributes augment computer performance for application in the following areas:

- Interactive information management systems that allow terminal devices to enter, retrieve and display data stored in an online disc information base.
- High speed, real-time computing systems that provide both analog and digital data acquisition, and that include control loops that require the computational capacity of a 980 computer.
- Batch processing environments for program development, scientific data processing and business applications.
- Combinations of the above systems for specialized applications.

1.2 HARDWARE REQUIREMENTS

The operating system nucleus occupies approximately 24,000 words of main memory and requires the following additional system hardware:

- Additional memory for support of subsystems, language processors, and application programs.
- Either a moving head disc or a DS330 disc system.
- Magnetic tape drive to initially load the disc, to allow a dump of the disc contents for reloading, and to provide offline storage.

- Operator console for system interaction with the user. (Model 733 ASR terminal or equivalent)
- Interval timer.

1.3 HARDWARE EXPANSION

In addition to the required hardware for operation of the system, DX980 supports a full 64K word memory computer system. The modular construction of the system allows easy addition of new peripheral devices and their related service routines. Standard device service routines include support for one or more of each of the following peripherals:

- Texas Instruments Model 733 ASR Terminal, Part Number 966645-0003
- Texas Instruments Model 733 KSR Terminal, TI Part Number 966671-0003
- Teletype Model 33 ASR Teletypewriter, TI Part Number 973346-0001
- Texas Instruments Model 912 CRT display unit, TI Part Number 973306-0012
- High Speed Paper Tape Reader/Punch Combination, TI Part Number 973526-0003
- High Speed Paper Tape Reader, TI Part Number 965946-0003
- 132 Column I/O Bus Line Printer, TI Part Numbers 966792-0001 and 966792-0011
- 80 Column DMAC Line Printer, TI Part Number 217065-0001
- Card Reader, TI Part Number 966323-0003
- Analog-to-digital and digital-to-analog conversion equipment
- DS330 Disc Controller and Disc Drive System, TI Part Numbers 942541-0002, 942541-0003, 942541-0004, 942541-0005, and 942541-0006.
- Moving Head Disc, TI Part Number 955157-0001
- Texas Instruments Model 979 Tape Transport, TI Part Number 217536-0001

1.4 SYSTEM STRUCTURE

The operating system is divided into a nucleus for executive functions common to most operations, and several subsystems that perform specific functions. In addition to the nucleus, two subsystem modules are supplied with the standard operating system: the Interactive Terminal Subsystem (ITS) and the Batch Processing Subsystem (BPS). Figure 1-1 illustrates the relationship

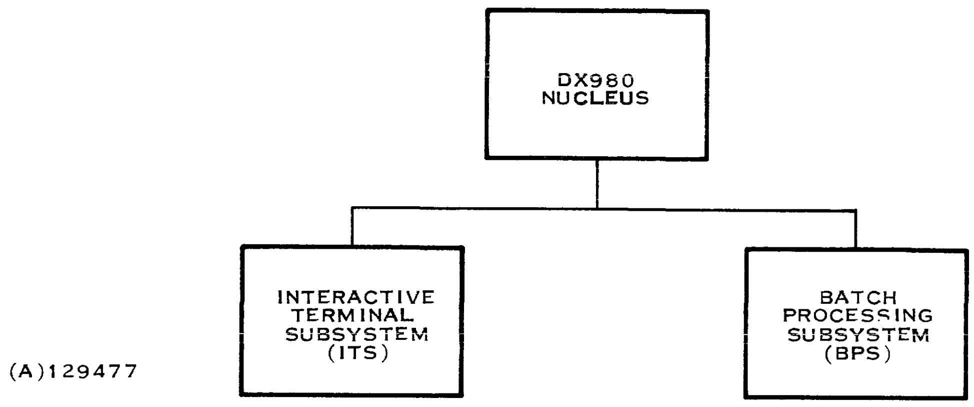

DX980 NUCLEUS
INTERACTIVE TERMINAL SUBSYSTEM (ITS)
BATCH PROCESSING SUBSYSTEM (BPS)
(A)129477

Figure 1-1. DX980 General Structure

of these subsystems to the nucleus. Other subsystems may also be added to perform functions suited to special applications. The following paragraphs describe the characteristics of the standard system components.

1.4.1 NUCLEUS

The nucleus is the portion of the operating system that defines the system parameters, handles input and output servicing, and provides real time reaction to the subsystems. The nucleus is divided into four main parts: the system table, the memory resident code, the procedure pool or system overlay area, and a Dynamic System Control Area (DSCA). The real time features of the operating system are integrated into these areas and do not form a separate subsystem as do the Interactive Terminal Subsystem and the Batch Processing Subsystem. Real time features include rollin and rollout, task synchronization supervisor calls, dynamic device allocation, the ability to start other jobs via a supervisor call, and a method for controlling file updating with multiple online users. Most of the code for these features occupies main memory only when in direct use and runs in the procedure pool.

1.4.1.1 SYSTEM TABLE. The System Table describes the operating parameters of the system including: system job parameters, system wide control blocks, and pointers to system queues, the DSCA, procedure pool and free area. The table is created during system generation.

1.4.1.2 MEMORY RESIDENT CODE. Memory resident code is that portion of the operating system that is always in main memory. This code performs the following functions in the DX980 system: interrupt processing, system disc device handling, task management, preliminary supervisor call processing, memory management, overlay management, and commonly used lower level system subroutines.

1.4.1.3 PROCEDURE POOL. The procedure pool is a dynamically allocated memory area used for system overlays. Approximately 80% of the system routines run in this area. These routines include: device service routines not handled by the memory resident code, job management, file management, operator communications, and modules of rollin and rollout. The user can select the size of the procedure pool to adapt the amount of memory available to the operating speed required.

1.4.1.4 DYNAMIC SYSTEM CONTROL AREA (DSCA). The DSCA is a volatile memory pool that is used by the procedure pool routines. Most of these routines are written in reentrant code. The DSCA provides a designated work area for each task executing a procedure pool routine. The user can select the size of the DSCA.

1.4.2 INTERACTIVE TERMINAL SUBSYSTEM (ITS)

The Interactive Terminal Subsystem (ITS) provides simplified concurrent communications with several terminals. Included in ITS is the ITS Supervisor (ITSUPV) and the Interactive File Editor (IFE). ITSUPV manages the terminals, and provides LOGON and LOGOFF functions plus the required housekeeping functions for multi-terminal usage. ITSUPV interfaces easily with user written application programs. IFE provides interactive generation and editing of disc files containing source text.

The ITS Supervisor initially checks the system's logical and physical device tables to derive a list of the terminals with which it can communicate (polling list). This list is based on the logical units assigned to the ITS at job start-up time. The ITS then assigns to each of the terminals on the list:

- A buffer for data exchange with the terminal
- A User Control Block

The User Control Block contains pointers, flags, and a scratch pad memory space for use by application programs operating under ITS. One key parameter in the control block indicates which body of code, or application program, is currently attached to the terminal. A single Command Table is available to ITSUPV for all terminals. The Command Table contains the names and entry points of the programs that can be run under ITS. After logging on, the terminal operator, through the ITS Supervisor, can select one of the programs in the Command Table for execution. Similarly, after completion of that program the operator can select another program.

The ITS uses a Branch and Link command to pass the data exchange buffer to the selected application program. That program then processes the input and places any response to the terminal in the data exchange buffer. When the application program returns control to the ITS, the program may instruct the ITS to do any of the following operations:

- Write the contents of the data exchange buffer on the corresponding terminal and return to the application program for more processing.
- Write the contents of the data exchange buffer on the corresponding terminal, acquire a response from the terminal, and return for more processing.

- Read input from the terminal and return for more processing.
- Return for more processing as time becomes available without intervening I/O operations.
- Discontinue use of this application program by the terminal and indicate readiness for the operator to select a new application program.

1.4.3 BATCH PROCESSING SUBSYSTEM (BPS)

The Batch Processing Subsystem (BPS) consists of three data handling subsystems that allow convenient batch input and output for the operating system. The three data handling subsystems are Batch Input Reader (BIR), Batch Input Spooler (BIS) and Batch Output Spooler (BOS). Any of these subsystems can be activated from the system console by addressing a RUN command to the proper subsystem.

1.4.3.1 BATCH INPUT READER (BIR). Specifying BIR causes the operating system to activate the assigned card reader and read input directly from that device.

1.4.3.2 BATCH INPUT SPOOLER (BIS). Specifying BIS instructs the operating system to activate the assigned card reader, read input continuously from that device, and copy the input data to a disc file (spooling). Processing can then proceed via the disc file (SYSIN) in parallel with the input operation to increase throughput.

1.4.3.3 BATCH OUTPUT SPOOLER (BOS). Specifying BOS instructs the operating system to access from the disc all data assigned to unit SYSOUT. The data is read from the disc and printed on the system output device (usually a line printer). This feature allows data to be spooled on the disc during execution of a job by specifying the output unit as SYSOUT. When the output device becomes available, the operator can then activate BOS to dump the spooled data on that device. Use of BOS in this manner significantly increases throughput.

1.5 INPUT/OUTPUT MANAGEMENT

User programs running under DX980 select I/O devices with a logical unit number (LUN) rather than a physical device number. A statement within the JCL input for the job equates the LUN to a physical device name, while a physical device table for the operating system specifies an actual I/O device to respond to the physical device name. This arrangement allows the I/O device assignments within the operating system and the logical unit numbers within the user program to remain constant. A simple change to the JCL input allows the user to change the I/O medium. The operating system includes a Device Service Routine (DSR) for each I/O device supported by the monitor. Each DSR translates the common program I/O interface to the unique I/O interface required by its type of I/O device. The physical device table ties a particular I/O device to the DSR that services it. Additional DSRs can be added to the system to support new I/O devices without serious change to the system.

1.6 MEMORY MANAGEMENT

All of memory that is not occupied by the resident operating system, system tables, and the procedure pool is designated as a free area. The operating system dynamically allocates this area to subsystems and to user jobs. When a program is loaded into memory, the operating system stores it in contiguous locations in a single partition within memory. It then loads the upper and lower bounds of that partition into the hardware memory protect address registers of the memory controller, and enables the hardware Priviledged Instruction Feature. All program parameters can then be referenced with a relative address with respect to the lower bound of the memory partition since the hardware automatically biases the addresses with the lower bound address. The operating system allocates memory partitions to particular jobs on a best fit basis.

1.7 PROGRAM HANDLING

Programs run under DX980 are performed as jobs; each job is further divided into tasks. The following paragraphs explain these concepts with respect to the operating system.

1.7.1 JOBS

A job is a unit of work in the DX980 system. A job consists of a set of user tasks to be executed and the control information to communicate resource requirements to the system. The system runs each job when all required resources become available. When a job is complete, all allocated resources are returned to the system. They may then be allocated to another job as necessary. If the system has adequate resources, more than one job can be active at one instant. Each job competes for execution time on a priority basis.

Jobs can be submitted to DX980 through the Batch Processing Subsystem (BPS), the Interactive Terminal Subsystem (ITS), the system console, or from another user job. The job submission procedure is the same regardless of source.

Once DX980 accepts the control information required to start a job, it places the information in a priority queue, called the job queue. Part of the job queue resides in main memory and part resides in a random access file on the system disc. As resources become available from jobs currently in execution, either through job termination or a specific request to release one or more resources, DX980 determines whether the additional resources are sufficient to execute one of the jobs in the job queue. If so, the system changes that job's state from Ready to Running and starts the job.

Each job in DX980 is requested with Job Control Language (JCL), and communicates with the operating system using supervisor calls. A series of jobs may be combined into related job steps to perform a larger function, such as Compile, Link and Execute. In addition, a job may subdivide itself into two or more concurrently resident tasks. All user jobs run in protected mode and must remain within the memory bounds defined for that job.

1.7.2 TASKS

A task is a currently active program that coexists in memory with other concurrent tasks, all of which may be in various stages of completion. One or more tasks combine to form a job. Tasks within a job may execute concurrently and share the same reentrant body of code though each task is in a different state of execution. All jobs initiate as a single task. The initial task then issues supervisor calls to activate other tasks within its job area. Each separate task has its execution environment (register contents) and receives a relative priority rank with other concurrent tasks.

1.8 FILES

A file is an organized collection of information. The file is divided into records that may be held in main memory while being used, but are stored on a disc when not being used. The following paragraphs describe the organization of files, the types of files available with the operating system, and the file handling features of the operating system.

1.8.1 FILE ORGANIZATION

Figure 1-2 illustrates the organization of files within a single disc unit for the DX980 operating system. The system locates a specific file with a three parameter entry supplied by the user. The <volume> parameter specifies the disc drive unit that contains the file, and allows the system to access the Master File Directory for that disc unit. The Master File Directory lists all of the User Directories that are stored on that disc unit, and the physical disc addresses used to locate each directory. A second parameter, <fileid>, tells the operating system which of the User Directories contains the file. The system then calls up the proper User Directory, which lists all of the files on the disc unit for that user, and their location on the disc. A third parameter, <filnam>, tells the operating system which of these files to access for the required information.

Volume number 1 is always the system disc unit containing all of the operating system files. Therefore, systems with only one disc unit must label that disc as volume 1. Within the Master File Directory for the system disc is an entry for the SYSTEM User Directory. This directory lists all of the files that are part of the operating system. One of these files, SJCBFL, contains the Job Control Language (output of the JCL Translator program) for the system. Another file, JCLSRC, is the JCL source file.

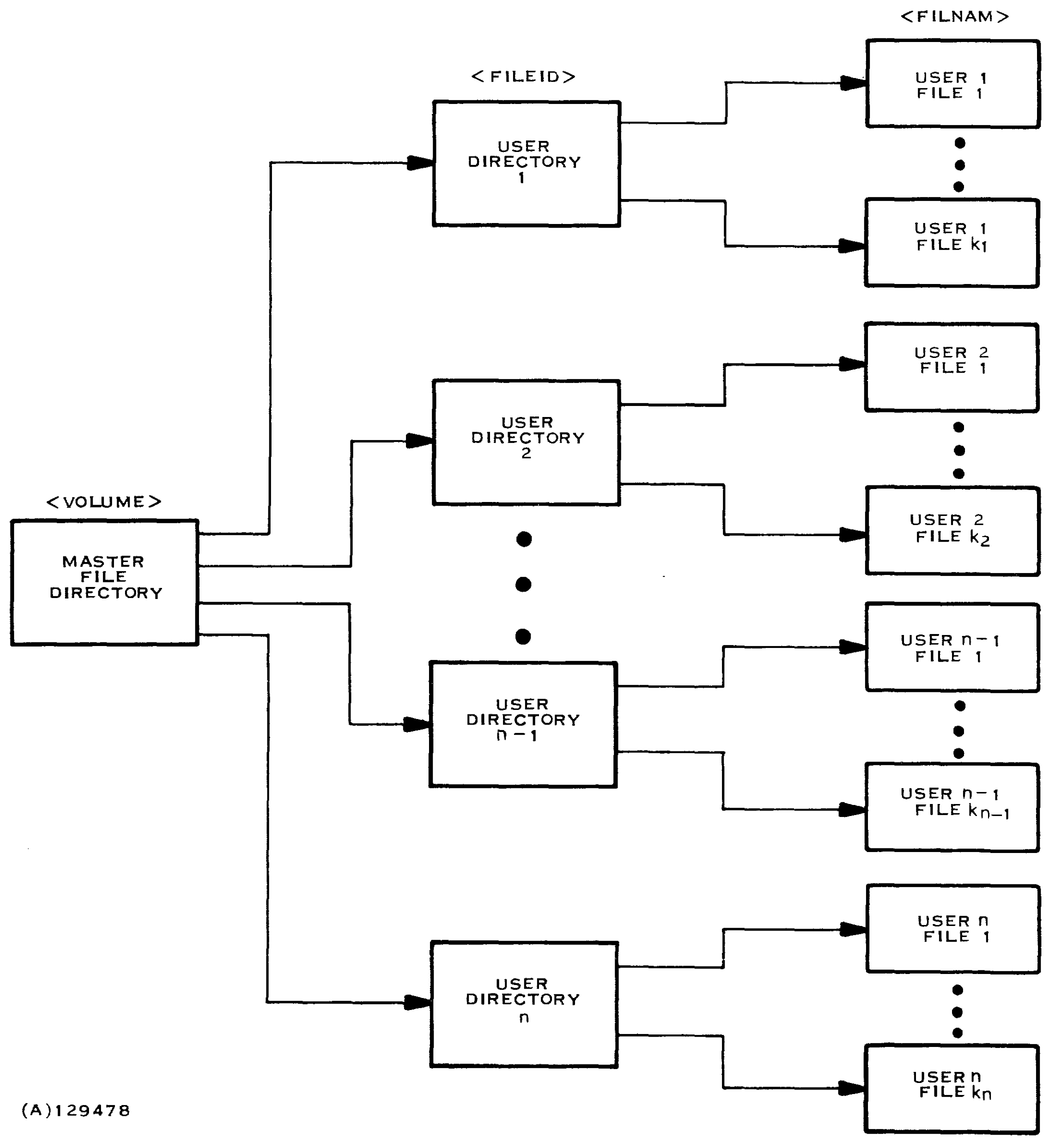


Figure 1-2. Organization of Data Files within a Disc Drive

1.8.2 FILE HANDLING

The operating system provides the user with the following features when working with files:

- File protection through both creator access and password entry; for either type of protection the user can specify whether it will be applicable for reading, writing or deleting, or any combination of those functions.
- Maintenance of several User Directories, so that files can be segregated in a multi-user environment.

- File mapping performed by the system can also be assisted by a user request for a specific physical location to store a file.
- Specification of exclusive files for use by one job only, or of shared access files for concurrent use by more than one job.
- Momentary cooperative locking of shared files during update to prevent degradation; this feature prevents one job from destroying a file update made by another job within an interactive information management system.
- Blocking of multiple logical records into physical records for data transfer and storage; these blocked records are built and disassembled in buffer areas within the job extension area of main memory.

1.8.3 TYPES OF FILES

The operating system supports three types of files as outlined in the following statements:

- Linked Sequential File - The records of this file are scattered throughout the storage medium, but are chained together by pointers within the file. This arrangement requires that the record be accessed in sequential order, but also allows the file to grow dynamically like a magnetic tape file.
- Relative Record File - This type of file contains records of a specified length that can be accessed randomly within the file by their relative position to the beginning of the file. Records are allocated in contiguous storage space.
- Key Indexed File - This type of file assigns a key to each record that is to be accessed in a random fashion, and sorts the keys in a tree structure for efficient location of each record. The keyed records can then be stored in non-contiguous areas, allowing the file to grow dynamically. A key indexed file may also include non-keyed records that are accessed sequentially with keyed records.

SECTION II
JOB CONTROL LANGUAGE

2.1 GENERAL

Job Control Language (JCL) specifies the structure and resource requirements of jobs submitted to DX980. Other commands, used to request special action from DX980, are described in the DX980 General Purpose Operating System, System Operation Guide under the title of Operator Communications.

2.2 JOB CONTROL LANGUAGE STRUCTURE

Each job submitted to DX980 consists of one or more independent job steps that are executed in the order in which they appear in the job stream. For example, if two job steps are submitted, they will be executed in the order of step one and then step two. If any step aborts because of an error condition, all subsequent steps will not be executed.

Job steps would normally be combined in such a way that the successful execution of each step depends on the successful completion of all preceding steps. A common example is a sequence of four steps to compile (2 phases), link and execute a FORTRAN program. The link step would be started only if the compilation was successful, and the execution step would be started only if both the compilation and link were successful.

Within the JCL statements for each step are assignment commands to associate Logical Unit Numbers (LUNs) specified in the user program with physical devices or files. These commands allow the user to specify Input/Output references logically when a program is created and defer the actual device assignments until the program is run (runtime). For most sequential devices the assignment command merely links a LUN to the device. For a disc file, however, the user must specify the type of file to be assigned, whether the file is new, old or temporary, blocksize of the data records, and disposition of the file after termination of the job step.

2.3 JCL PROCESSING

There are two forms of Job Control Language (JCL): expanded JCL and abbreviated JCL.

NOTE

In the following discussion and throughout the manual, the term "card image" is equivalent to "source line image" and is applicable to other input devices as well as a card reader.

2.3.1 EXPANDED JCL

Expanded JCL can specify the structure of individual job steps, the location of the object program (load module) for each step, and the resources required for each step. These specifications would normally be prepared once for each type of job (e.g., FORTRAN compile, link, and execute) and saved on a disc file for subsequent retrieval. Saving the JCL in card image form requires reprocessing each time it is retrieved. To avoid this reprocessing, DX980 saves the JCL in a binary form by passing the card images through a program that produces a savable binary image. Thereafter, the binary images can be processed on subsequent retrievals. The utility program that translates the card images to binary is called the JCL translator.

Expanded JCL is not recognizable to the DX980 operating system, but only to the JCL translator program that runs as a job under DX980. Expanded JCL is the input to the JCL translator.

2.3.2 ABBREVIATED JCL

Abbreviated JCL, the mechanism for job submittal to DX980, is "cookbook" oriented. It allows a user to run a FORTRAN program by directing the operating system to "RUN FORTRAN". The user does not have to specify the detailed job step structure and resource assignments required to invoke the two passes of the compiler, to activate the link editor and finally to execute the program itself. This form must have a binary image of the expanded JCL saved on a disc file under the appropriate name, i.e., "FORTRAN". That is, the expanded JCL must be prepared and processed for the FORTRAN example. However, this can be done by a systems programmer, leaving the applications programmer to debug his FORTRAN program without having to learn all ideosyncrasies of an expanded JCL.

In addition to invoking an existing JCL sequence (a sequence of expanded JCL statements) by name only, abbreviated JCL allows modification of one or more JCL specifications at runtime. For example, the original JCL specifications might have assumed that a FORTRAN source program would be supplied on cards when in fact the source for a particular program is stored on a disc file. To satisfy this requirement, the user can specify at the time the expanded JCL is prepared that any given JCL parameter is either:

(1) completely specified and cannot be modified at runtime,

(2) completely specified but can be modified at runtime, or

(3) not specified at all and must be supplied at runtime.

With this facility, the user of a JCL sequence can decide that the original specifications or defaults are acceptable and invoke the sequence by name only, or he can decide that modifications are required and can specify the parameters to be modified. Abbreviated JCL is processed by the DX980 subsystems or by the DX980 operator communications facility via the system console.

2.3.3 JCL TRANSLATOR

The JCL translator is a utility program that runs in user memory (as opposed to running in memory that is allocated to the operating system). This program translates expanded JCL statements into an internal binary representation for later processing. Input to the translator can be supplied from any of the standard 980 input devices (card reader, cassette, magnetic tape, disc files, etc.). Output from the translator can be stored on a disc file for later submission. Since the translator is a utility program, it can be invoked as any other user program by using abbreviated JCL form. The saved JCL sequence for the translator is stored on the system disc together with several other sequences that are supplied with each DX980 installation.

2.3.4 DX980 SUBSYSTEMS

In any particular DX980 installation one or more of the subsystems may be active for submission of user jobs. For a system with a card reader the Batch Processing Subsystem is active and user jobs can be submitted from the card reader. For systems with several terminals attached, the Interactive Terminal Subsystem is normally active to allow jobs to be submitted from any of the terminals. In any case, jobs can be submitted from the system console.

Job submission from each of these sources is consistent with the cookbook JCL approach. The JCL sequence produced by the translator is retrieved from a disc file, modified according to the user's specifications, and submitted to the operating system for execution.

2.4 JOB SUBMITTAL

Job submittal commands in DX980 are separated into two categories: JOB commands and RUN commands. The JOB command is required for a job submission and identifies the user and the JCL sequence file to be used for that particular job. The RUN command represents abbreviated JCL and invokes a particular JCL sequence from the JCL file specified in the JOB command. When submitting a job via the system console or the Interactive Terminal Subsystem, only one JOB command is required. A job submitted for the Batch Processing Subsystem requires a JOB command for each RUN command. The identifier, //, must appear as the first two characters of a source line image for both JOB and RUN commands. The system console supplies this identifier automatically; however, it must be specifically supplied when submitting jobs under the Batch Processing Subsystem. The Interactive Terminal Subsystem does not require the double slash identifier. The following conventions will be adhered to for all JCL descriptions throughout this manual:

- Brackets [ ] enclose optional fields.
- Operands may be separated by either blank(s) or a single comma with or without intervening blanks. Successive commas may be

used to indicate a null operand. Both comma (,) and the symbol, ƀ, denote specified field separations.

- All numerical value parameters must be either decimal (base ten) or hexadecimal (base sixteen). A numeric parameter is assumed to be base ten unless it is immediately preceded by a "greater than" symbol (>) to denote a hexadecimal value.
- The first character of a keyword or operand must be alphabetic and the remaining characters may be either alphabetic or numeric characters intermixed in any order unless specifically noted otherwise.
- A period terminates a job control command or expanded JCL specification. Comments may be written after the period. If no comments are written, the period is optional.
- If the number of operands in a command exceeds the limits of a single record, the command operands can be continued in succeeding records by terminating each incomplete record with a semi-colon (;). Continuation records must begin past the first two spaces of the record, since the first two spaces will be ignored by the computer. If the input is on the system console, the computer automatically fills these two positions with two periods (..) following a record terminated with a semi-colon. Operands may then be entered immediately following the two periods. The number of continuation records is unlimited. This option can be used with both expanded and abbreviated JCL commands.
- Items shown enclosed in angle brackets (< >) are variables that must be supplied by the user when submitting the command to DX980.

2.4.1 JOB COMMAND (JOB)

All jobs submitted to DX980 must be preceded by a JOB command. JOB identifies the user submitting a job, the name of the job, and the file where the JCL sequence to be executed is located. Two acceptable forms of JOB commands are as follows:

```
//JOB ƀ<jsname>ƀ<userid>

//JOB ƀ <jsname>ƀ <userid>ƀ FILE=(<volume>,<fileid>,<filnam>[,<pswd>])
```

In the first example containing no FILE specification, DX980 defaults to the System JCS file [FILE=(1, SYSTEM, SJCBFL, AB).].

2.4.1.1 JOB KEYWORD.

JOB is the keyword that identifies the remainder of a record as a JOB command. JOB must appear in positions three through six of the input record. Following the JOB keyword are the command operands. These operands are position sensitive and must be separated by one or more blanks, or by a single comma with or without intervening blanks.

2.4.1.2 <jsname> OPERAND. The nomenclature, <jsname> refers to a name that is from one to six characters long. This name remains with the job throughout execution and represents the job name.

2.4.1.3 <userid> OPERAND. The nomenclature <userid> refers to a name that is from one to six characters long and that identifies the user to the DX980 system. The <userid> operand appears only in the JOB command. This operand identifies the disc directory that points to the disc files owned by the user. Each directory is itself a disc file that contains information about each of the files that belong to it. A utility program, CATLOG, that is described later in this manual, creates and destroys the directories. When created, each directory is given a unique name, represented by the term <fileid>. A file may also be assigned an access restriction at the time it is first defined. If access to the file is restricted to only the originator (CREATOR) of the file, then the <userid> in the JOB command must match the <fileid> of the target file before access to the file is granted.

2.4.1.4 FILE SPECIFICATION. FILE specification identifies the JCL sequence (JCS) file for a particular job. Each DX980 installation is supplied with one file of commonly used JCL sequences referred to as the system JCS file. In addition, users may create their own JCS files for particular applications. In either case the FILE specification directs the system to the appropriate file. If the FILE specification is not explicitly provided, the FILE specification defaults to the system JCS file [FILE=(1, SYSTEM, SJCBFL, AB)]. The following paragraphs explain the parameters of the FILE specification.

<volume> Operand. The nomenclature <volume> represents an integer number that specifies the disc unit containing the JCS file. The DX980 system disc is always disc number 1, and always contains the system JCS file. Additional disc units are labeled 2 through n ($n \leq 20$), and can be specified at Initial Program Loading (IPL).

<fileid> Operand. The nomenclature <fileid> refers to a name that is from one to six characters long, and that specifies the file directory containing the JCS file. For the system JCS file this operand should be specified as SYSTEM . For a user JCS file this operand must correspond to the <fileid> that was specified when the JCS file was created.

<filnam> Operand. The nomenclature <filnam> refers to a name that is from one to six characters long and that designates the file within <fileid> where the JCL sequences are stored. The file name for the system JCS file is SJCBFL. For a user JCS file this operand must correspond to the <filnam> that was specified when the file was created.

<pswd> Operand. The nomenclature <pswd> represents a name that is from one to four characters long and that is required only if the JCS file was designated for password protection when it was created. Normally JCS files would not be protected from reading (the process of acquiring a JCL sequence from a file is a read operation) and this operand is not required. However, if the file is password protected from reading, this operand is required and must match the password that was specified when the file was created. The file, SJCBFL, is not password protected from reading.

2.4.1.5 JOB EXAMPLES. The following examples illustrate the use of the JOB command.

(1)	//JOB NAME1 SYSTEM	The job with job name "NAME1" will be run under the userid "SYSTEM". Defaults on the three remaining JOB parameters are "1" for <volume>, "SYSTEM" for <fileid> and "SJCBFL" (acronym for System JCL File) for <filnam>.
(2)	//JOB NAME2 USER01	The job with job name "NAME2" will run under the userid "USER01" The three default parameters are the same as example one.
(3)	//JOB NAME3 SYSTEM FILE=(1, USER01, JCLFIL)	The job with job name "NAME3" will be run under the userid "SYSTEM". The JCL sequence will be retrieved from the file name "JCLFIL" under <fileid> "USER01" on disc number one. JCLFIL must have been previously created or the job will abort with a JCL error.

2.4.2 RUN COMMAND (RUN)

The RUN command is the control statement interpreted by DX980 subsystems and operator communications as a request to submit a job. This command and its operands comprise the abbreviated JCL for DX980 and must be supplied for each job submission. Only one RUN command is permitted within a JOB command. Thus each job submitted to DX980 requires a single JOB command followed immediately by a RUN command.

The format of the RUN command is as follows:

//RUNƀ<jcsnam>ƀ[<k_1=p_1>ƀ<k_2=p_2>. . .<k_n=p_n>]

2.4.2.1 RUN KEYWORD. The RUN keyword, following the command delimiter (//), identifies the remainder of the record as a RUN command. This format is consistent whether the job is submitted via the Batch Processing Subsystem (BPS), or the operators console. If submitted under the Interactive Terminal Subsystem (ITS), the command delimiter does not precede the RUN keyword.

2.4.2.2 <jcsnam>OPERAND. The nomenclature <jcsnam> refers to a name that is from one to six characters long and that represents a JCL sequence within the JCS file specified in the JOB command. This operand allows several JCL sequences to be stored in the same file with a unique name for each sequence. This operand is an extension of the FILE specification on the JOB command in that the FILE specification directs the system to the appropriate JCS file and <jcsnam> specifies the JCL sequence to be invoked.

2.4.2.3 USER DEFINED KEYWORDS (k_1, k_2, ..., k_n). The user defined keywords in the RUN command permit modification of JCL parameters at runtime. Changing parameters is frequently required when the default specifications for one or more resource assignments do not match the requirements for a particular job submission. When expanded JCL is prepared for input to the translator, the individual parameter specifications may include a user defined keyword which allows those parameters to be overridden. Selection of the keywords is at the discretion of the programmer preparing the JCL sequence. For example, a user defined keyword for specifying the master data file input to a payroll program might be specified as "MASTFL" in one environment and as "PAYFIL" in another.

All user defined keywords supplied in the RUN command must match the corresponding keywords in the expanded JCL. Unused keywords in the expanded JCL assume their default values. If no default was specified, a JCL error results.

2.4.2.4 PARAMETERS FOR USER DEFINED KEYWORDS. The nomenclature "p_1 p_2 ... p_n" represents either numeric or mnemonic values to be substituted into the JCL sequence <jcsnam> that the system retrieves from the <filnam> file. The size of the numeric values must fit within the limits of the parameter being replaced. For example, if priority were the variable being specified, "p" must be between 1 and 31.

2.4.2.5 RUN EXAMPLES. The following examples illustrate the use of the RUN command to invoke the JCL translator. A list of the expanded JCL for the translator is included in Appendix B of this manual.

(1) //RUN JCL	"JCL" is the <jcsnam> for the JCL translator. This sequence is stored in "SJCBFL" under the fileid "SYSTEM". This example

invokes the JCL translator with standard default assignments for input and output. The standard defaults are listed in the JCLTRN JCS in Appendix B of this manual.

(2) //RUN JCL DSRC=CR2, DERR=LP1,DLST=LP1

Run the JCL translator with override parameters "LP1" (line printer number 1) for DERR (error messages) and for DLST (JCL listing), and with "CR2" (card reader number 2) for DSRC (JCL source device).

(3) //RUN JCL OBJ=(USER01,JCL,RLK)

Run the JCL translator with default parameters for DSRC, DERR DERR and DLST; use override parameters "JCL" for the file name under fileid "USER01" with password "RLK".

2.5 EXPANDED JCL SPECIFICATION

The following sections describe the format of JCL translator input. A description of the output is not appropriate at this point since it is only accessible to the DX980 subsystems. However, this information is available in Section V within the description of the Start Job supervisor call.

2.5.1 JCL TRANSLATOR INPUT FORMAT

All input commands to the JCL Translator require a single / (slash) as the first character of the source line image. This is followed by a legal command (1 of 6) and any applicable keywords. The six commands are:

CREATE

REPLACE

DELETE

EXEC

ASSIGN

END

These commands are explained later in this section. There are three acceptable formats for these input commands as follows:

(1) /COMMAND <keyword> ...

(2) /COMMAND <keyword>=<value> ...

(3) /COMMAND <keyword>:=<label> ...

2.5.1.1 FORM ONE. The nomenclature <keyword> represents a binary condition that is either "true" or "false". There is always one <keyword> for the "true" condition and another for the "false" condition. If both <keyword>s are specified for the same operand, the dual assignment will be detected as a JCL error.

2.5.1.2 FORM TWO. The notation <keyword>=<value> assigns a value or values to a particular <keyword> or <keyword>s respectively. This form may be used in either of the two formats as follows:

a) $<k_1>=<p_1>, <k_2>=<p_2>, \ldots, <k_n> =<p_n>$

b) $<k>=(<p_1>, <p_2>, \ldots, <p_n>)$

If the second format is used, all parameters are positional and must be supplied or replaced with successive commas.

2.5.1.3 FORM THREE. The notation <keyword>:=<label>, permits the user to specify a label which becomes a <keyword> in the RUN command for operand replacement at job submittal. The translator will accept a combination of form three and either form one or form two for the same operand. This combination creates an operand that may be submitted with an appropriate default value if not overridden in the RUN command. Form three must be used if the operand is to be supplied when the job is submitted.

2.5.1.4 DEFAULTS. Each operand in a given command must either be supplied in one of the three forms or have an acceptable default value of zero. When operands are required but not supplied, DX980 inserts zero for the operands. If the default zero is not acceptable, omission of an operand results in a JCL error. Table 2-1 lists the JCL operand defaults.

Table 2-1. JCL Operand Defaults

Command	Parameter	Default Value
EXEC	PROT/PRIV	PROT
ASSIGN	EXCLUSIVE/SHARE	EXCLUSIVE
	RELEASE/PASS	RELEASE
	OLD/NEW/REPLACE	OLD
	SAVE/DELETE	SAVE
	PASSWORD	0

2.5.2 CONTROL COMMANDS

The first input record to the JCL translator must be a CREATE, REPLACE or DELETE command. The formats for this command are as follows:

```
/CREATE <jcsnam>
/REPLACE <jcsnam>
/DELETE <jcsnam>
```

The CREATE command adds a new JCL sequence, the REPLACE command replaces an existing sequence, and the DELETE command deletes an existing sequence in the JCS file. If CREATE or REPLACE is specified, additional input is expected after the command to define a new sequence. If DELETE is specified, no additional input is expected other than a /END command to terminate the operation. The name of the JCL sequence is <jcsnam>. Subsequent RUN commands to invoke the sequence must specify the same <jcsnam>.

2.5.3 EXECUTE COMMAND

Each job step defined for a sequence must begin with an Execute (EXEC) command. This command contains information to set up the execution environment for the program to be run. Input and output information is not included. If there are multiple steps in a sequence, each step must begin with an Execute command. The Execute command supplies the following information about a program:

- object program
- memory usage
- priority

- time limit
- execution mode

The format of the execute command is as follows:

/EXEC operands

The Execute operands may appear in any order following the command. The following paragraphs explain each operand.

2.5.3.1 OBJECT PROGRAM SPECIFICATION. The object program specification operand supplies the location of the disc file that contains a binary memory image of the object program. The object file to be used as input must have been previously created by DXOLE. There are four parameters required to access a unique object file. The format of these parameters is as follows:

OBJV=<volume>

OBJN=<userid>

OBJF=<filnam>

OBJP=<pswd>

An optional form can also be used:

OBJ=(<volume>, <userid>, <filnam>, <pswd>)

The OBJ parameters can be interpreted in the same manner as the corresponding parameters in the JOB command. For example, <volume> is an integer number from 1 to n that indicates a particular disc drive unit where the object file is located. The other parameters are user assigned. The <pswd> parameter is required only when the file is password protected for the execute attribute.

2.5.3.2 MEMORY USAGE SPECIFICATION. The memory usage specification refers to the runtime memory allocation for the object program, file/device buffers, and user control information. All memory specifications are represented in units of memory words (16 bits per word). Three parameters are required to completely specify user memory and are input in the following format:

MEMT=<stksiz>

MEMU=<jarea>

MEMJ=<jearea>

or in the optional form:

MEM=(<stksiz>,<jarea>,<jearea>)

The parameter <stksiz> specifies the stack size supplied for the initial user task created by DX980 when the job is started. This memory area is used for temporary storage whenever a Supervisor Call (SVC) is executed by the user program (the DX980 SVCs are discussed in Section V.) Since the DX980 SVCs are generally reentrant, execution of an SVC requires a work area for storage of local variables. If several reentrant modules are executed to perform the SVC function, the work areas for each module must be stacked as the SVC is processed. This work area stack must be supplied within the user program's job extension area.

Although each DX980 SVC requires a different amount of memory for the work area stack, the I/O SVCs require the largest amount. Thus, a <stksiz> specification that satisifes I/O SVCs will also satisfy the remaining SVCs. The two types of I/O SVCs are I/O calls either to peripheral devices or to disc. I/O to peripheral devices requires approximately 125 words of <stksiz>, whereas disc file I/O calls can require up to 300 words. Therefore, specify 125 words if there are no disc files assigned and 300 words otherwise. Either size will be sufficient to handle all other SVCs. This <stksiz> specification applies only to the first task. Other tasks within the job define their own stack allocation.

The parameter <jarea> defines the program job area for storage of the user's object program plus any work space required by that program. The size of an object program can be determined from the output of the DX980 Linkage Editor DXOLE. If the program was subjected to a pre-planned overlay when input to the Linkage Editor, <jarea> must be large enough to accommodate the longest or largest overlay segment plus three words per phase (including the root phase) for overlay directory. This information is also available from the Linkage Editor.

If the user requires a program with a dynamic work area that can be specified at runtime, <jarea> must specify sufficient memory for both the object program and the work space. The user program can then determine available memory for the run through Get Memory supervisor calls within the program logic. This concept is particularly useful to allow the amount of program workspace to be specified at runtime rather than when the program is compiled.

The parameter <jearea> defines the user program job extension area and refers to that portion of the user's memory region used for file or device buffers and control information. This area, though not directly accessible from a user program, must be supplied for execution of the program.

The determination of the <jearea> size is much more difficult than that of <jarea> because the size of <jearea> is dependent on the logic of the user program and will vary throughout the course of program executions. The following algorithm, though not exact, provides a guideline for the specification of <jearea>:

$$<\text{jearea}> = \max \left\{ \sum_{i=1}^{NF} \sum_{j=1}^{NB} (BS_i + 1) + 17*NT + 7*NL + SS + 11 \right\}$$

where

NF	=	Number of files assigned to the job step
NB	=	Number of buffers requested for each file
BS	=	Block size
NT	=	Number of co-resident program tasks
NL	=	Number of logical units (LUNs) assigned concurrently to the job step
SS	=	Dynamic TCB stack requirements

Since the terms inside the braces vary as a function of the number of files or devices that are open and of the number of concurrent tasks that exist, the user must determine the maximum value of the expression to specify <jearea> size.

The first term, disc file blocksize, is the sum of all blocking buffers that coexist at any instant in time plus one word of overhead for each block. Since memory is allocated to blocking buffers only when a file is open, this term varies according to the disc file activity in the user program. If multiple buffers are specified for a single disc file, each buffer must be accounted for.

The second term encompasses the memory requirements for task control blocks. Each task control block requires 17 words. Therefore, this term can be determined by multiplying the number of tasks that exist at any instant in time by 17. The operating system creates one task for the user job when the job is started. Therefore, there is always at least one task control block in the job extension area. Any additional tasks are directly created by the program using the Create Task supervisor call (Section V), or indirectly created by the program using an Initiate I/O supervisor call (Section III).

The third term provides space for Logical Device Tables (LDT). There is one LDT for every file or device assigned to the job. Thus, this term can be determined as the number of logical units assigned multiplied by 7 (the size of each LDT).

The fourth term, TCB stacksize, is a companion to the <stksiz> parameter. The memory specified for <stksiz> is allocated from the job extension area when the initial user task is created. If the user creates additional tasks within the program, the <stksiz> specification for the additional tasks must be supplied with the create task SVC. As each task is created, the memory required by the accompanying <stksiz> is also allocated from the job extension area. Thus the TCB stacksize term must provide sufficient memory to accommodate all concurrent tasks.

The fifth term, a constant 11, is due to overhead required by the DX980 memory manager for control information, plus one extra LUN assignment that is made by the operating system for every user program.

Examples:

(1) MEM=(125, 2000, 152)

Allocate memory for a program with a single task and two LUN assignments to non-disc peripherals. The interpretation of this specification is:

(1) <stksiz> of 125 words provides a stack for I/O to non-disc peripherals

(2) <jarea> of 2000 words for the object program. This number is output from DXOLE.

(3) <jearea> of 152 words accounts for one co-resident program task (17 words, two LUNs (14 words), 125 words of TCB stacksize, and 11 words of overhead.

(2) MEM=(300, 4000, 917)

Allocate memory for a program with two tasks, one system created task and one user created task, and two LUN assignments to disc files, each with a physical record length of 128 words. The interpretation of this specification is:

(1) <stksiz> of 300 words for file I/O

(2) <jarea> of 4000 words as specified on the DXOLE load map

(3) <jearea> of 917 words accounts for two co-resident program tasks (34 words), two LUNs (14 words), two physical record buffers (258 words), 600 words of TCB stacksize (assuming that the user created task was also created with a <stksiz> specification of 300 words for disc I/O), and 11 words overhead.

2.5.3.3 PRIORITY SPECIFICATION. Priority specifies the priority of the job step at execution, and the number of unique priority levels needed within the step. This information is input to the DX980 computations which provide resources to the resultant program during job submission and program execution. When the job is submitted, this job step contends for system resources such as memory and peripherals along with all other job steps waiting to be initiated. When the job is executed, scheduling of the CPU is also determined by priority. The required parameters and appropriate mnemonic representations are as follows:

```
PRTL=<nprty>

PRTS=<jsprty>
```

or in the optional form:

```
PRTY=(<nprty>,<jsprty>)
```

The nomenclature <nprty> represents the number of task priority levels and is greater than one only if the program uses the multi-tasking feature of DX980. This number corresponds to the maximum number of separate task priority levels (not the number of separate tasks) that may be created within the user program. Several tasks may share a common priority level.

The notation <jsprty> represents the job step priority, which is the priority assigned to the job step for scheduling purposes and to the initial task created by the operating system. The limits on <jsprty> are 1 to 31 where 1 is the highest and 31 the lowest priority permissible to a user program. More than one job may execute concurrently while sharing a common priority level.

When a job step is submitted, DX980 checks the limits of the priority parameters as follows:

1 ≤ <nprty> ≤ 31

1 ≤ <jsprty> ≤ 31

1 ≤ <nprty> + <jsprty> ≤ 32

The limits on each parameter are due to the priority limits in DX980. The combined limit is due to the method of creating one task from another in DX980. If the user program creates additional tasks, the priority of each created task is specified relative to the job step priority (0, 1, 2, etc.), where the relative numbers are always positive. Thus, a task cannot be created from a user program with a higher priority than the job step priority. However, tasks can be created with priority equal to or lower than job priority.

If any task is created with a priority lower than 31, the creation will be recognized as a fatal error and the job will be aborted. This criteria produces the combined <nprty> and <jsprty> limit.

Examples:

PRTY=(1,15)	This job step contains only one task priority level. Both the job step and the task(s) created for program execution are assigned priority 15.
PRTY=(2,1)	There are two task priority levels in this job step, levels 1 and 2. The job step is assigned priority 1.

2.5.3.4 TIME LIMIT SPECIFICATION. Time limit specifies the maximum amount of time for execution to be allotted to a job step. The time monitored represents actual time that the program has control of the CPU (rather than wall-clock time). If the program has not terminated normally when the time limit passes, the operating system abnormally terminates the program. Time parameter format is as follows:

```
TIME= time limit in seconds
```

To instruct the translator and operating system to run a job with an infinite time limit, set the time limit to a minus one: TIME=-1. This input is interpreted as a directive to ignore timeout checking completely.

2.5.3.5 EXECUTION MODE SPECIFICATION. Privileged or protected refers to the mode of the program during execution. Normally user programs run in the protected mode so that they are prevented from damaging the operating system. System programs always run in the privileged mode, allowing them to move freely within available memory. The format of this parameter is as follows:

```
PRIV

PROT
```

2.5.3.6 EXECUTE COMMAND EXAMPLE. The following input sample illustrates the use of the EXEC command:

```
/EXEC OBJ=(1,SYSTEM,ASMBLR), MEM=(300,5000,1000),PRTY=(1,15);

/TIME=-1 MEM:=MEM, PRTY:=PRI, TIME:=TIM
```

2.5.4 ASSIGNMENT COMMAND

Each job step must define the logical units for I/O operations. I/O assignments are not carried forward from one job step to another. Therefore, an assignment command must be included for each Logical Unit Number (LUN) referenced during execution of each job step, except when using the Execution Time Allocate SVC (see Section V). An assignment command has a slash (/) in column 1 and the keyword "ASSIGN" in columns 2-7. Following ASSIGN, the assignment operands may be specified in the same three forms as on the execute command. For convenience, however, the logical unit number and device name operands may be specified positionally, logical unit number first and device name second, or with the< keyword>=<value> form. As with all operands, the< keyword>:=<label> form can provide override capability when the job is submitted.

Each assignment command in each job step must have the following minimal information:

- LUN
- device name
- device sharability
- device disposition

If device name specifies a random access device (moving head disc), then the following information must be supplied also:

- file creation
- file identification
- file disposition
- number of file buffers (key indexed files only)

If file creation is specified as "NEW", then the following must also be specified:

- file type
- file integrity
- file allocation
- key length (key indexed files only)
- logical record length (relative record files only)

Each operand, together with the operand limits and default value, if any, is described in the following sections.

2.5.4.1 LUN OPERAND. LUNs provide a consistent method of communication between a user program and the DX980 I/O system. All I/O requests from a user program must be accompanied by a LUN. Each LUN must be coupled to a physical device or file through the assignment command. This feature allows I/O requests in a program to be device independent, and defers the actual device assignments until the program is submitted for execution. The LUN operand can take either of the following forms:

LUNO= <lun>

or

/ASSIGN <lun>

The notation <lun> is a decimal number in the range of 0 to 250 (LUNs from 251 through 255 are reserved for operating system use), and must be unique within a job step. Duplicate assignments of the same LUN within a job step will be recognized as an error. However, the same LUN can be used by more than one job, since each assignment applies only to a specific job.

2.5.4.2 DEVICE NAME. Device names refer to the mnemonics that represent the physical devices attached to the computer. The translator accepts any of the permissible device mnemonics that could exist for a DX980 installation. If an assigned device does not actually exist, one of the subsystems detects the error when the job is submitted and the job is aborted. The DEVICE NAME operand can take the form of either of the following forms:

DEVICE=<devnam>

or

/ASSIGN <lun> <devnam>

The notation <devnam> represents a user supplied mnemonic that must be selected from the list in table 2-2.

Table 2-2. Suggested Device Names

Mnemonic	Device Index Range	Device Description
DISC1-DISCn n ≤ 20	1 - 20	Disc (Moving head or DS330) 1 through n, n ≤ 20
KEY1-KEYn	21 - 30	Teleprinter keyboard (ASR/KSR 33, 730, 733, 735) 1 through n
CRT1-CRTn	21 - 30	Video display 1 through n
SC	21 - 30	System console data terminal (Teleprinter keyboard or video display)

Table 2-2. Suggested Device Names (Continued)

Mnemonic	Device Index Range	Device Description
MT1-MTn	31 - 40	Magnetic tape 1 through n
LP1-LPn	41 - 50	Line printer 1 through n
CR1-CRn	51 - 60	Card reader 1 through n
PTR1-PTRn	61 - 70	Paper tape reader 1 through n
PTP1-PTPn	71 - 80	Paper tape punch 1 through n
DM1-DMn	81 - 90	Data module interface devices 1 through n
ADDA1-ADDAn	91 - 100	Analog-to-digital/digital-to-analog converter 1 through n
CS11, CS12, . . . , CSn1, CSn2	101 - 110	Cassette 1 on teleprinter 1 through cassette 2 on teleprinter n
DUMMY	255	Dummy device - responds with an immediate end-of-file on input and with no-op on output
SYSIN	-	System input spooler
SYSOUT	-	System output spooler
TERMIO	-	Interactive terminal log on device

2.5.4.3 DEVICE SHARABILITY. A device or file can be designated as either shared or exclusive. These attributes are specified by the binary representation of the keywords "SHARE" or "EXCLUSIVE". The default value is exclusive. If a device or file is exclusively assigned to one job, then any other job that is submitted with an assignment to that device or file, either shared or exclusive, will not be started until the first job releases it. If a device or file is selected as shared by all assigning programs, then all programs may run concurrently.

2.5.4.4 DEVICE DISPOSITION. The device assignment may be kept or released when the current job step is completed. This choice is specified by the binary keywords "RELEASE" or "PASS". The default value is release.

Assignment passing in DX980 is extremely useful for multi-step jobs that require the same file or device in more than one step. Further, passing a device or linked sequential file will cause the physical position to be maintained between job steps (key indexed and relative record files do not hold position between steps). Even though passed, a device must be assigned to a LUN for the next job step.

The passing of only a portion of the required resources for a job string can cause a resource deadlock. However, DX980 guards against any deadlock by cancelling jobs that have passed resources from a previous job step and that are also requiring unavailable resources. Thus, to prevent a job cancellation caused by the running job environment, the user should specify that all resources are passed from job step to job step.

2.5.4.5 FILE CREATION. A file may be either old, new, or a replacement. These attributes are specified by the binary representation of the keywords "OLD", "NEW" and "REPLACE" which are defined as follows:

- OLD: File already exists (default value).
- NEW: Create a file. Error if file already exists.
- REPLACE: Replace file if it is there; create a file if it is not there.

2.5.4.6 FILE IDENTIFICATION. A file is accessed via a file directory (or dictionary) and file name, and can be protected by a password. The directory, name and password can be specified with either of the following formats:

```
    FILDIR=<fileid>

    FILNAME=<filnam>

    PASSWORD=<pswd>
```

or

```
    FILE=(<fileid>,<filnam>, [<pswd>])
```

The <fileid> and <filnam> parameters are identical to the <fileid> and <filnam> parameters in the JOB command. A special case of the <fileid> parameter is provided for temporary files. The user can specify a file for use only during a job string, by entering a <fileid> of "TEMP". This file is never entered into a User File Directory, but is totally unique to the job string using it. The file creation parameter should be NEW for the first reference to a temporary file, and OLD for subsequent references. All other parameters are specified as usual. The file is deleted at the end of the job string unless deletion is specifically called for at the end of a job step by the DELETE directive.

The notation <pswd> denotes a user supplied password that is used in conjunction with file integrity to control file access as described under "File Integrity" in this section. If this parameter is specified when a file is created, it will be required to access the file. Otherwise, <pswd> is ignored.

2.5.4.7 FILE DISPOSITION. The user can choose to either delete or save a file after a job step. These two alternatives are specified by the binary keywords:

SAVE

DELETE

The default value is SAVE. If the device disposition parameter was "PASS", specifying "DELETE" produces an error condition that is detected by the operating system.

2.5.4.8 FILE BUFFERS. Linked sequential files always require one buffer. Relative record files require no buffers if the logical record length equals the physical record length (unblocked), and require one buffer if the logical record length is less than the physical record length (blocked). These specifications are fixed and are unaffected by user input in JCL. However, file management supports a variable number of buffers for key indexed files. For a sequential access of the data records within a key indexed file, only one buffer is required. For keyed access, a minimum of two buffers is required: one for key records and one for data records. Using more than two buffers for these files reduces the amount of disc I/O required and increases program speed. Therefore, the number of buffers assigned to a key indexed file is controlled by the user through the Buffers specification. The format for this specification is:

BUFFERS=<nbufs>

2.5.4.9 FILE TYPES. A file may be either linked sequential, relative record, or indexed. These attributes are specified by the binary representation of the keywords "LINKSEQ", "RELREC", and "INDEXED". There is no default for file type, so it must always be specified. See Section IV for a description of the three file types.

2.5.4.10 FILE INTEGRITY. Files can be accessed under DX980 for one of four functions: read, write, delete, or execute. Data management purposes frequently require unlimited access to a file for one or more of these function, but only limited access for others. Therefore, DX980 provides an integrity mechanism for each of these functions. For example, DX980 can be instructed to allow any user access to a file for reading but require a password for access during writing or deleting. The format for supplying integrity codes (only appropriate for newly created files) is either:

READCODE=<integ>

WRITCODE=<integ>

DELCODE=<integ>

EXECODE=<integ>

or

ACCESS=(<integ>,<integ>, <integ>,<integ>)

The notation <integ> represents the integrity code supplied for each function. Select the code from the list of values in table 2-3.

Table 2-3. File Integrity Codes

Code	Access Granted To
ANY	All users for the specified function.
PSWD	User with password only (those users that specify a <pswd> parameter corresponding to the <pswd> parameter of the file).
CREAT	Creator of the file only (the users that specify a <pswd> parameter corresponding to the <pswd> parameter of the file and whose <userid> in the JOB command matches the <fileid> of the file).
NONE	No one (no access allowed for the specified function).

An "ANY" code specifies unlimited access, and allows a user to access the file for any specified function for which he knows the appropriate <fileid> and <filnam>. In this case the <pswd> parameter is not required for file specification.

A "PSWD" code specifies that access for the associated function is possible only if the proper <pswd> parameter is supplied in file specification. When a file is created with "PSWD" specified for a function, a <pswd> parameter must also be supplied during file specification. Integrity code violations are detected when the accessing program is running rather than when it is submitted. For each I/O request DX980 verifies that the I/O opcode is valid for the integrity code that was established when the file was created.

A "CREAT" code specifies that the <pswd> parameter must match the <pswd> parameter of the file and that the <userid> parameter in the JOB command must match the <fileid> parameter in the file specification operand for a user to gain access for the associated function.

A "NONE" code specifies that access is prohibited for the associated function. Normally, the only function that has "NONE" specified for an integrity code is "EXECODE"; however, "DELCODE" may also be specified as "NONE". The JCL translator ensures that "NONE" is not specified for READCODE or

WRITCODE, and flags such a specification as a JCL error. A "NONE" code is automatically specified for EXECODE in the case of linked sequential and key indexed files since those file types are not compatible with the program loader in DX980.

2.5.4.11 FILE ALLOCATION. File allocation refers to the allocation of file space on a disc. This operand is pertinent only for files being newly created (NEW or REPLACE). Under DX980 the user must specify the initial and maximum space allocation, plus the physical record size of the file that will occupy the space. The user can also indicate to DX980 where the file is located on the disc. The format for supplying this information is one of the following:

```
INITIAL=<itrks>
LOCN=<trknum>
PRECL=<prwrds>
MAXTRACK=<mtrks>
```

or

```
ALLOCATE=(<itrks>,<trknum>,<prwrds>,<mtrks>)
```

The nomenclature <itrks> represents the initial number of disc tracks that are allocated to the file when the file is created. If the file type operand is "RELREC", the initial allocation is forced to match the maximum allocation because relative record files must be stored contiguously on the disc and cannot grow. For the other file types, however, the file organizations are noncontiguous and allow the file to grow beyond its initial allocation. The default value for <itrks> is 1.

The nomenclature <trknum> represents the disc track where DX980 will start searching for space to allocate to the file. The track allocation mechanism in DX980 starts searching for the number of tracks specified in <itrks> at track number <trknum> and continues the search until it finds sufficient contiguous space to satisfy the request. Normally the search starts at track 0 and continues across the entire disc if necessary. Although disc tracks on a moving head disc are not physically constructed in ascending order across the disc, they are considered as such for the specification of <trknum>. In general, specifying a nonzero <trknum> may be used to position the new file close to another file.

This specification is useful only to reduce head movement on a moving head disc drive when both files are accessed by a single program. This arrangement can result in a significant throughput increase for I/O bound programs that access more than one disc file.

The notation <prwrds> designates the number of words in each physical record. Each physical record stored on the disc is preceded by a record header. The moving head disc has a header at the beginning of each disc sector regardless of physical record length and individual track formatting is unnecessary. The DS330 disc system requires track formatting that involves writing record headers at the beginning of each physical record across the entire track. Various track formats are shown in table 2-4 for the DS330 disc type. Maximum values for <prwrds> are shown in table 2-5. Formatting time for DS330 is 80 ms/track and occurs when the tracks are allocated to the file, either initially or when expanding to another track. Since the disc systems supported by DX980 store data in blocks of 32 words, it is necessary to constrain <prwrds> to multiples of 32 words. Furthermore, since physical record length is analogous to blocking buffer size, memory allocation for the job extension area (<jearea> in the JOB command) must comprehend the physical record length of each file assigned to a program.

The parameter <mtrks> defines the maximum number of tracks that are allocated to a file that can grow, and in particular, to linked sequential and key indexed files. The initial allocation (<itrks>) for these file types is made when the file is created. When an operation tries to add data after the initial allocation is used up, the file grows one track at a time until either the additions are completed or the number of tracks specified by <mtrks> is reached. In the latter case, the offending program terminates abnormally after file management allocates one extra track to perform the write operation in progress.

Examples:

(1) ALLOCATE=(1, 0, 32, 1)	Allocate 1 initial track, starting the search at track 0. Format each physical record into 32 words and allocate no more than 1 track regardless of program activity.
(2) ALLOCATE=(5, 100, 512, 10)	Allocate 5 initial tracks, starting the search at track 100. Format each physical record into 512 words and allow the file to grow to a maximum of 10 tracks.

Table 2-4. DS330 Disc Formatting

Records/Track	Sectors/Record	Words/Record <prwrd>	Useful Words/Track	Efficiency
88	1	32	2816	41.90
44	2	96	4224	62.86
29	3	160	4640	69.05
22	4	224	4928	73.33
17	5	288	4896	72.86
14	6	384	5376	80.00
12	7	448	5376	80.00
11	8	512	5632	83.81
9	9	576	5184	77.14
8	10	672	5376	80.00
8	11	736	5888	87.62
7	12	800	5600	83.33
6	13	864	5184	77.14
6	14	960	5760	85.71
5	15	1024	5120	76.19
5	16	1088	5440	80.95
5	17	1152	5760	85.71
4	18	1248	4992	74.29
4	19	1312	5248	78.10
4	20	1376	5504	81.90
4	21	1440	5760	85.71
4	22	1536	6144	91.43
3	23	1600	4800	71.43
3	24	1664	4992	74.29
3	25	1728	5184	77.14
3	26	1824	5472	81.43
3	27	1888	5664	84.29

Table 2-4. DS330 Disc Formatting (Continued)

Records/Track	Sectors/Record	Words/Record <prwrd>	Useful Words/Track	Efficiency
3	28	1952	5856	87.14
3	29	2016	6048	90.00
2	30	2112	4224	62.86
2	31	2176	4352	64.76
2	32	2240	4480	66.67
2	33	2304	4608	68.57
2	34	2400	4800	71.43
2	35	2464	4928	73.33
2	36	2528	5056	75.24
2	37	2592	5184	77.14
2	38	2688	5376	80.00
2	39	2752	5504	81.90
2	40	2816	5632	83.81
2	41	2880	5760	85.71
2	42	2976	5952	88.57
2	43	3040	6080	90.48
2	44	3104	6208	92.38
1	45	3168	3168	47.14
1	46	3264	3264	48.57
1	47	3328	3328	49.52
1	48	3392	3392	50.48
1	49	3456	3456	51.43
1	50	3520	3520	52.38
1	51	3616	3616	53.81
1	52	3680	3680	54.76
1	53	3744	3744	55.71
1	54	3808	3808	56.67

Table 2-4. DS330 Disc Formatting (Continued)

Records/Track	Sectors/Record	Words/Record <prwrd>	Useful Words/Track	Efficiency
1	55	3904	3904	58.10
1	56	3968	3968	59.05
1	57	4032	4032	60.00
1	58	4096	4096	60.95
1	59	4192	4192	62.38
1	60	4256	4256	63.33
1	61	4320	4320	64.29
1	62	4384	4384	65.24
1	63	4480	4480	66.67
1	64	4544	4544	67.62
1	65	4608	4608	68.57
1	66	4672	4672	69.52
1	67	4768	4768	70.95
1	68	4832	4832	71.90
1	69	4896	4896	72.86
1	70	4960	4960	73.81
1	71	5056	5056	75.24
1	72	5120	5120	76.19
1	73	5184	5184	77.14
1	74	5248	5248	78.10
1	75	5344	5344	79.52
1	76	5408	5408	80.48
1	77	5472	5472	81.43
1	78	5536	5536	82.38
1	79	5632	5632	83.81
1	80	5696	5696	84.76
1	81	5760	5760	85.71

Table 2-4. DS330 Disc Formatting (Continued)

Records/Track	Sectors/Record	Words/Record <prwrd>	Useful Words/Track	Efficiency
1	82	5824	5824	86.67
1	83	5920	5920	88.10
1	84	5984	5984	89.05
1	85	6048	6048	90.00
1	86	6112	6112	90.95
1	87	6208	6208	92.38
1	88	6272	6272	93.33

Table 2-5. Maximum <prwds> Physical Record Lengths for Disc Files

File Type	DIABLO Type Disc	DS330 Type Disc
Relative Record	2816 (full track)	6272 (full track)
Linked Sequential	1408 (half track)	3104 (half track)
Key Indexed	2816 (full track)	6272 (full track)

2.5.4.12 KEY LENGTH. The length of keys for new indexed files can be specified as in the following format:

KEYLEN=<klchar>

The notation <klchar> represents the number of characters in each key for a key-indexed file and must be in the range of 1 to 30. The physical record length must be able to hold at least 2 keys plus 14 words. Providing a physical record length that can hold 10 or more keys increases search efficiency. Section IV contains a detailed description of key index file directory formats.

2.5.4.13 LOGICAL RECORD LENGTH. The logical record length for new, relative record files can be specified in the following format:

LRECL=<lrchar>

The parameter <lrchar> specifies the logical record length in characters. If the relative record file is blocked, then the logical record length must be less than or equal to the physical record length. If the file is unblocked, then the logical record length must be a multiple of 32, and must be equal to the physical record length. LRECL is not appropriate for linked sequential and key indexed files since both allow variable length logical records.

2.5.4.14 ASSIGNMENT EXAMPLES. The following examples illustrate possible uses of the ASSIGN parameters.

(1) /ASSIGN, 5, CR1, EXCLUSIVE, RELEASE.
Assign LUN 5 to the card reader (CR). The card reader is not shared with any other user (EXCLUSIVE), and is released at the completion of the job step (RELEASE). The period (.) indicates the end of the assignment and allows addition of user comments.

(2) /ASSIGN 5 CR1 EXCLUSIVE RELEASE
This example is equivalent to example 1. Delimiters between operands can be either blanks or commas. The period at the end of the statement is optional.

(3) /ASSIGN 13 DISC1 SHARE PASS OLD, FILE=(USRNAM, NAME)
Assign LUN 13 to disc 1. The disc file is shared (SHARE) and passed to the next job step (PASS). Since a disc is specified, certain file information must be given. The file is old (already exists), and can be referenced by name NAME in directory USRNAM.

(4) /ASSIGN 250 DISC1 SHARE PASS;
/NEW, RELREC, FILE=(USRNAM, NAME), SAVE;
/ACCESS=(ANY, CREAT, CREAT, PSWD);
/ALLOCATE=(1, 38, 64, 1), LRECL=40
A new file was specified so that more information is required in addition to that specified in example 3. The new file can be read by anyone (ANY), can only be written into or deleted by the creator of the file (CREAT, CREAT), and requires a password to execute code from the file (PSWD). The file is to have 1 track allocated and the search for this track must start at track 38. Each physical record is 64 words. Finally, the logical record length is 40 characters.

(5) /ASSIGN 6 SYSOUT
SYSIN and SYSOUT are system spooling devices.

(6) /ASSIGN DEVICE=SYSOUT, LUNO=6
An alternative form of example 5.

(7) /ASSIGN 6 SYSOUT LUNO:=NEWLUN
This example has the same effect as examples 5 and 6 except that the LUN can be changed for this assignment by specifying //RUN ... NEWLUN=7... when the job is submitted.

2.5.5 JOB CONTINUATION/TERMINATION

Either another /EXEC card to specify a new job step, or a /END card to signify that all job steps for the job have been defined through the JCL Translation phase must follow the last assignment of a job step. If the system detects no errors in processing the sequence of job steps, it writes the information specifying the job in the assigned output file, and locates that file in a position specified on either the CREATE or REPLACE control card.

2.6 FORMAT SUMMARY

Figure 2-1 summarizes the formatting options and requirements for job subsubmittal commands and for extended JCL commands. The format for submittal commands appears at the top of the figure. The JOB command format is totally generalized; whereas the RUN command is an example that references an existing JCL procedure. The JCS for this existing procedure appears in table 2-6. To clarify this example, table 2-7 lists the keynames that appear in table 2-6 that the JCL Translator recognizes, together with the default parameters and the default override labels for each keyname. If both a default value and an override label appear for the parameter, then it can be modified at runtime. If no default value is listed, then the parameter must be specified at runtime. If no override label is listed, then the parameter cannot be modified at runtime. Following the submittal commands in the figure are the extended JCL commands. These commands constitute the input to the JCL translator.

The figure displays, one above the other, all possible options for supplying job step information to the JCL translator. Dotted lines illustrate the points of convergence for the options. For example at the beginning of the line labeled "New File Specification", the three possible means for specifying file types appear as follows:

```
/INDEXED, KEYLEN=<klchar ≤30>,
                              /
 RELREC, LRECL=<lrchar>-----/
                            /
 LINKSEQ ------------------/
```

This notation indicates that immediately following the slash (/) one of the three forms must be used to designate the file type. The dotted line indicates that all of the forms must be immediately followed by a comma before entering the next parameter.

```
JOB SUBMITTAL COMMANDS

JOB          //JOB <jsname> <userid> FILE=(<volume>,<fileid>,<filnam>,<pswd>)
Command                              FILE=(<volume>,<fileid>,<filnam>)
                                     ƀ, 1>
    2
RUN
Command      //RUN XJD,DSRC=<devnam>,FSRC=(<fileid>,<filnam>,<pswd>),DERR=<devnam>,LIST=<devnam>,DOBJ=DISC<n>,FOBJ=(<volume>,<fileid>,<filnam>),REP;
                   ƀ 3>         ƀ 3>                                ƀ 3>           ƀ 3>          ƀ 3>          ƀ 3>                          OLD;
                                                                                                                                             NEW;
                                                                                                                                             ƀ
EXTENDED JCL COMMANDS (JCL TRANSLATOR INPUT)                   LOBJ=(<itrks>,<trknum>,<prwrds>,<mtrks>),MEM=<stksiz><jarea>,<jearea>)
                                                                                                   ƀ. 3>
Control      /DELETE <jcsnam>  4>
Commands     /REPLACE <jcsnam>

             /CREATE <jcsnam>

Execute      /EXEC OBJV=<volume>,OBJN=<userid>,OBJF=<filnam>,OBJP=<pswd>,MEMT=<stksiz>,MEMU=<jarea>,MEMJ=<jearea>,PRTL=<nprty>,PRTS=<jsprty>,TIME=<seconds>,PRIV
Command            OBJ=(<volume>,<userid>,<filnam>,<pswd>)            MEM=(<stksiz>,<jarea>,<jearea>)        PRTY=(<nprty>,<jsprty>)         <-1>     PROT
                                                                                                                                        5>       ƀ

ASSIGN       /ASSIGN LUNO=<lun>,DEVICE=<devnam>,EXCLUSIVE,RELEASE,FILDIR=<fileid>,FILNAME=<filnam>,PASSWORD=<pswd>,BUFFERS=<nbufs>,REPLACE;
Command              <lun>,<devnam>            SHARE      PASS    FILDIR=TEMP,                    ƀ 6>              8>        OLD;
                                               ƀ 6>       ƀ 6>    FILE=(<fileid>,<filnam>,<pswd>)                                 NEW;
                                                                  FILE=(<fileid>,<filnam>)
                                                                  FILE=(TEMP,<filnam>,<pswd>)                                     ƀ 6>
                                                                  FILE=(TEMP,<filnam>)
                                         9>
New File      /INDEXED,KEYLEN=<klchar≤30>,READCODE=<integ>,WRITCODE=<integ>,DELCODE=<integ>,EXECODE=<integ>,DELETE,INITIAL=<itrks>,LOCN=<trknum>,PRECL=<prwrds>,MAXTRACK=<mtrks>
Specification  RELREC,LRECL=<lrchar>   ACCESS=(<integ>,<integ>,<integ>,<integ>)                        SAVE   ALLOCATE=(<itrks>,<trknum>,<prwrds>,<mtrks>)
               LINKSEQ                                                                                 ƀ
                                         9>                                                            6> 10>

End of job    /END
Command
```

NOTES:

1> Refer to section 2.4.1.5 for file default on JOB Command.

2> The RUN Command given here is an example. Refer to table 2-5 for the referenced JCS

3> Refer to table 2-6 for defaults specified in the referenced JCS

4> DELETE does not require further input.

5> -1 designates infinite run time.

6> Refer to table 2-6 for standard defaults.

7> No further input required for non file devices.

8> BUFFERS = 1 for RELREQ and LINKSEQ; > 1 for INDEXED

9> <integ> must be one of following: ANY,PSWD,CREAT,NONE.

10> PASS and DELETE cannot be specified simultaneously.

(B)129979

Figure 2-1. JCL Translator Formatting Summary

Table 2-6. Job Control Sequence (JCS) for RUN Command Example

```
,* CREATE JCL   ,COMMENT,"CREATE JCL PROCEDURE        "
/REPLACE JCL          . CREATE JCL PROCEDURE .
/EXEC OBJ=(1,SYSTEM,JCLTRN) MEM=(300,7550,1000) PRTY=(1,15);
/           TIME=-1 MEM:=MEM PRTY:=PRI
/ASSIGN  1 SC    DEVICE:=DSRC FILE:=FSRC BUFFERS=1            , SOURCE INPUT
/ASSIGN  2 SC    DEVICE:=DERR SHARE:=SERR                     . ERROR MESSAGE
/ASSIGN  3 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.SOURCE LISTING
/ASSIGN  4 DISC1 DEVICE:=DOBJ FILE=(SYSTEM,SJCBFL,AB);
/          FILE:=FOBJ REPLACE:=ROBJ BUFFERS=2 INDEXED;
/          ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=COBJ;
/          ALLOCATE=(1,0,96,20) ALLOCATE:=LOBJ KEYLEN=6       . OBJECT OUT FILE
/END
```

Table 2-7. Parameter Keynames and Defaults for JCS Example

Extended JCL Keyname	Default	Abbreviated JCL Override Label
OBJ	(1, SYSTEM, JCLTRN)	-
MEM	(300, 7600, 800)	MEM
PRTY	(1, 15)	PRI
TIME	-1	-
LUNO	1	
DEVICE	SC	DSRC
FILE	-	FSRC
BUFFERS	2	-
LUNO	2	
DEVICE	SC	DERR
LUNO	3	
DEVICE	SC	DLST
LUNO	4	
DEVICE	DISC1	DOBJ
FILE	(SYSTEM, SJCBFL, AB)	FOBJ
OLD/NEW/REPLACE	OLD	-/NEW/REP
LINKSEQ/RELREC/INDEXED	INDEXED	-
ACCESS	(ANY, ANY, ANY, ANY)	COBJ
ALLOCATE	-	LOBJ
KEYLEN	6	-

SECTION III
INPUT/OUTPUT STRUCTURE

3.1 GENERAL

A user program initiates input/output operations with an Input/Output Supervisor Call (I/O SVC). The SVC is an illegal machine instruction that generates an internal interrupt. The internal interrupt decoder turns control over to the SVC processor. After determining that an I/O SVC has been made, the SVC processor gives control to the I/O Management portion of the operating system. A set of modules, called Device Service Routines (DSRs), handles the primary I/O workload.

A Device Service Routine contains several logical paths. When an I/O SVC is made, the system follows the initial entry path. The initial entry path begins the I/O operation and may or may not complete it. I/O device interrupt processing follows another path through the DSR. I/O interrupts can occur from the I/O Bus or the Direct Memory Access Channel (DMAC) of the computer. They occur while an I/O operation is in progress or when it is finished. Interrupts from the I/O Bus typically occur on a character-by-character basis. Interrupts from DMAC devices occur only when the operation is complete. The system also has a reset path through the DSR to stop uncompleted I/O and to initialize the device and DSR. Separate operating system modules, working with a disc DSR, control file management for the disc. Section IV of this manual describes DX980 file management.

3.2 I/O SUPERVISOR CALLS

The I/O SVC is made by attempting to execute one of two illegal machine instructions: >C380 or >F800. The programmer usually defines the I/O SVC instruction word with a symbolic name using the OPD (Operation Definition) assembler directive. To complete the call, the M-register must contain the address of either a Physical Record Block (PRB) for the C380 instruction, or an argument list for the F800 instruction. The PRB is a list of parameters describing the I/O operation that has been requested. For example, if the programmer defines the I/O SVC as IOC (using OPD), a typical I/O call using the C380 machine instruction would be coded as shown on the following page.

```
SAP R2LC
                                                                 SHEET  0001

               0001          IDT   SAMPLE
               0002   *
               0003   *
               0004   IOC    OPD   >C380,3           DEFINE IOC
               0005   *
               0006   *
  0000  1800   0007          @LDM  =PRB              PRB ADDR TO MREG
P 0001  0003
  0002  C380   0008          IOC   0                 SVC INSTRUCTION
               0009   *******************************************************
               0010   *
  0003  0000   0011   PRB    DATA  0                 DEFINE PRB
               0012   *
        0000   0013          END
```

The F800 instruction is a special form of the more general F8XX format. The letters XX indicate the SVC number to be executed (Section V lists additional SVC numbers). Using this format, the M-register must point to an argument list with the following arrangement:

- Word 0 - This argument list word specifies the number of arguments in the list. Word 0 of an I/O SVC contains a 1. If the SVC has no argument, word 0 contains a 0.
- Word 1 - This argument list word contains a pointer to the first argument. Word 1 of an I/O SVC points to the PRB.
- Word 2 through word n - These argument list words contain pointers to arguments 2 through n, respectively. These arguments do not apply to an I/O SVC since the PRB is the only argument.

```
SAP R2LC
                                                                 SHEET  0001

               0001          IDT   SAMPLE
        0000   0002   IO     EQU   0
               0003   SVX    OPD   >F800,8           DEFINE SVX
               0004   *
               0005   *
  0000  1800   0006          @LDM  =ARGLST           LIST ADDR TO MREG
P 0001  0003
  0002  F800   0007          SVX   IO                EXECUTE CALL
               0008   *
               0009   *
               0010   ********************************************
               0011   *
  0003  0001   0012   ARGLST DATA  1,PRB             NUM ARGS,ADDRS
P 0004  0005
               0013   *
  0005  0000   0014   PRB    DATA  0                 DEFINE PRB
               0015   *
        0000   0016          END
```

3.3 PHYSICAL RECORD BLOCK

The Physical Record Block (PRB) is the list of parameters necessary for the supervisor to perform an I/O operation. The PRB is either four or five words long as shown in figure 3-1. The following paragraphs explain the use of these parameters. Variations of this PRB structure occur for non-standard I/O operations. Also, AD/DA and data module devices use a slightly different PRB format. Refer to the description of these device characteristics later in this section.

3.3.1 WORD 0

PRB Word 0 contains flags that are controlled by the operating system I/O routines, and a logical unit number specified by the calling program.

3.3.1.1 BIT 0 (BUSY BIT). Bit 0 applies only to Initiate I/O calls. This bit is set during the I/O operation. When set, this bit indicates that the program should not disturb the PRB or referenced data buffer.

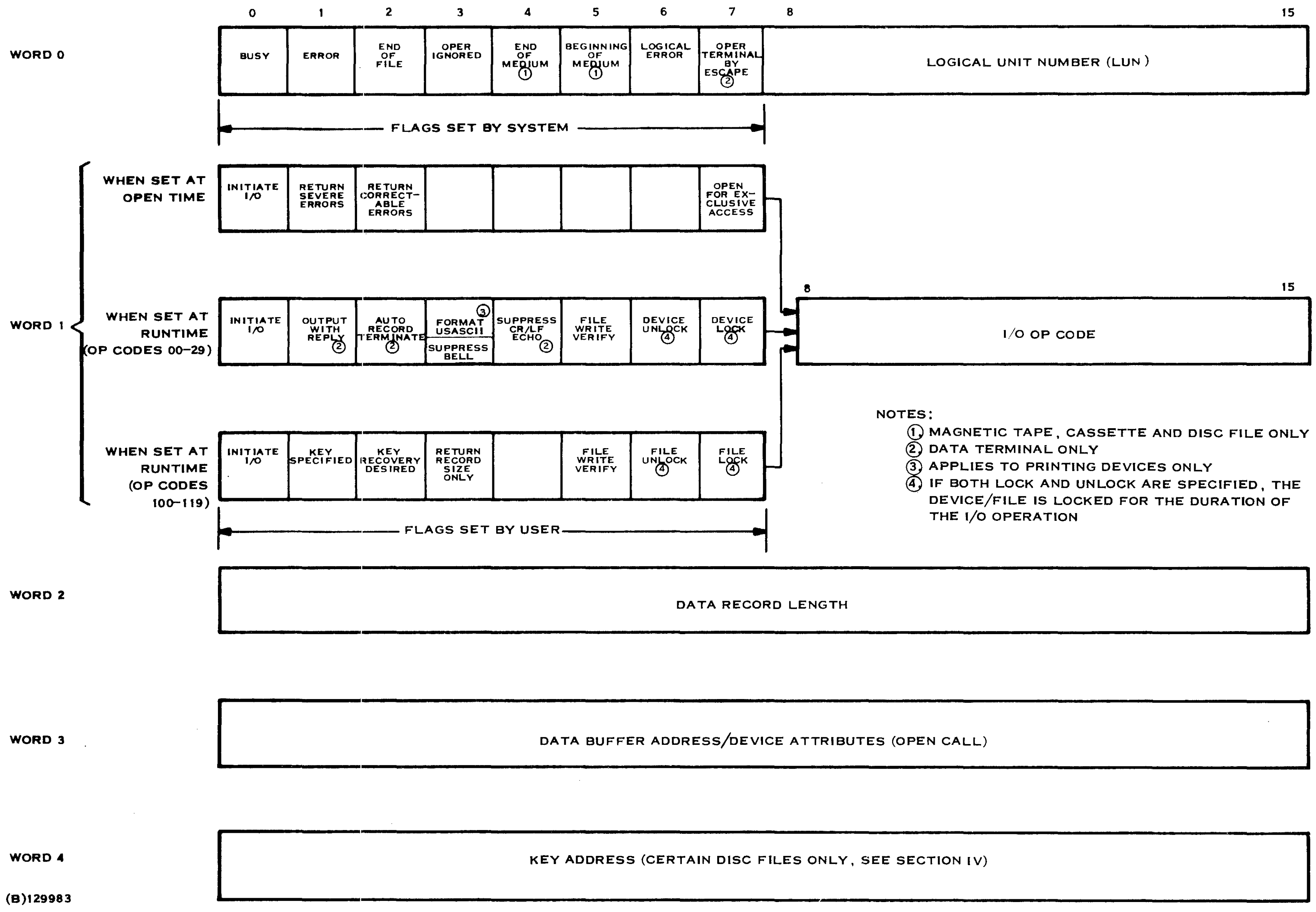
3.3.1.2 BIT 1 (ERROR BIT). Bit 1 sets if an unrecoverable I/O error or a logical error (see bit 6) was detected in the last operation performed using this PRB. This bit resets if no error was detected. If Return on I/O Errors was not requested when the device was opened, the job aborts on an unrecoverable I/O error. If this bit is set following a Return, then PRB word 2 contains an error code greater than 200. Appendix A of this manual defines these error codes.

3.3.1.3 BIT 2 (END OF FILE BIT). Bit 2 sets if the last record read with this PRB was an end of file. For most media an end of file is a record whose first two characters are /*. End of File applies only to reading and spacing opcodes. (Opcodes 00, 01, 05, 06, 14 and 15.)

3.3.1.4 BIT 3 (OPCODE IGNORED BIT). Bit 3 sets if the last I/O operation was ignored because of the physical limitations of the I/O device. For example, an attempt to backspace a card reader sets this bit. Opcode Ignored is not necessarily an error condition.

3.3.1.5 BIT 4 (END OF MEDIUM BIT). Bit 4 sets if the physical end of the storage medium was reached. Magnetic tape, cassette, and disc files provide end of medium detection.

3.3.1.6 BIT 5 (BEGINNING OF MEDIUM). Bit 5 sets if the physical beginning of the storage medium was reached.



NOTES:
① MAGNETIC TAPE, CASSETTE AND DISC FILE ONLY
② DATA TERMINAL ONLY
③ APPLIES TO PRINTING DEVICES ONLY
④ IF BOTH LOCK AND UNLOCK ARE SPECIFIED, THE DEVICE/FILE IS LOCKED FOR THE DURATION OF THE I/O OPERATION

(B)129983

Figure 3-1. Physical Record Block (PRB) Format

3.3.1.7 BIT 6 (LOGICAL ERROR). Bit 6 sets to indicate than an error occurred that may have been a successful request under other conditions. A Logical Error applies only to file management. For example, a write with key to a key indexed file when the specified key already exists. When bit 6 is set, bit 1 is also set.

3.3.1.8 BIT 7 (OPERATION TERMINATION). Bit 7 sets when the escape key (ESC) of any data terminal is pressed, terminating any input or output record in progress without completion. For read operations on key indexed files, this bit indicates the return of a key value.

3.3.1.9 BITS 8 - 15. This field contains the Logical Unit Number (LUN) for the I/O Operation as specified by the calling program.

3.3.2 WORD 1

PRB Word 1 contains flags that are set by the user program, plus an opcode for the I/O operation as specified by the user program. The opcode appears in bits 8 through 15 of word 1. The flags appear in bits 0 through 7 of word 1. The function of these flag bits varies with the time that the bit is set and the operation being executed. The following paragraphs describe the function of the flags under the three possible circumstances.

3.3.2.1 OPEN TIME FLAGS. If the calling program sets a user flag in word 1 when opening an I/O device or a file, the operating system assigns the following definitions to the flags:

- Bit 0 - Initiate I/O: The calling program sets this bit to indicate that it is making an Initiate I/O call. When clear, this bit indicates that the program is making an Execute I/O call.
- Bit 1 - Return on Uncorrectable Severe Errors: The calling program sets this bit to prevent the system from aborting the program if an uncorrectable severe error occurs during an I/O operation with the opened device. Instead, the system returns control to the calling program to recover from the error.
- Bit 2 - Return on Correctable Error: The calling program sets this bit to prevent the system from asking for operator assistance if a correctable severe error occurs during an I/O operation with the opened device. Instead, the system returns control to the calling program with bit 1 in word 0 set to recover from the error.
- Bits 3 through 6 - These flags are unused during an open call.
- Bit 7 - Open for Exclusive Access: The calling program sets this bit to indicate that the file or device being opened by the call cannot be shared, but must remain in exclusive control of the calling program until released.

3.3.2.2 RUNTIME I/O FLAGS. If the calling program sets a user flag in word 1 at runtime when requesting an I/O operation (Opcodes 00 through 29), the operating system assigns the following definitions to the flags:

- Bit 0 - Initiate I/O: The calling program sets this bit to indicate that it is making an Initiate I/O call. When clear, this bit indicates that the program is making an Execute I/O call.
- Bit 1 - Output with Reply: If the calling program sets bit 1 and the I/O device is a data terminal, the terminal performs an output, usually a question, and waits for a reply. If multiple programs are using a data terminal, the reply is given to the correct program. If the I/O is assigned to a device other than a data terminal, setting bit 1 produces an input operation only.
- Bit 2 - Automatic Record Termination: When bit 2 is set, data terminal input is automatically terminated upon reaching the input record length. Normal termination occurs when a carriage return is pressed.
- Bit 3 - Formatted USASCII Output or Suppress Bell on Input: When set during an output call, this bit instructs printing devices to use the first word of the output record as form control instead of data.

  Table 3-1 lists these form control characters. This function applies only to a Write USASCII output. When this bit is set during an input call, it prevents the bell on the input terminal from sounding.
- Bit 4 - Suppress CR/LF Echo on Input: Setting bit 4 prevents the data terminal from echoing an input of CR/LF.
- Bit 5 - File Write Verify: Setting this bit enables verification of disc data after writing. Use of this bit is described in the File Management Documentation in Section IV of this manual.

Table 3-1. USASCII Format Control Word

Bit(s)	Definition
0-11	Reserved for future expansion. Should all be 0.
12	0 - Format before record. 1 - Format after record.
13	0 - No carriage-return 1 - Carriage-return
14	0 - No line feed 1 - Line feed
15	0 - No form-feed or second line feed 1 - Form-feed or second line feed (depending on Bit 14)

- Bit 6 - Device Unlock: Setting this bit allows other jobs to use a shared device that has been previously locked to the calling program.
- Bit 7 - Device Lock: Setting this bit prevents other jobs that may be sharing the device from using the device until the device is unlocked. If both bit 6 and bit 7 are set concurrently, the device is locked until the I/O operation is complete.

3.3.2.3 RUNTIME FILE FLAGS. If the calling program sets a user flag in word 1 at runtime when requesting a file operation (opcodes 100 through 119), the operating system assigns the following definitions to the flags:

- Bit 1 - Initiate I/O: The calling program sets this bit to indicate that it is making an Initiate I/O call. When clear, this bit indicates that the program is making an Execute I/O call.
- Bit 1 - Key Specified: When set, this bit indicates that the calling program has specified a key to locate the desired file. The address of that key is contained in word 4 of the PRB.
- Bit 2 - Key Recovery Desired: Setting this bit instructs the operating system to recover a key and place it in the location specified in word 4 of the PRB if the record accessed has a key and bit 1 of PRB Word 1 (Key Specified) is not set.
- Bit 3 - Return Record Size Only: Setting this bit transfers the length of the accessed record into the Data Record Length field of the PRB (word 2). No data transfer occurs.
- Bit 4 - Unused for file management.
- Bit 5 - File Write Verify: Setting this bit enables verification of disc data after writing. Use of this bit is described in the File Management Documentation.
- Bit 6 - File Unlock: Setting this bit allows other jobs to use a shared file that has been previously locked to the calling program.
- Bit 7 - File Lock: Setting this bit prevents other jobs that may be sharing the file from using the file until the file is unlocked. If both bit 6 and bit 7 are set concurrently, the file is locked until the I/O operation is complete.

3.3.3 WORD 2

Word 2 contains the data record length in characters for the I/O operation. Depending upon the type of I/O call, this field has various interpretations. For an open or change record length call (opcodes 7 and 12, respectively), this word sets the input record length for the logical unit. This record length limits the number of characters stored on subsequent input calls. For an input call, the I/O routines load this word with the number of characters actually contained in the input record. For an output call, this word contains the actual output character count. For utility operations such as back space or forward space, this word indicates the number of operations to perform. At the termination of these utility operations, the system returns the number of operations that were not performed (that is, Word 2 contains "0" if the operation was successfully completed, and a number greater than zero if the system encountered an EOF or BOF before reaching the prescribed number of backspace or forward space operations). For operations on an AD/DA device, this word is zero.

When a return on error is specified (Bit 1, Word 0 set) and an error does occur during the specified operation, this word contains an error code greater than 200 (Appendix A describes the error codes for the system). When this occurs, the word must be reset before issuing another I/O request using this PRB. Table 3-2 summarizes the functions of PRB Word 2.

3.3.4 WORD 3

Word 3 contains the data buffer address which is the starting address of the logical data buffer for input and output operations.

An open call causes the device attributes word to be placed in word 3. Therefore, the data buffer address must be placed in word 3 following an open call.

Table 3-2. PRB Word 2

Case	Word 2	User/System Set
Open Change record length	Input record length specification (limit of number of characters transferred on subsequent input calls)	User
Input	Input record character count	System
Output	Output record character count	User
Back/forward space	Number of operations to perform	User
	Number of operations unperformed	System
Return with error	Unchanged if no error occurred during operation	
	Error code >200 if error occurred during operation	System
Data Module	Output data	User
AD/DA Converter	Zeros	User

3.3.5 WORD 4

Word 4 contains a key address that applies only to key indexed or relative files. Section IV of this manual explains disc files.

3.4 FUNCTION OF SPECIFIC OP CODES

Before a program can perform an I/O operation to a device or file, the program must open the device or file. An open call prepares the device or file to do an I/O operation. After the program finishes with the device or file, it should close the device or file. Closing a device or file does not unassign it from the job; it can be re-opened later and used again. Closing does initiate proper end-action for the device to ensure that no data is lost. Devices are opened and closed using I/O supervisor calls.

When the program makes an open call, the I/O routines place the device attributes word in PRB Word 3. Table 3-3 summarizes the device attributes for standard 980 peripherals. Table 3-4 defines the function of I/O operation codes (op codes).

Table 3-3. Device Attributes Word after Execution of an Open Call

Bits	Value	Device Attribute
0	1	System console
1	1	Dummy device
2	1	Can be rewound
3	1	Can be forward spaced
4	1	Can be back spaced
5	1	Printing device
6	1	Model 733 ASR cassette
7	1	Data terminal or CRT
8	1	Disc
9	1	Input device
10	1	Output device
11	1	USASCII transmission
12	1	Binary transmission
13	1	Reserved
14-15	00	Peripheral device
14-15	01	Linked sequential file
14-15	10	Relative record file
14-15	11	Key indexed file

Table 3-4. Input/Output Opcodes

Decimal Opcode	PRB Words	Operation	Function
00	4	Read USASCII	One logical record is read from the specified LUN and stored in memory at the specified buffer address. The characters are packed two-per-word and the maximum number stored does not exceed the data record length specified in the last open or change record length call. The actual number of characters stored is returned in PRB Word 2, and may be less than or equal to the requested input data record length. Any needed conversion to obtain internal USASCII representation of the data is performed. An end of file record, when detected, will set the EOF BIT. The most significant bit of all USASCII characters is equal to a 1 in memory.
01	4	Read Binary	One logical record is read from the specified LUN and stored in memory in a manner similar to that described for Read USASCII. A character in this case is an 8-bit byte, and any necessary data conversion to obtain the binary format is performed. End of file records are detected and cause the EOF BIT to be set.
02	4	Write USASCII	One logical record is transferred from the specified buffer address to the indicated LUN. The number of characters transferred is specified in PRB Word 2, and the data is packed two characters-per-word. If the formatted USASCII record bit (PRB Word 1 bit 3) is a 1, the printing DSR's

Table 3-4. Input/Output Opcodes (Continued)

Decimal Opcode	PRB Words	Operation	Function
02 (Cont'd)			will interpret the first word in the buffer as form control. Any necessary conversion from the internal USASCII representation to the medium storage format is performed.
03	4	Write Binary	One binary record is transferred from the specified buffer address to the indicated logical unit. The number of characters transmitted is taken from PRB Word 2. Any necessary conversion from binary to the medium storage format is performed.
04	2	Rewind	The physical device or sequential disc file is positioned at the beginning of the medium.
05	3	Back Space Record	The number of logical records specified in PRB Word 2 are skipped in the reverse direction. When an end of file record or beginning of medium status is detected, the operation stops with the appropriate PRB bit set and the operation count decremented. If an end of file caused the stop, the medium is positioned in front of the end-of-file record.
06	3	Forward Space Record	The number of logical records specified in PRB Word 2 are skipped in the forward direction. When an end of file record or end of medium status is detected, the operation stops with the appropriate PRB bit set and the operation count decremented. If an end of file caused the stop, the medium is positioned to the beginning of the record that follows the end of file record.

Table 3-4. Input/Output Opcodes (Continued)

Decimal Opcode	PRB Words	Operation	Function
07	4	Open	Initialize the logical and physical devices. PRB Word 2 sets the maximum input record length. The device attributes are returned in PRB Word 3.
08	4	Open Rewind	The I/O device is opened and rewound as previously described.
09	2	Close	The necessary functions to terminate I/O to a device are performed.
10	2	Close, Write EOF	An end of file record is written and the device is closed.
11	2	Write EOF	An end of file record is written.
12	3	Change Record Length	The maximum input record length is changed as specified in PRB Word 2.
13	3	Read Device Status	The Device Status Word is placed in PRB Word 2. The Device Status Word is device dependent and is individually described with each DSR in this section.
14	3	Back Space File	The number of files (as delimited by end of file records) specified in PRB Word 2 are skipped in the reverse direction. If the start of the medium is encountered, the operation stops with the beginning-of-medium flag set and the operation count decremented. Otherwise, the end of file bit is set. The medium is always positioned at the beginning of the first data record in a file. A backspace of one file remains within the current file so that if it is preceded by another backspace file it does nothing. A backspace two files

Table 3-4. Input/Output Opcodes (Continued)

Decimal Opcode	PRB Words	Operation	Function
14 (Cont'd)			actually skips one file. A back-space record can be used to cross an end of file record in the reverse direction.
15	3	Forward Space File	The specified number of end of file records are skipped in the forward direction. If the end of the medium is encountered, the operation stops with the end of medium bit set and the operation count decremented. Otherwise, the end of file bit is set.
16	2	Unload	Magnetic tape and cassette units are rewound and unloaded. Magnetic tape units are placed off-line.
17-18		Reserved	Ignored by all DSR's.
19/20		Write/Read Direct	Device dependent calls supported by some DSR's for transfer without data conversion. Data formats are described with individual DSR description in this section.

3.5 INITIATE AND EXECUTE I/O CALLS

The calling program sets bit 0 in PRB Word 1 to signify Initiate I/O call and clears bit 0 to indicate Execute I/O call. The Execute call suspends the user program until the entire I/O operation is completed, making the I/O appear to be a single, instantaneous operation of the calling program. In a system with many programs executing, the Execute call does not degrade total system efficiency since other programs are executed during the I/O call processing time. When one or a small number of programs are running in the system, the computer is substantially idle during the I/O transfers.

If the Initiate/Execute Bit is a 1, the system returns to the program for further execution immediately following an I/O SVC. A program that is frequently used and has high I/O activity can use the Initiate Call to increase program throughput and attain maximum speed from I/O devices. The PRB from which an initiate I/O call has been made and its associated data buffer should not be modified until the I/O operation has been completed. The program can monitor the PRB Busy Bit (Word 0, bit 0) to detect completion of the I/O (if the Busy Bit is a 1, I/O is not complete). Doing an Initiate I/O call to a device while a previous call to the same device remains incomplete suspends the job until the first call is complete.

If an Initiate I/O call has been issued and processing is complete to the point that the I/O operation must be complete in order to continue, use the Wait for I/O Complete supervisor call to suspend the program. This SVC converts the Initiate call that shares the same PRB to an Execute call.

A typical Wait for I/O may look as follows:

```
SAP R2LC
                                                                    SHEET  0001

                  0001            IDT  SAMPLF
          002B    0002  WAIT      FQU >2B
                  0003  SVC       OPD  >F800,8                DEFINE SVC
                  0004  *
                  0005  *
  0000    1800    0006            @LDM =ARGLST                LIST ADDR TO MREG
P 0001    0003
  0002    F82B    0007            SVC  WAIT                   EXFCUTE CALL
                  0008  *
                  0009  *
                  0010  ****************************************
                  0011  *
  0003    0001    0012  ARGLST DATA 1,PRB                     NUM ARGS,ADDRS
P 0004    0005
                  0013  *
  0005    0022    0014  PRB       DATA >0022                  LUND >22
  0006    9002    0015            DATA >9002                  WAS FRMTTD WRITE
  0007    000A    0016            DATA 10                     8 CHR+FMT CNTRL
P 0008    0009    0017            DATA BUFFER                 ADDR OF DATA
                  0018  *
                  0019  *
  0009    0006    0020  BUFFER DATA >0006,'SAMPLE.'
  000A    D3C1
  000B    CDD0
  000C    CCC5
  000D    AFFF
                  0021  *
          0000    0022            END
```

3.6 STANDARD USASCII RECORDS

DX980 uses two types of USASCII records, formatted and unformatted. An unformatted record is a versatile record that is used to take advantage of special characteristics of a device. Therefore, unformatted records are usually installation and device dependent, whereas formatted records rely on the DSR to allow for specific device characteristics.

Formatted USASCII records use the first word of the record to control physical device formatting on printing devices. To output an 80-character record in the formatted USASCII mode, the program must make a call for 82-characters. The extra two characters are the first word that contains format control. Similarly, when an 80-character formatted record is read, the program must make an input call for 82-characters to ensure that the entire record is read. The program can ignore the first word.

To use the formatted USASCII mode, bit 3 of PRB Word 1 should be set to a 1. Bit 3 is meaningful only for a Write USASCII operation (Opcode 02) that is directed to a printing device. Storage devices write the format control character at the beginning of the data without modification. However, always set bit 3 to ensure device independence if the designated output device is changed.

The format control is not punched on paper tape, since that medium assumes a one-record-per-line format regardless of the format specified.

3.7 I/O ERRORS

Three types of errors are possible when executing an I/O operation: logical errors, severe errors, and fatal errors. Two of these types, logical and fatal errors, result from an error in the calling program. The other type, severe, results from a malfunction of the I/O device. When severe and fatal errors occur, the operating system prints an error message on the system console. This message contains an error number that identifies the type of error and the reason for the error. Table 3-5 lists the error numbers for I/O errors, their level of severity, and the cause of the error message. The following paragraphs describe the three types of errors.

Table 3-5. I/O Errors

Error Number	Severity	Description
201	Severe Correctable	Device Not Ready
202	Severe Uncorrectable	Controller Error
203	Severe Correctable	Data Error
204	Severe Uncorrectable	Controller Busy Error
205	Severe Correctable	Write Protect Error
206	Severe Uncorrectable	End of Record Sequence Error
207	Severe Uncorrectable	Read-After-Write Error
208	Severe Correctable	Offline
209	Fatal	Illegal I/O Opcode

3.7.1 LOGICAL ERRORS

Logical errors occur only during file management operations and are the result of an error in the user call. For example, a user call to create a record with key when the key already exists in the file. This type of error is not catastrophic to execution of the program. Therefore, when a logical error occurs, the operating system returns control to the calling program so that it can take an alternate course of action. When the system returns control to the program, it also sets the Logical Error and Error flags in the PRB (word 0, bits 6 and 1), and places the error number of the error in the Record Length field of the PRB (word 2).

3.7.2 SEVERE ERRORS

Severe errors occur during I/O operations and are the result of a status condition of the peripheral device that prevents it from performing the requested I/O operation. The error condition may be correctable through operator intervention, or it may be uncorrectable.

3.7.2.1 CORRECTABLE. A correctable severe error results from a status condition of the I/O device that the operator can fix. For example, a printer that is out of paper or a card reader offline causes a correctable error if the system can detect the required status condition. When such an error occurs, the system prints a message on the system console asking the operator to decide if he can correct the malfunction. The operator responds either by correcting the condition and then entering the response YES, or by entering the response NO. If the operator enters a NO response, the error becomes uncorrectable. If the operator enters a YES response, the system retries the operation.

The user can choose to bypass the operator notification step and return directly to the calling program when a correctable error occurs. Setting PRB word 1, bit 2 during the SVC that opens the I/O device selects this option. If the site does not have a full time operator, this option allows the user to select an alternate I/O device rather than wait for the operator to correct the malfunction on the original device.

3.7.2.2 UNCORRECTABLE. A severe error becomes uncorrectable if the operator enters a NO response to the system's request for a correctable error, or if the faulty status indication from the device is the result of an inherently uncorrectable error such as failed components in the peripheral interface. Uncorrectable severe errors abort the associated program. However, the user can choose to avoid this outcome by setting PRB word 1, bit 1, during the SVC that opens the I/O device. Setting this bit instructs the operating system to return control to the user program if an uncorrectable severe error occurs. When selecting this option, the user program must allow for alternate devices or remedial routines to cope with severe errors.

3.7.3 FATAL ERRORS

Fatal errors are the result of errors in the user program that cannot be reconciled by returning control to the program. For example, a call to read information from a line printer is impossible to execute. Since the error is part of the program or job control coding, the program is unable to adjust for the error. Fatal errors abort the associated program. This outcome cannot be avoided.

		Response From Device										
Opcode	Operation	Dummy	Full Duplex Terminal	Paper Tape Reader	Paper Tape Punch	Card Reader	Magnetic Tape	Line Printer	Tape Cassette	Sequential File	Relative Record File	Key-Index File
00	Read USASCII	Returns EOF	Responds (/* = EOF)	Responds [1], [2] (/* = EOF)	Error	At least 1 card read (/* = EOF) [4]	Responds (Error if after output) [6]	Error	(Error if after output)(/* = EOF) [6]	(End of medium if after output) [7]	Responds	Responds [11]
01	Read Binary	Returns EOF	Error	Responds [2] (/* = EOF)	Error	At least 1 card read (/* = EOF) [4]	Responds (Error if after output) [6]	Error	(Error if after output)(/* = EOF) [6]	(End of medium if after output) [7]	Responds	Responds [11]
02	Write USASCII	Responds	Responds	Error	Responds (/* = EOF)	Error	Responds	Responds	Responds (/* = EOF)	Responds [7]	Acts as a replace/add function	Acts as a replace/add function [10]
03	Write Binary	↕	Error	↕	Responds (/* = EOF)	↕	↕	Error	Responds (/* = EOF)	Responds [7]		
04	Rewind	↕	↕	↕	Error	↕	↕	Generates form feed	Responds	Responds	Responds	Responds
05	Back space record	↕	↕	Error	↕	Error	Responds	Error	Responds	Responds	Responds: EOF detection not possible, since not allowed	
06	Forward space record	↕	Error	Responds [2]	Error	Responds [2]	Responds (Error if after output) [6]	Error	Responds [6]	Responds [8]	Responds: EOF detection not possible, since not allowed [8]	
07	Open	↕	CR/LF output	Responds	Responds	Responds	Responds	Responds	Responds	Responds [9]	Responds	Responds
08	Open-rewind	↕	CR/LF output	↕	Responds (Punches leader)	↕	↕	Generates form feed	↕	Responds	↕	↕
09	Close	↕	Responds	Responds	Responds	Responds	↕	Responds	↕	↕	Responds	Responds
10	Close-write EOF	↕	LF 3 lines	Error	EOF Followed by trailer	Error	↕	Generates form feed	↕	↕	Close, but does not write EOF	
11	Write EOF	↕	LF 3 lines	Error	EOF Followed by trailer	Error	↕	Generates form feed	↕	↕	Error	Error
12	Change record length	↕	Responds	Responds	Ignores	Responds	↕	Ignores	↕	↕	Responds	Responds
13	Read device status	↕	Responds	Ignores	Ignores	Responds	↕	Responds	↕	↕	Responds	Responds
14	Back space file	↕	Error	Error	Error	Error	Responds	Error	Responds	↕	Response results in begin or end of medium	
15	Forward space file	Responds	Error	Responds [2]	Error	Responds [2]	Responds [6]	Error	(Error if after output) [6]	Responds		
16	Unload	Ignores	Ignores	Ignores	Ignores	Ignores	Responds	Ignores	Responds	Ignores	Ignores	Ignores
17	Not assigned	↕	↕	↕	↕	↕	Ignores	↕	Ignores	↕	↕	↕
18	Not assigned	Ignores	Ignores	Ignores	Ignores	Ignores	Ignores	Ignores	Ignores	Ignores	Ignores	Ignores
19	Write direct	Responds	Responds [12]	Error	Error	Error	Error	Error	Error	Error	Error	Error
20	Read direct	Responds	Responds [13]	Responds [3]	Error	Responds [5]	Error	Error	Error	Error	Error	Error

NOTES:

1 Ignores blank frames and delete (rubout) frames.

2 End of medium not detectable.

3 Reads specified number of frames and stores them packed as received into user buffer.

4 DSR translates invalid columns to valid characters without error indication.

5 Stores 12-bit card image, right-justified within words of user buffer. Character count = 160 for complete card.

6 End of medium = physical end; logical end not indicated.

7 End of medium = current position when op complete; performance of op when not at EOM destroys data from current position to EOM.

8 EOM status possible with decremented record count returned.

9 If opened for exclusive access, position is retrieved from disc when opened and stored on disc when closed.

10 The new record does not have an index key. A replace operation occurs if not at end of medium and both the old record and the index key (if any) are deleted.

11 Options allow recovery of both data record and index key, or functional deletion of both data record and index key from file.

12 All characters are written directly from the buffer; caution should be exercised when the device has paper tape or cassette attached.

13 This operation expands the read ASCII key to include a set of control characters.

Table 3-6. Device Response to Sequential I/O Commands

3.8 INDIVIDUAL DEVICE CHARACTERISTICS

All sequential I/O devices and sequential disc files restrict the operations that can successfully be performed with them. For example, devices capable of both reading and writing cannot arbitrarily switch between Read and Write modes. Other restrictions vary from device to device. Do not close and re-open a device to circumvent these restrictions since the results will vary for each device. Table 3-6 catalogues responses to I/O commands, including read-after-write restrictions. Most devices represent an end of file with a /* record, but some devices do not. Because of this inconsistency, always use the end of file command in lieu of a /* data record.

3.8.1 DATA TERMINAL AND CRT DEVICES

The Data Terminal Device Service Routine (DSR) operates with any of several devices interfaced using the Communications Module (TI Part Number 966637). These devices include Texas Instruments Models 730 and 733 Data Terminals, Model 912 CRT, teleprinter devices similar to the 33 ASR or 33 KSR, and the Hazeltine 2000 CRT. The DSR operates differently with each of these devices to compensate for operating variations of the devices. The DSR allows input to come from the paper tape reader device or the teleprinter type device. The DSR echoes input characters to the terminal as they are entered. For this reason the DSR does not use the terminals in true full-duplex mode. Performing a read direct operation with the full duplex terminal expands the read USASCII capability to accept the control code defined in table 3-7.

CAUTION

Do not press the keys on the keyboard during a data transfer using either the tape cassette of an ASR 733 or the paper tape of an ASR 33. The printer will not respond during the data transfer, and pressing the keys may result in loss of data.

3.8.1.1 INPUT/OUTPUT OPERATION. The PRB of the calling program directs output to the data terminal. If the formatted USASCII mode is specified for output, the DSR adds carriage and form control as directed in the first word of the record. An input call rings the terminal bell unless PRB Word 1, bit 3 (Suppress Bell) is set. The DSR accepts input characters from the keyboard until a carriage return is selected. At that time the DSR echoes a carriage return - line feed to the printer. Regardless of how many characters are typed, the number of characters stored is limited by the record length supplied in the open SVC. Two variations of this input operation are available with special PRB bits. If the Automatic Terminate Bit is set (PRB word 1, bit 2), the input terminates on a carriage return or full buffer, whichever occurs first. The second option suppresses echo of the carriage return - line feed when the record is terminated. The user selects this option by setting PRB word 1, bit 4. All input characters are then echoed to

Table 3-7. USASCII Control Characters

Hexadecimal Code	Control Function	Control Character	Valid Character on Read Direct	Postable	System Special
00	NULL	Shift/CTRL/p		X	
01	SOH	CTRL/a	X		
02	STX	CTRL/b		X	
03	ETX	CTRL/c	X		
04	EDT	CTRL/d	X		X(on output)
05	ENQ	CTRL/e	X		
06	ACK	CTRL/f		X	
07	BEL	CTRL/g		X	
08	BS	CTRL/h			X
09	HT	CTRL/i			X
0A	LF	CTRL/j			X
0B	VT	CTRL/k		X	
0C	FF	CTRL/l	X		
0D	CR	CTRL/m			X
0E	S0	CTRL/n			X
0F	SI	CTRL/o			X
10	DLE	CTRL/p	X		
11	DC1	CTRL/q	X		
12	DC2	CTRL/r	X		
13	DC3	CTRL/s	X		
14	DC4	CTRL/t	X		
15	NAK	CTRL/u			X
16	SYN	CTRL/v	X		
17	ETB	CTRL/w	X		
18	CAN	CTRL/x	X		
19	EM	CTRL/y	X		
1A	SUB	CTRL/z			X

Table 3-7. USASCII Control Characters (Continued)

Hexadecimal Code	Control Function	Control Character	Valid Character on Read Direct	Postable	System Special
1B	ESC	Shift/CRTL/k			X
1C	FS	Shift/CTRL/l			X
1D	GS	Shift/CRTL/m	X		
1E	RS	Shift/CTRL/n	X		
1F	US	Shift/CTRL/o			X

the printer as they are received from the keyboard until the buffer is filled. Special characters are not placed in the buffer nor echoed to the printer. Tab and Escape also terminate an input record as described below.

3.8.1.2 LINE EDITING. The operating system provides several online editing features as follows:

- Delete Input Characters - Backspace (CTRL/H) and ←— (left arrow) deletes one input character. On Silent 700's, the print head backspaces and the DSR supplies a line feed when the first valid character is entered. CRT's perform a left-cursor. Teletype machines type a backward slash (\).
- Delete Input Record - Rubout deletes an entire input record which may then be re-entered.
- List Input Record - CTRL/N lists the input record as currently stored.
- Terminate I/O - The escape key terminates any active I/O operation. The Terminate-on-Escape Bit in the PRB system flag area (Word 0, Bit 7) is set.
- Tab - Tab is a special character than can be detected by the user program during an input call. The input is immediately terminated and the tab is placed in the data buffer, but the input record length in the PRB does not reflect its presence.
- System Mode - If the data terminal is the system console, CTRL/O puts the data terminal in system mode. When in system mode, any user output assigned to the terminal is held in a queue until the terminal returns to user mode.
- User Mode - If the data terminal is the system console, CTRL/U puts the data terminal in user mode. When in user mode, any system output assigned to the terminal is output without regard to mode.

NOTE

The following features insert or delete text within a record and are convenient for use with a CRT terminal only.

- Increase Character Count - Line Feed or ↓ (down cursor) increases the input character count by the length of a line if it does not exceed the record size specified in the open SVC. Line length is specified in the Physical Device Table (PDT).
- Decrease Character Count - ↑ (up cursor) decreases the input character count by the length of a line. Line length is specified in the PDT.
- ⟶ (right cursor) increases the input character count by 1 if the input character size does not exceed the record size specified in the open SVC.
- Freeze the output - While the terminal is outputting, pressing the control D key halts the current output. Pressing any other key while in this mode will reactivate the output.

3.8.1.3 INTERACTIVE EXTENSIONS. Four device-dependent features are included in the DSR for data terminal input operations. They are activated by four separate flag bits in the PRB and may be used in several combinations. These flags do not apply to, and are ignored by, other device service routines. The features are:

- Do not ring bell on input call.
- Do not echo carriage return/line feed on input call.
- Output with reply operation.
- Terminate input automatically when the data record length is reached.

3.8.1.4 OUTPUT WITH REPLY. The Data Terminal DSR provides a special output/input function for question and answer operations. Setting word 1, bit 1 (output with reply) in the input PRB directs the system to locate an output PRB immediately following the input PRB. This bit should not be set in the output PRB. The output (second) PRB is executed first. The responding input is executed immediately, even if multiple programs are using the same data terminal. If the I/O is assigned to a device other than a terminal, it is considered an input operation only. All other DSR's ignore the output with reply PRB bit and execute only the first (input) PRB.

3.8.1.5 FORM FEED DETECTION. The data terminal DSR detects the form feed function when it is issued by either a formatted output record or a Form Feed Character in the output data stream. The terminal outputs six blank lines when a form feed is detected unless the Home-on-Form-Feed PDT Bit is set. In that case a home cursor command is issued.

3.8.1.6 OPEN/CLOSE OPERATIONS. An Open or Open-Rewind SVC causes a CR/LF to be performed. A Close causes no special action, but a Close-Write EOF writes three blank lines on the terminal. A Write End-of-File command also results in three blank lines.

3.8.1.7 AUTOMATIC LINE CONTINUATION. The DSR can print data records up to twice the carriage size on consecutive lines and without data loss. This feature is subject to two restrictions:

(1) The print head must be at the left margin when the record begins to print

(2) Formatted output must be used

3.8.1.8 FORMATTED OUTPUT. This DSR supports formatted USASCII output. A form feed is defined as six blank lines unless the Home-on-Form-Feed PDT Bit is set. If that bit is set a form feed is interpreted as a home cursor operation. Unformatted I/O is device dependent.

3.8.1.9 CARRIAGE RETURN DELAY. Each different terminal device requires from 1 to 5 null characters following carriage return commands to allow for print head travel time. The DSR checks the Physical Device Table (PDT) to determine the type of terminal so that it can generate the proper number of null characters.

3.8.1.10 LINE FEED DELAY AND LINE CLEARING. A Clear to End-of-Line code is sent following line feed operations. For a CRT, this clears the display lines before output is done on a particular line. For other terminals it provides a one character delay. This delay following line feeds is equivalent to a null character.

3.8.1.11 CHARACTER LOSS. Character loss does not occur if unformatted USASCII output with imbedded CR/LF's is used. For Silent 700's sufficient null characters are sent following a carriage return to avoid character loss. For other devices a single null character follows each carriage return. For all devices a clear end-of-line character is sent following a line feed. Unless the device is a CRT, this character is equivalent to a null character. Each line of a CRT is cleared before output is done on that line.

3.8.1.12 REPEAT KEY. The repeat key may be used to transfer a character repeatedly. The number of characters read will be echoed and printed.

3.8.1.13 CTRL/H ON SILENT 700. The print head back spaces one position each time that the back space key is activated until the entire input record is deleted. The first new character echo following a sequence of back spaces is preceded by a line feed.

3.8.1.14 CTRL/H ON CRT. The cursor moves left one position each time that the back space or left cursor key is activated until the entire input record is deleted.

3.8.1.15 CTRL/H ON ASR/KSR 33. A backward slash is typed each time that the back space key is activated until the entire input record is deleted.

3.8.1.16 ERROR CONDITIONS. No task or system errors are generated by the Device Service routine. Read operations with timeout specified may cause data error indication in the PRB. (See paragraph 3.8.1.19). For detectable input errors the DSR discards the characters, echoing a Bell to the terminal. Opcode errors cause an abort condition.

3.8.1.17 DEVICE STATUS WORD. A Read Device Status call reads the status register of the Communications Module and transfers it to the calling program. If bit 0 of the status word is a 1, the status is not valid and must be requested again. Bit 0 can only be set to a 1 if the Communications Module receives a character between the time the DSR is entered for the read status and when the status is actually read. This is an unlikely occurrence, since the window for this to happen is less than 50 μ s. Use of the status bits varies with the type of terminal. Refer to the applicable Terminal User's Guide listed in the Preface of this manual for the specific use of each status bit. The status word bits are assigned as follows:

- Bit 0 - Invalid status if set.
- Bits 1 through 10 - Always zero.
- Bit 11 - Ring indicator.
- Bit 12 - Reverse channel receive.
- Bit 13 - Data carrier detect.
- Bit 14 - Clear to send.
- Bit 15 - Data set ready.

3.8.1.18 PAPER TAPE INPUT. The paper tape reader on the teletype devices may be used to input data as if the data was being typed from the keyboard. The user must momentarily set the start button for each record. Each record must be delimited by a carriage return (CR) and at least three "don't care" characters. All rubout characters at the beginning of each record will be ignored.

3.8.1.19 USER SPECIFIED TIMEOUT ON READ OPERATIONS. Read operations may be timed out by setting bit 5 of the user set flags or the read PRB. If this bit is set, word 4 of the PRB is a pointer to a one-word field containing a right justified, non-zero, 8-bit timeout value in seconds. A >FF specifies no timeout.

The data terminal device must be opened to return on correctable errors. (See paragraph 3.3.2.1.) A timeout will be indicated if the error bit of the system set flag is set and the returned error code is a 210.

3.8.2 MODEL 733 ASR CASSETTE

The Device Service Routine for the 733 ASR cassette handles 1200 baud cassette units. Each cassette is treated as a separate physical device. The DSR is functional only if the 733 ASR is operated in the line mode for USASCII processing. The device must include 1200 baud and remote device control options. The 733 ASR must be interfaced to the computer using the Communications Module. The DSR supports reading USASCII records that have been recorded in the continuous mode.

3.8.2.1 TAPE RECORD FORMATS. Some data conversion is done prior to the recording on tape. The conversions described in this section apply to both Binary and USASCII records. These conversions are performed after initial conversions that vary between Binary and USASCII records.

An USASCII cassette tape record is defined as data followed by a carriage return character. A record is more than one physical tape block long if it is greater than 86-characters (including the CR). The record always begins on a physical block boundry. All USASCII tape records require at least one 86-character tape block for storage. Since a CR determines the end of a physical record, physical USASCII records rarely correspond exactly with logical USASCII records that were output by the DSR. For example, if an USASCII record ends with a carriage return/line feed, the LF becomes the first character of the next tape block following the rest of the record. Since the DSR does not pass the CR or the LF to the user when reading USASCII, this arrangement causes no problems.

All USASCII records require at least one tape block for storage. Short output logical records may tend to overrun the capacity or the 733 ASR recording buffer in the device. For this reason the DSR adds delete characters to all records as necessary so that any two consecutive tape blocks contain at least 60 characters.

3.8.2.2 USASCII RECORDS. To allow offline cassette preparation and playback, cassettes are treated as printing devices instead of storage devices. Therefore, all formatted write USASCII operations expand the format specification into form control characters before writing on tape. Formatted expansion is identical to the data terminal DSR expansion.

Since a carriage return separates records on tape, the DSR appends a CR to the end of any USASCII write operation that does not have at least one CR in the data to be written. The hexadecimal characters 10 through 14 are reserved control characters and are not allowed in USASCII records. If one of these characters is encountered, the DSR converts it to a delete character before transmission to the 733 ASR. A write direct operation is equivalent to a USASCII write operation and is, therefore, not supported. The most significant bit of an USASCII character is not written on the tape. To transfer memory image data to cassette a write binary must be used.

The USASCII read operation passes all recorded characters from hexadecimal 20 to 5F to the caller. USASCII records are terminated by a carriage return on the tape, but the CR is not placed in the caller's buffer. Read direct passes all recorded data, including delete and null characters. It is also terminated by a carriage return and the CR is passed to the caller.

3.8.2.3 BINARY RECORDS. Each word in memory for a binary record is written on the tape as three 7-bit characters. For example, the 16-bit word:

ABCDEFGHIJKLMNOP

It is written on tape in three 7-bit characters as follows:

110ABCD

1EFGHIJ

1KLMNOP

3.8.2.4 END-OF-FILE RECORDS. An end-of-file in both binary and USASCII modes is a record with the first two characters being a (/*) combination, followed by a carriage return, line feed, X-off, and more than one delete (rub-out) character. A write end of file command writes a /, *, carriage return, line feed, X-off, and 256 delete characters.

3.8.2.5 OFFLINE PREPARATION AND PLAYBACK. USASCII records can be prepared offline by typing with the recording cassette in the Line mode (RECORD switch in the LINE position). Any cassette tape recorded using the USASCII write operation can be listed offline. An offline end of file is a "/*" followed by a carriage return. Binary records cannot be prepared or listed offline.

3.8.2.6 TAPE POSITIONING FUNCTIONS. Forward Space Record and Forward Space File commands operate with the DSR reading the cassette at its normal read rate. The Back Space Record command is not implemented. For USASCII records less than 86 characters each, a Block Reverse (DLE, 8) operation is equivalent to a Back Space Record command. A Back Space File command assumes that the cassette was positioned at the beginning of a file when it was opened. The tape then block-reverses to the open point or to the last end of file read in the forward direction from the open point. The Backspace File opcode backspaces only one file. Unload rewinds the tape and leaves the cassette positioned on the clear leader of the tape.

3.8.3 PAPER TAPE READERS

The paper tape readers read 8-level paper tape in either an USASCII, Binary or Direct mode. Support is provided for ASR33 type devices and high speed paper tape readers.

3.8.3.1 RECORD FORMATS. USASCII records are read as one tape frame per character. The most significant bit of each character is set to a binary one regardless of its state on the paper tape. USASCII records may be delimited by either of the following sets of characters:

Reader Off (XOFF), delete, delete

Carriage Return (CR), Line Feed (LF), delete, delete

Records delimited by the second set of characters may have any set of characters between the CR and the LF, and any one character between the LF and the delete, as follows:

CR, X_1, LF, X_2, delete, delete

where,

X_1 is any set of characters not containing an LF

X_2 is any character

Blank, punch on, punch off, and delete (rubout) frames are ignored.

Binary records are read as four frames per word using the conversions shown in table 3-8. Records are terminated by an X-off character.

Read direct (opcode 20) is implemented for the paper tape reader. It operates the same as read USASCII except for the following characteristics:

- The entire 8-bit character is stored unmodified.
- No characters are ignored; deletes and null frames are stored.
- There is no end of record character.
- No end of file record is recognized.
- Read direct paper tape on an ASR33 type device does not always read the number of characters for which the device was opened. The user should depend on the returned (PRB word) character count.

3.8.3.2 ERROR DETECTION. The DSR detects invalid punches during a read binary operation only. An illegal operation aborts the job.

3.8.3.3 END OF FILE RECORDS. In USASCII and Binary modes any record beginning with a slash-asterisk (/*) is detected as an end of file record. For Binary mode the "/*" record is not stored in memory.

3.8.4 PAPER TAPE PUNCH DEVICES

The paper tape punch devices punch data on 8-level paper tapes in an USASCII, Binary, or Direct format. Support is provided for ASR33 type devices and high speed paper tape punches.

Table 3-8. Binary Internal Code to Paper Tape Binary Code

Internal Code	Tape Code Frames
0	00010 000
1	00000 001
2	00000 010
3	10000 011
4	00000 100
5	00010 101
6	00010 110
7	10010 111
8	10011 000
9	00011 001
A	00011 010
B	10011 011
C	00011 100
D	10011 101
E	10011 110
F	00011 111

3.8.4.1 RECORD FORMATS. USASCII records are written as one frame per character. Data is punched directly from the data buffer with no conversions. Each record is terminated with a Carriage Return (CR), Line Feed (LF), reader-off character (X-off), and two delete (rubout) characters, and a punch-off character if the device is a Model 33 ASR. Binary records are punched as four frames per word. The end-of-record indicator is the same as for the USASCII write operation, without the carriage return and line feed. Write Direct punches data directly from memory as in a USASCII write operation, but no end-of-record indicator is added to the punched data. Write Direct is not supported on Model 33 ASR paper tape punches.

3.8.4.2 ERROR DETECTION. The punch does not detect punch errors.

3.8.4.3 END OF FILE RECORDS. An end of file record is defined for both Binary and USASCII modes as a record beginning with "/*". A write end of file punches a "/*" followed by CR, LF, X-off and two delete characters, then a 10 inch blank tape trailer.

3.8.4.4 OTHER OPERATIONS. Open-rewind punches 10 inches of leader on the paper tape punch.

3.8.5 DMAC AND I/O BUS LINE PRINTERS

Characters are printed directly from the user's data buffer. If the formatted USASCII bit in the PRB is set, the first word of the data buffer is used for format control.

3.8.5.1 STATUS. If the printer is in a Not Ready condition (as opposed to offline), the DSR prints a Printer Not Ready message on the system console terminal. Offline status is not detectable.

3.8.5.2 ERRORS. The calling program aborts on illegal operations. If the device is not ready, the DSR generates retry errors.

3.8.5.3 I/O BUS PRINTER. A carriage return precedes any form feed operation. Bold letter records may be output by setting bit 1 in the user set flags of the PRB. Bit 1 instructs the line printer to overprint the line with redundant text.

3.8.5.4 OTHER FUNCTIONS. The page moves to top of form when the following operations are done:

- Rewind
- Open Rewind
- Close and Write EOF
- Write EOF

The printer performs a carriage return and a line feed when it receives an Open command.

3.8.6 CARD READER

For handling formatted I/O records, cards are considered as a hard-copy medium rather than a storage medium. Therefore, do not punch the format control characters when preparing the input deck. In addition the deck must follow an implied one-record-per-card structure. The following restrictions are imposed upon cards as a storage medium:

- An implied CR/LF on every record
- Each blank line is indicated by a blank card
- A form-feed cannot be stored
- A maximum limit of 80-characters per record.

3.8.6.1 RECORD FORMATS. USASCII records actually appear as Hollerith punches on the cards. An USASCII read operation causes the DSR to convert the Hollerith characters to USASCII characters. Table 3-9 lists the characters and their Hollerith and USASCII codes. Binary records are punched two characters per card column as shown in Table 3-10.

Read Direct (opcode 20) reads a card without any data conversion. The data is stored in memory as one card column per word. The word is right-justified with bits 0 through 3 equal to 0. The least significant bit in memory represents card row 12 (top of card). The input record length must be given in characters, or 8-bit bytes, and requires a character count of twice the number of columns to be stored. If the character count is odd, only one byte of the last column is stored. That byte is stored in the most significant half of the buffer word.

3.8.6.2 ERROR DETECTION. The DSR detects errors for timing and invalid punches. A timing error occurs when either an interrupt for one card column is not serviced before the next is read, or when the punches in the card are out of alignment. Not all mispunched card columns are detectable. The DSR detects all invalid binary punches. Mispunched USASCII characters that are not detected as errors are converted to valid USASCII characters.

3.8.6.3 END OF FILE RECORDS. A card with a slash-asterisk (/*) in the first two columns represents an end of file record in both Binary and USASCII modes.

3.8.7 DUMMY DEVICE

The dummy device handles all possible operations. All necessary system set flags are returned. For example, read operations set the end of file bit.

Table 3-9. USASCII Character Internal Code to Hollerith Code Conversion

USASCII Code	Hollerith Code	Character	USASCII Code	Hollerith Code	Character
20	No Punches	SP	40	8·4	@
21	12·8·7	!	41	12·1	A
22	8·7	"	42	12.2	B
23	8·3	#	43	12.3	C
24	11·8·3	$	44	12.4	D
25	0·8·4	%	45	12.5	E
26	12	&	46	12.6	F
27	8·5	'	47	12.7	G
28	12·8·5	(	48	12.8	H
29	11·8·5	)	49	12.9	I
2A	11·8·4	*	4A	11.1	J
2B	12·8·6	+	4B	11.2	K
2C	0·8·3	,	4C	11.3	L
2D	11	—	4D	11.4	M
2E	12·8·3	.	4E	11.5	N
2F	0·1	/	4F	11.6	O
30	0	0	50	11.7	P
31	1	1	51	11.8	Q
32	2	2	52	11.9	R
33	3	3	53	0.2	S
34	4	4	54	0.3	T
35	5	5	55	0.4	U
36	6	6	56	0.5	V
37	7	7	57	0.6	W
38	8	8	58	0.7	X
39	9	9	59	0.8	Y
3A	8·2	:	5A	0.9	Z
3B	11·8·6	;	5B	12.8.2	[
3C	12·8·4	<	5C	0.8.2	\
3D	8·6	=	5D	11.8.2	]
3E	0·8·6	>	5E	11.8.7	↑
3F	0·8·7	?	5F	0.8.5	←

3.8.8 16 INPUT/16 OUTPUT DATA MODULE

This DSR does not support interrupt processing. The 16 I/O Data Module is not compatible with any other I/O device, and has its own functions and op-codes. They are listed in table 3-11. Figure 3-2 illustrates the Data Module PRB.

Table 3-10. Binary Character Internal Code to Binary Card Code Conversion

Internal Code	Card Code	Internal Code	Card Code
Most Significant Digit	Rows 12-11-0-9	Least Significant Digit	Rows 8-1-2-3-4-5-6-7
0	Blank	0	blank
1	9	1	1
2	0	2	2
3	0-9	3	3
4	11	4	4
5	11-9	5	5
6	11-0	6	6
7	11-0-9	7	7
8	12	8	8
9	12-9	9	8-1
A	12-0	A	8-2
B	12-0-9	B	8-3
C	12-11	C	8-4
D	12-11-9	D	8-5
E	12-11-0	E	8-6
F	12-11-0-9	F	8-7
Example: The binary card character for hexadecimal CA is 12-11-8-2.			

3.8.9 AD/DA DEVICES

Figures 3-3 and 3-4 illustrate the PRB for Analog-to-Digital converters and Digital-to-Analog converters, respectively.

3.8.10 MAGNETIC TAPE

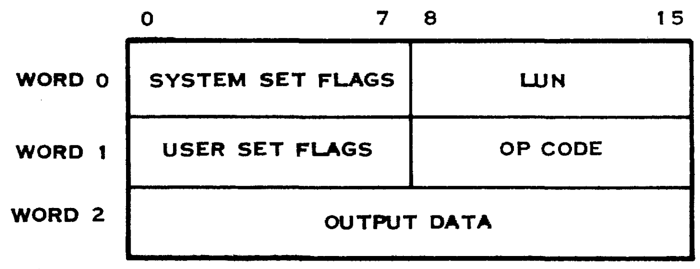
The TI Model 979 Magnetic Tape Unit is a standard peripheral of the minimum DX980 hardware configuration.

This unit is a 1/2 inch, 9 track, IBM compatible format, 800 bits per inch tape drive. It uses an NRZI recording format with a standard fixed speed of 37 1/2 inches per second.

A detailed description of the unit is contained in the Model 979 Tape Transport Operators Manual, part number 216316-9701.

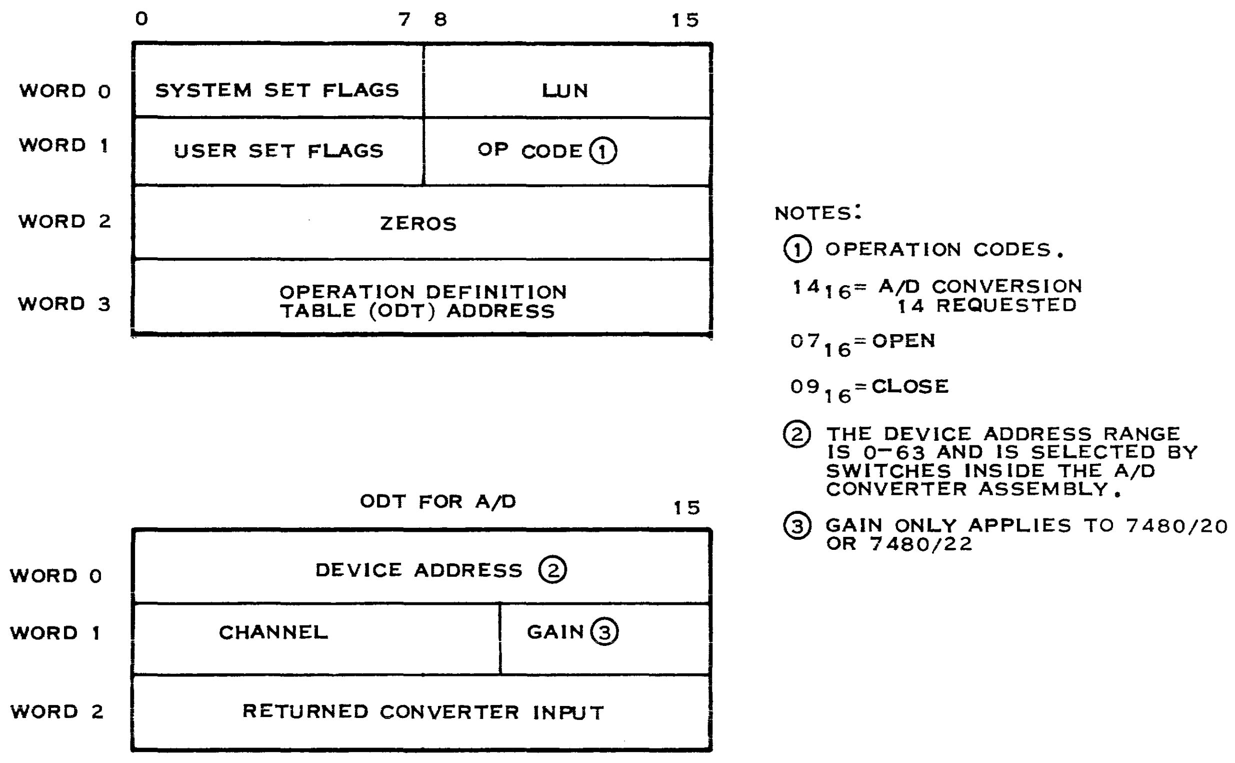
Table 3-11. 16 I/O Data Module Instructions

Op Code	Function
30	Reset - Initialize Data Module logic.
31	Output Word - Transfers a word from computer to Data Module.
32	Output Bit - Transfers a single bit from computer to a specified bit of the Data Module.
33	Read Status - Transfers current Data Module status to computer.
34	Read Data - Transfers a word from Data Module input lines to computer.
35	Read Output Register - Transfers the contents of the Data Module Output lines to the computer.
7	Open - Initialize the Data Module and set-up PRB.
9	Close - Terminate I/O operations.



(A)129984

Figure 3-2. Data Module PRB



NOTES:

① OPERATION CODES.

14_{16}= A/D CONVERSION 14 REQUESTED

07_{16}=OPEN

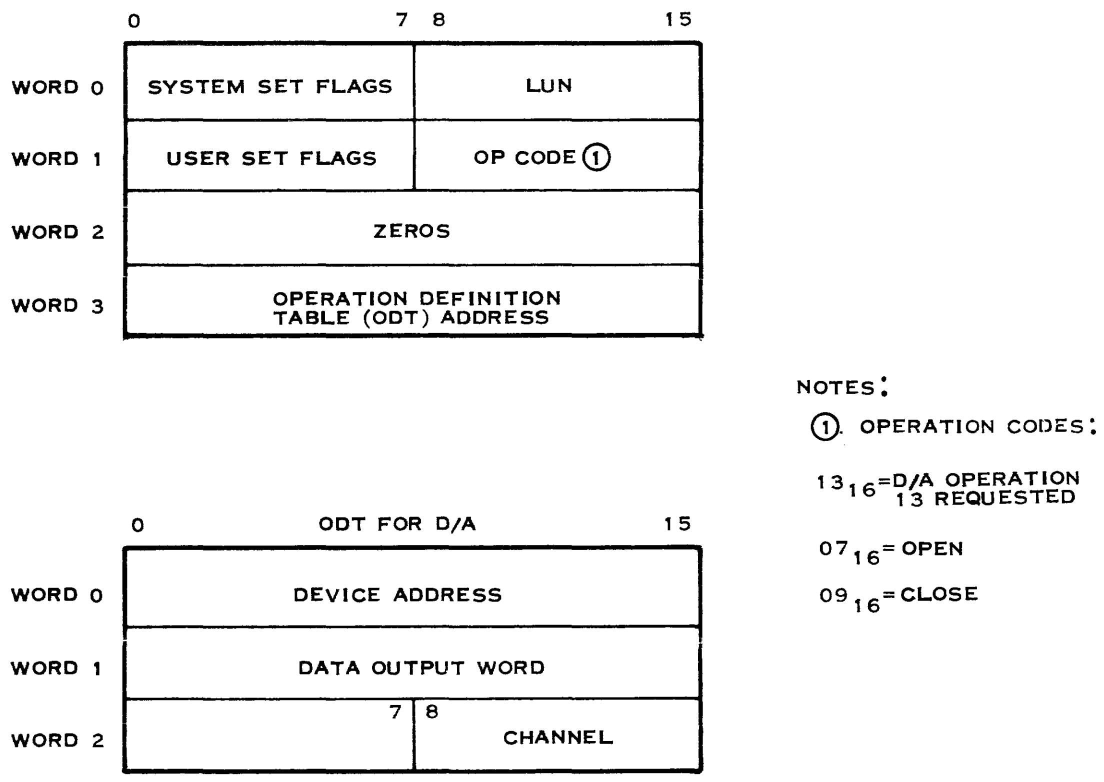
09_{16}=CLOSE

② THE DEVICE ADDRESS RANGE IS 0–63 AND IS SELECTED BY SWITCHES INSIDE THE A/D CONVERTER ASSEMBLY.

③ GAIN ONLY APPLIES TO 7480/20 OR 7480/22

(A)129985

Figure 3-3. Analog-to-Digital Converter PRB



NOTES:

①. OPERATION CODES:

13_{16}=D/A OPERATION 13 REQUESTED

07_{16}= OPEN

09_{16}=CLOSE

(A)129986

Figure 3-4. Digital-to-Analog Converter PRB

SECTION IV
DISC FILE MANAGEMENT

4.1 FILE STRUCTURES

A DX980 file is a logical collection of related data stored on a random access device. A file consists of a number of logical records each containing a collection of related data items that the program treats as a unit. Logical records may also combine to form a physical record. A physical record is a collection of data items that the operating system treats as a unit when transferring data between main memory and the random access device. DX980 supports two types of record transfers: blocked and unblocked. An unblocked file exists if each physical record in the file consists of a single logical record. If the physical records contain more than one logical record, the file is described as blocked.

4.2 FILE HANDLING

When the program issues an I/O call for a file transfer, the file management system intercepts the call. For output operations, the logical record indicated by the Physical Record Block (PRB) is transferred to a physical record buffer. If the logical record transfer completes the physical record, the physical record is transferred to the random access device. Similarly for input operations, a physical record may be transferred from the device to the buffer. The buffer then supplies logical records to the program until all logical records are used. At that time a new physical record is retrieved from the device. Thus, several logical record transfers can be made at memory speed before a single transfer at peripheral speed. This arrangement can minimize transfer time in programs with a high degree of I/O activity.

4.2.1 MEMORY ALLOCATION

The operating system allocates memory space for physical record buffers from the user program's job extension area (<jearea>) when the user opens a file. The memory space is released when the user closes the file. The Job Control Language (JCL) tells the system that a file is blocked and the number of physical record buffers to be allocated. The job area (<jarea>) for the user program provides memory space for logical record buffers. The user can provide this space explicitly by using a Block Starting with Symbol (BSS) directive in an assembly language program or implicitly within the Fortran Input/Output package when the program is run.

4.2.2 FILE INTEGRITY

DX980 maintains file integrity through access restriction and through file locking. When the file is defined or assigned, the user specifies access restrictions for operations on the file of reading, writing, executing and deleting the file. He can restrict access completely for any of the operations (NONE), he can allow access to anyone (ANY), or he can selectively restrict access to either the creator of the file (CREAT) or to those having the proper password (PSWD). Following this initial definition of access restrictions, DX980 enforces them for each type of file operation.

In addition, DX980 provides three levels of file locking:

1. Assigned exclusive
2. Assigned shared - open exclusive
3. Assigned shared - open shared, then lock

The first type provides an exclusive access to the file until the file is deassigned through job step termination if the file is not being passed, or through job string termination if the file is passed from one step to another. Exclusive access can also be removed through runtime resource deallocation. The second type of file locking provides exclusive access to the file until the file is closed. The third type of file lock secures the file on an operation by operation basis. The file is locked by setting a bit in the User Set Flag area of the PRB for that operation. This type of lock secures the entire file rather than a single record. Also, the user is not locked out unless he specifies the lock by setting the bit in the PRB.

4.3 DISC ORGANIZATION

The operating system allocates file space and maintains disc directories to support both DS330 and moving head discs. DX980 uses two levels of directories:

- one master file directory for each disc pack (<volume>)
- a user file directory for each user (<fileid>) on each disc

These directories are standard indexed files with a physical record size of 96 words. For a single master file directory, this indexed file contains keyed entries consisting of file control blocks for each user file directory. The file control blocks are keyed on each valid user <fileid> in the system. A particular user file directory contains keyed entries consisting of file control blocks for each file defined under the user. The file control blocks are keyed on the file name <filenam>.

Each user file directory identifies files by file name (<filnam>). Thus, a user can have several files under a particular user file directory on each disc volume. If an installation has a single disc drive, all packs to be mounted on that drive must also contain the operating system. For multiple drive installations one drive, designated as the system disc, supplies the operating system plus user files. The remaining drives are dedicated completely to user files. The selection of the drive or volume for file storage is a JCL assignment parameter.

The operating system employs two techniques for file allocation: contiguous allocation and noncontiguous allocation. Contiguous allocation places the entire file on consecutive disc tracks (disc tracks are numbered consecutively from 0 through the total number of tracks on the disc; the last track on one disc cylinder and the first track on the next cylinder are numbered consecutively on a moving head disc). Noncontiguous allocation is accomplished dynamically as the file grows on a track-by-track basis. The initial allocation for noncontiguous files is specified as a JCL parameter and may be any number of tracks. The entire initial allocation is assigned to consecutive tracks. As I/O operations add records to the file and the initial allocation is used up, additional tracks are allocated one at a time from any available disc space. As each additional track is used up, another is added until the final allocation limit, specified in JCL as <mtrks>, is reached. The file management system can also start searching for the initial allocation quantity (or total quantity for contiguous allocation) at a particular track <trknum>. This user option minimizes head movement by grouping together files to be processed by a program.

4.4 FILE TYPES

DX980 supports three types of files: linked sequential, relative record and key indexed files. The following paragraphs explain each of these file types.

Table 4-1 summarizes the features of each file type.

4.4.1 LINKED SEQUENTIAL FILES

Linked sequential files are files whose records can only be reached through sequential access. File allocation is noncontiguous. The operations supported on a Linked Sequential File are identical to those outlined in Section III for I/O to sequential devices. A linked sequential file supports an imbedded end of file so that a single linked sequential file can replace a multiple file stack on a sequential access device (for example an entire reel of magnetic tape). The random access capabilities of the disc facilitate the search file operations. A record cannot be inserted between two existing records. Individual records cannot be deleted. An end of medium pointer always follows the last write performed. This end of medium is equivalent to an end of volume on a tape reel.

Table 4-1. Summary of DX980 File Features

Parameter	File Type		
	Linked Sequential	Relative Record	Key Indexed
Access	Sequential	Sequential; Random	Sequential; Random
Allocation	Noncontiguous	Contiguous	Noncontiguous
Logical Order	Chronological	Disc Address	Keyed; alphabetic Nonkeyed; Chronological after key
Key	None	15-bit Binary record number	1-30 byte
Transfer	Blocked	Direct (LR = PR) Blocked (LR<PR)	Blocked
Physical Record Length	Multiple of 32	Multiple of 32	Multiple of 32
Logical Record Length	Variable	Fixed	Variable
Logical/Physical Record Relationship	LR $\lesseqgtr$ PR	LR ≤ PR	LR ≤ PR-8 unkeyed; LR ≤ PR-8-KEY keyed where KEY=key
Logical Record Split Over Physical Record Boundary	Yes	No	No
Inter Job File Position	Last File Position	Beginning of Medium	Beginning of Medium

The logical record length for linked sequential files is always variable. Therefore word 2 of the PRB specifies the record length of each individual record when the record is written. When a record is read, the actual record length is returned in word 2 of the PRB subject to the maximum record length specified when the file was opened. Insofar as the I/O program is concerned, the physical record length of records stored on disc is transparent. If the physical record length is greater than logical record length, the number of actual disc transfers is less than the number of I/O calls. If the physical record length is less than the logical record length, the number of disc transfers will be greater than the number of I/O calls. Figure 4-1 illustrates the composition of a linked sequential file.

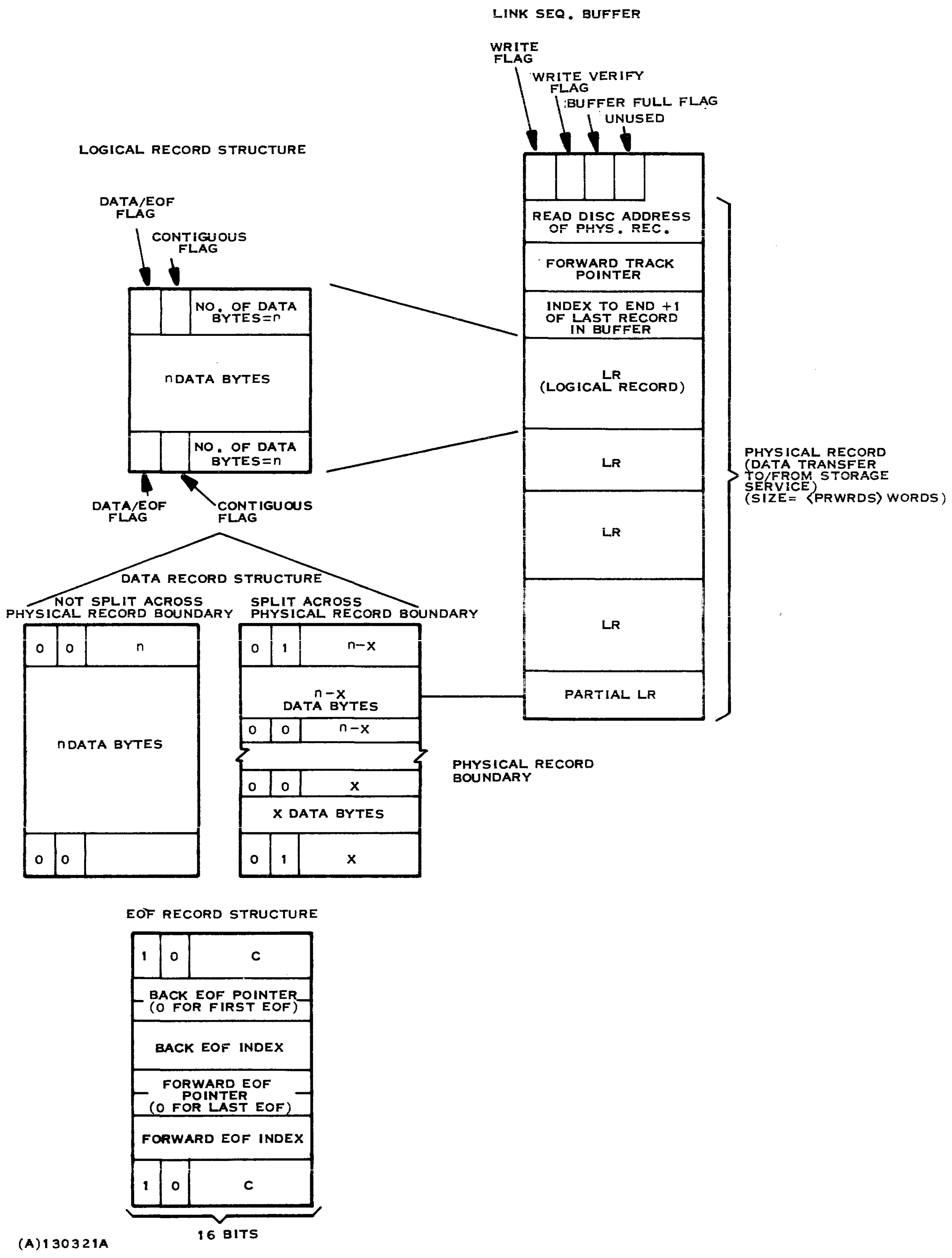


Figure 4-1. Linked Sequential File Parameters

4.4.2 RELATIVE RECORD FILES

A relative record file allows random access in addition to sequential access for locating records within the file. The random access method uses a number to directly specify the numerical position of the record within the file. The first record is designated as record number 0. Records may be added to the file using either access method. Existing records in the file may be changed or read using either access method. A record cannot be inserted between two existing records. Single records cannot be deleted. The operating system does not support an end of file interior to the file.

File allocation for a relative record file is contiguous so that the size of the file must be specified when the file is defined. The entire contiguous data area assigned to the file is partitioned into fixed length logical records. If blocking is specified, the blocking buffer length (physical record length) must be larger than the logical record length. In either case access to a record does not require a directory search. A maximum of one disc access is required to fetch a record by either method and may be none if the record is within a blocked physical record that is in the buffer.

A relative record file is unblocked if the logical record length is equal to physical record length. For unblocked files the file management system does not create an intermediate buffer in the job extension area. Instead, it transfers the data directly to or from the logical record buffer specified by the PRB. This direct transfer requires a disc transfer for each I/O call, and increases the running time of the accessing program, but reduces the memory requirements due to a smaller job extension area. Figure 4-2 illustrates both blocked and unblocked transfers.

The operating system supports the following random access functions:

- Write/Replace using key - Data record with same numeric key to be replaced with new data record.
- Replace using key - same as Write/Replace using key.
- Read using key - Read the logical record as specified by the key.

Any access to a record in the file (read or write, random or sequential method) establishes a logical position that follows the record previously accessed. This logical position has no effect on a subsequent random access operation, but defines the location for a subsequent sequential operation.

4.4.3 KEY INDEXED FILES

A key indexed file allows random access in addition to sequential access for locating records within the file. The operations allowed for both random and sequential access methods are considerably more powerful than those allowed for either linked sequential or relative record files. Key indexed files are noncontiguous.

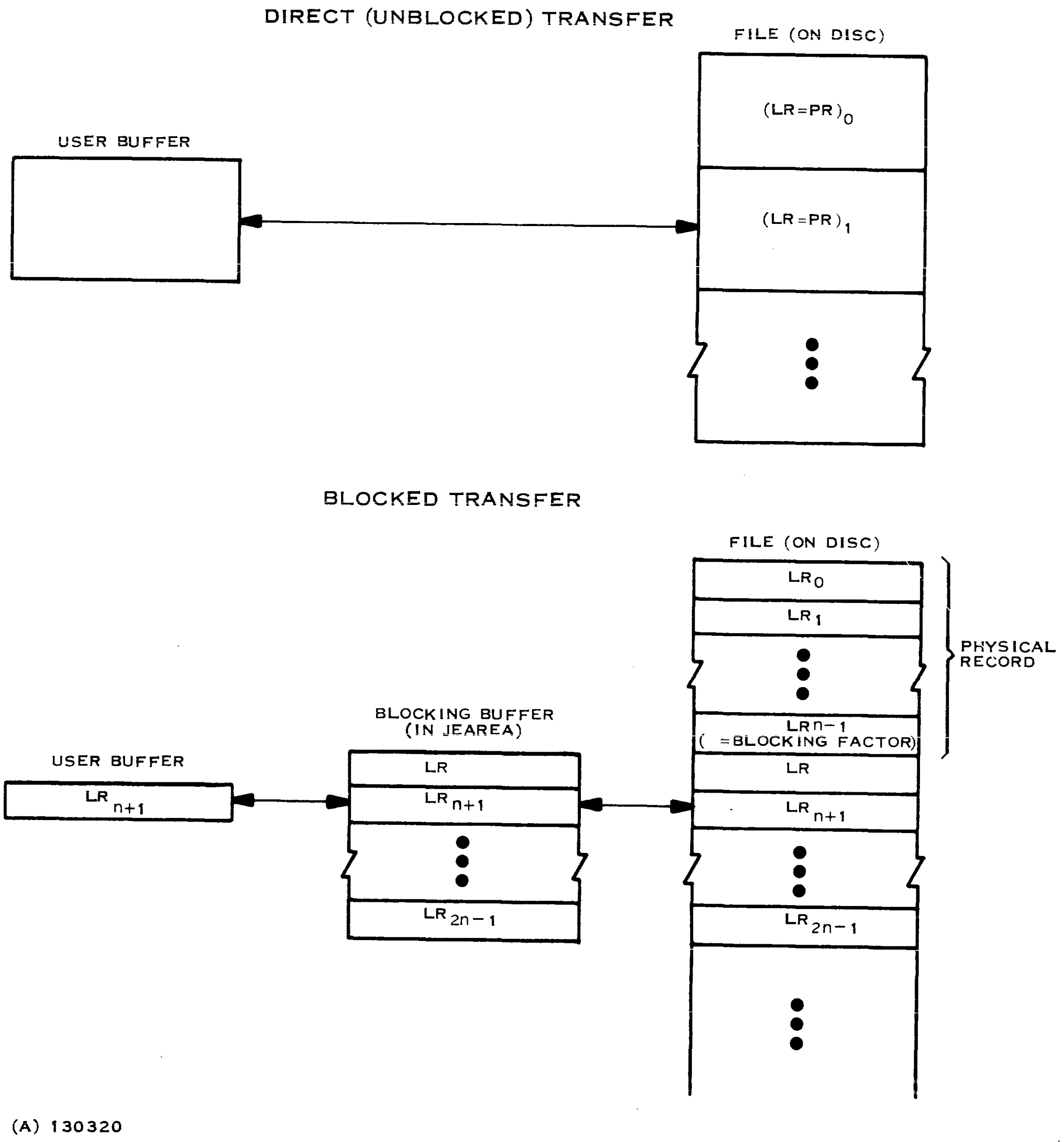


Figure 4-2. Relative Record File Transfers

The key for random access within a key indexed file is an n-byte name ($n \leq 30$ USASCII bytes or binary bytes). The number of bytes in the key is fixed for all keyed records in the file when the file is defined. However, different key indexed files may have different key sizes. Keyed records are added by setting a bit in the PRB (random access method). Records without a key are added sequentially to the file. Records that are placed sequentially in the file without a key can only be retrieved with the sequential access method. No two records may have the same key (name). Most of the sequential access capabilities of key indexed files are identical to those described in Section III for sequential I/O devices. Insertion of a nonkeyed record is also allowed.

4.4.3.1 RANDOM ACCESS FUNCTIONS. The operating system supports the following random access functions:

- Read using key - With option to delete record and key.
- Write insert using key - Presence of same key considered error.
- Write replace/add using key - Data record with same key to be replaced with new data record.
- Delete at record level - keyed and non-keyed records.
- Read next higher or lower key.

4.4.3.2 SEQUENTIAL ACCESS FUNCTIONS. When performing a sequential access read within a key indexed file, logical records are retrieved in increasing order of key with the key treated as an unsigned integer (the order is alphabetical if the key is USASCII bytes). Unnamed records are positioned in the file as sequential records following either a keyed record or the beginning of file. Any access to a record in the file (read or write, random or sequential method) establishes a logical position that follows the accessed record. This logical position has no effect on a subsequent random access operation, but defines the location for a subsequent sequential access operation.

4.4.3.3 LIBRARY MANAGEMENT. Libraries consisting of groups of logical records identified with a common name are easily maintained through a combination of keyed and nonkeyed records in the same file. For example, a source program library could be written with a keyed record for the first source record. The remaining records in the program would be nonkeyed, sequential records. To retrieve a program, the system first performs a read using key to locate the start of the program. Subsequent sequential read operations retrieve the remaining records. The Key Recovery Desired bit in the PRB (word 1, bit 2) must be set during this operation so that the first time a key is returned, the system recognizes the corresponding record as the first record of the next program. This indication can be treated as an end-of-file if only one program is desired. If the program is the last in the file, the EOF flag (PRB word 0, bit 2) denotes end-of-file.

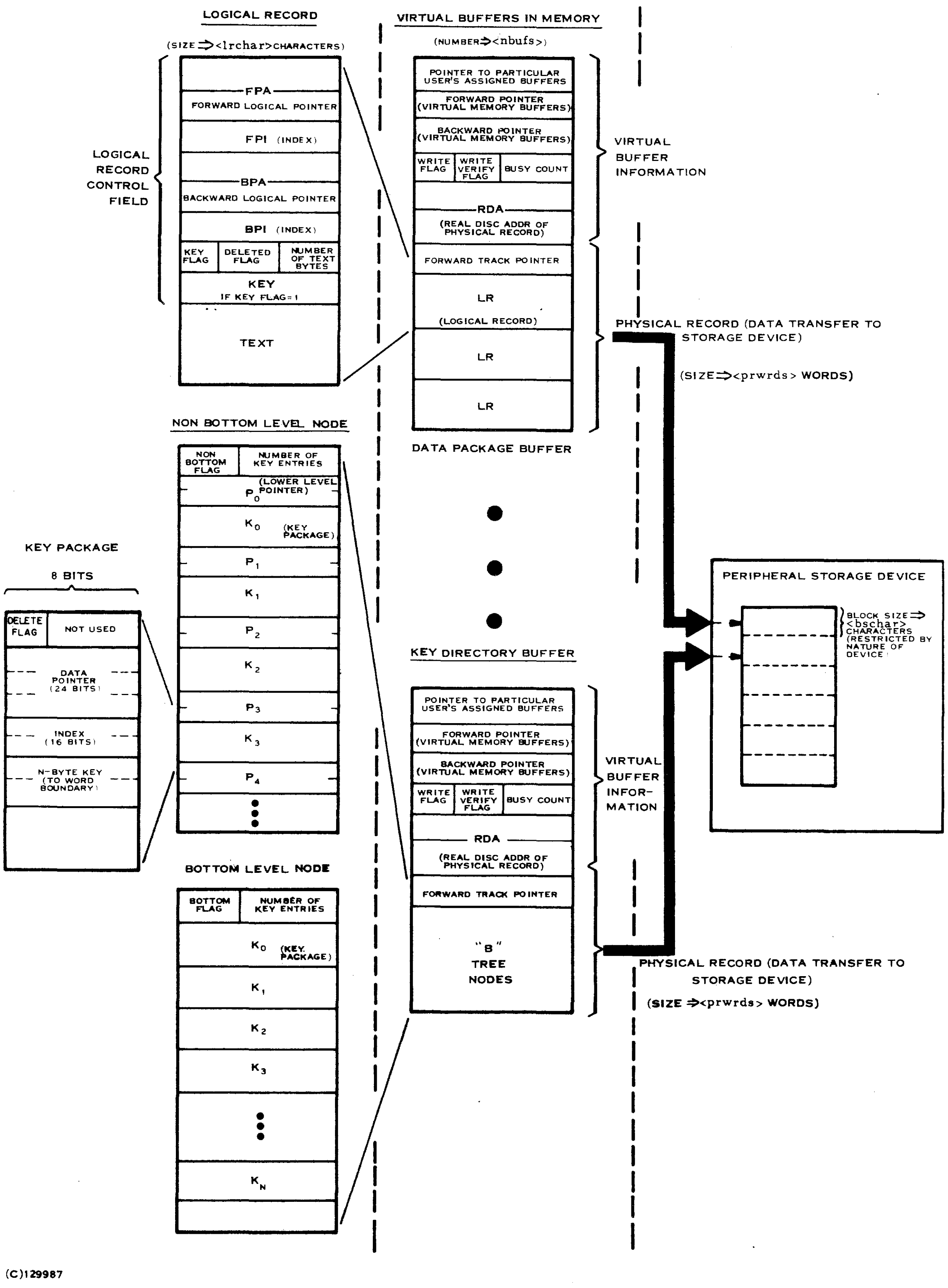
4.4.3.4 MULTIPLE RECORDS WITH THE SAME KEY. If the operating system detects an existing identical key within the file while performing a write using key operation, it returns a logical error indication to PRB word 0, bit 6. The user can still make an entry using that key, however. By performing a read using key operation to locate the key within the file, and then performing a sequential write without key at that location, the new record can be inserted at the beginning of a sequential string following the record with the target key. This operation generates a string of records for each key that operates so that the last record inserted will become the first record obtained sequentially following the keyed record itself.

4.4.3.5 BUFFER MANAGEMENT. A system wide buffering scheme provides buffer management for key indexed files. A major factor in gaining access to a key-controlled data record is the number of disc accesses required to search through the key structure. To minimize the number of accesses and reduce search time, DX980 allows multiple memory buffers for storing keyed records. The minimum number of buffers per job for key indexed files is two: one buffer for keys and another for data. The operating system provides two mechanisms for acquiring multiple buffers. The "BUFFERS=" parameter in the assignment command specifies the number of buffers to be allocated in the accessing program's job extension area.

In addition if multiple programs are sharing a key indexed file, the buffers from all sharing programs are grouped together for searching purposes. The user should specify a sufficient number of buffers in his job extension area to achieve the access time required for the application. If the file is being shared, the access time will be less than anticipated.

4.4.3.6 BUFFER SIZE. Buffer size selection for key indexed files is based on the length of each key, the number of keyed records in the file, and the desired data access time. Access time for data records is approximately equal to the number of disc accesses required to find and retrieve the record, times the average seek time for the disc drive. The number of disc accesses to get a data record is approximately equal to $\log_m n+1$, where n is the number of data records in the file and m is the number of keys in each physical record. The number of keys per physical record is equal to the physical record length minus four divided by the effective length of each key. The effective key length, including pointers and information about the data record, is five words plus the key length in words. Since key length is specified in characters in the JCL, the key length in words is the JCL specification divided by two and rounded up. The minimum buffer size specification must provide for two keys and at least one logical record: 14 words + 2* (key length/2). In addition the buffer size must be a multiple of 32 words.

Figure 4-3 illustrates the file structure for key indexed files. The terms used in the figure are defined in the following paragraphs. The structure is handled by DX980 rather than the user. The information is supplied for increased comprehension only.



(C)129987

Figure 4-3. Key Index File Parameters

4.4.3.7 LOGICAL RECORD. The logical records include both data and control fields for operating system use. The control area of the record is comprised of the following fields:

- Forward Pointer Address (FPA) - This two word space in each logical record contains the disc address of the physical record containing the next logical record in a linked file.
- Forward Pointer Index (FPI) - This one word area in the logical record designates the relative position of the next logical record within the physical record specified by FPA.
- Backward Pointer Address (BPA) - This two word space in each logical record contains the disc address of the physical record containing the previous logical record in a linked file.
- Backward Pointer Index (BPI) - This one word area in the logical record designates the position of the next logical record within the physical record specified by BPA.
- Key Flag - The operating system sets this bit to indicate that the logical record has a key associated with it.
- Delete Flag - The operating system sets this bit to indicate that the logical record has been deleted from the file.
- Text Count - This field indicates the number of bytes contained in the data (text) field of the logical record.
- Key - If the Key Flag is set, this field contains the key for the logical record.

4.4.3.8 DATA PACKAGE BUFFER. The data package buffer contains logical records and a track pointer that comprise the physical record stored on the disc, plus control information used by the operating system. The number of buffers allocated in a particular job is specified by the user through the <nbufs> parameter in JCL.

Track Pointer. The track pointer becomes part of the physical record on the disc. If the record is the last record on a track, it points to the address of the track containing the next physical record of the file. For the first physical record on a track, this field is used for a pointer to the previous track of the file. For interim records on a track, this field is not used.

Data Package Control. The control area of the buffer is comprised of the following fields:

- User Buffer Pointer - This field contains the main memory address of the first buffer assigned to the current user of the buffer.

- Forward Pointer (FP) - This one word field contains the main memory address of the next buffer in a series of buffers assigned to a job.
- Backward Pointer (BP) - This one word field contains the main memory address of the previous buffer in a series of buffers assigned to a job.
- Write Flag - When a user alters the information contained in a file buffer, the Write Flag sets. When set, this flag indicates to the operating system that the physical record must be written back to the disc before it is discarded after being used. This flag ensures that the changed data will be recorded in place of the old data on the disc.
- Write Verify Flag - When set, this flag indicates that the operating system must read data back from the disc following a write to ensure that the record was stored accurately.
- Busy Count - This field contains the number of users that are currently using the buffer. When the count equals zero, the system can replace the buffer contents with new information.
- Real Disc Address (RDA) - This two word field contains the disc address of the physical record currently in the buffer.

4.4.3.9 KEY DIRECTORY BUFFER. The key directory buffer is identical to the data package buffer, except that the logical record fields of the data package buffer are replaced with nodes of the sorting tree used to locate a particular key. Each node contains one control word that specifies if the node is a bottom node or an intermediate node, and that also indicates the number of keys contained in the node. The remainder of the node contains key packages, and if it is an intermediate node, pointers to the next lower level node in the sorting tree.

Lower Level Pointers. If the desired key is not contained in a current node, the operating system must access another block of keys (node) to locate the key. The system searches the current node until it finds the first key that is logically greater than (alphabetically past) the desired key. Associated with each key is a two word pointer. The pointer is the disc address of the node that contains keys in the level of the sorting tree below the associated key. The operating system uses that pointer to access the next node for the search. All of the keys in next lower level node are sorted and are alphabetically between the surrounding keys at the higher level.

Key Package. The key package consists of the following fields:

- Delete Flag - The operating system sets this bit to indicate that the key and its associated logical record have been deleted from the file.

- Data Pointer - This 24-bit field contains the disc address of the physical record that contains the logical record associated with the key.
- Index - This one-word field designates the position of the correct logical record within the physical record indicated by the data pointer.
- Key - This field contains a name that is from 1 to 30 bytes long and that identifies the logical record within the file. This key is repeated in the control field of the logical record associated with the key.

4.4.4 FILE ERRORS

As with I/O devices, three levels of errors can result from a SVC to perform file I/O. The conventions described in Section III for I/O errors apply to file errors also. Table 4-2 lists the possible file errors together with their associated severity and error number.

4.5 PHYSICAL RECORD BLOCK

The Physical Record Block (PRB) for file I/O is similar to that for device I/O as outlined in Section III except for the key address field (word 4). The PRB is four words in length unless bit 1 of word 2 is set. That bit indicates the presence of the key address in a fifth word. Key address is used by key indexed or relative record files. The field points to the memory address in the user program where the key can be located. For relative record files the indicated key contains a 15-bit binary value within the range of 0 to 32,767. Bit 0 of this word must be zero and bits 1 through 15 contain the record number. This number indicates the logical record number within the particular relative record file. For key indexed files the key contains a block of from 1 to 15 words (1 to 30 bytes) that functions as an alphanumeric byte string or a binary number. The length of this block (key) is specified with JCL. The key constitutes the name of the referenced logical record. Keyed reads and writes require both a key and a logical data record pointer. Table 3-4 outlines the opcodes that are applicable to I/O for relative record and key indexed files. Linked Sequential Files are accessed with I/O calls as described in Section III.

Table 4-2. File Errors

Error Number	Severity	Description
233	Severe	No space available on disc volume
234	Severe	File full. File status: can only be accessed for reading. No additional records can be written into the file, not even following a rewind operation. To reuse the disc space the file must be replaced.
235	Logical	Attempted write, logical record greater than physical record
236	Severe	Hardware failure on disc volume
237	Logical	Key indexed file - replace attempted on non-existent key
238	Logical	Key indexed file - write attempted on existing key
239	Logical	Key indexed file - write/replace (op code 101) attempted without specifying key.
240	Logical	Key indexed file - replace (op code 102) attempted on a keyed record without specifying key.
241	Logical	Key indexed file - replace (op code 102) attempted when file was positioned at EOF.
243	Logical	Key indexed and relative record files - no key match in the file
250	Severe	Insufficient tracks available for allocation
251	Severe	Insufficient contiguous tracks left for allocation
252	Severe	Allocation exceeds disc volume capacity
254	Severe	Unable to allocate buffers, job extension area too small
256	Severe	Insufficient number of buffers for attempted operation
257	Fatal	Opcode is either non-existent or illegal
258	Severe	Access violation for integrity code

Table 4-3. Relative Record and Key Indexed File Management Opcodes

I/O Code	Operation	File Type	Function
100	Write	R.R.	Fatal error - Must use Replace (102) or Write/Replace (101)
		K.I.	Write a logical record from the user's buffer to disc. 1) Key specified: a) If the specified key already exists in the file, do not write the logical record to disc. Return logical error status to the user, and do not reposition file pointer. b) If the specified key does not exist, make a key-directory entry for the key and write the logical record to disc under the new key entry. Position file pointer beyond completed write. 2) Key not specified: Insert the nonkeyed record where the file pointer is positioned; advance pointer.
101	Write/ Replace	R.R.	Perform a replace operation. 1) Key Specified: Replace the logical record of the specified key with the logical record from the user's buffer. Position file pointer for next sequential record. 2) Key not specified: Replace the record where the file pointer is positioned and advance the file pointer.
		K.I.	Unconditionally write the logical record from the user's buffer to the disc. 1) Key specified: a) If the specified key exists in the key-directory, replace the logical record associated with the key with the logical record from the user's buffer. Position file pointer to next sequential record.

Table 4-3. Relative Record and Key Indexed File Management Opcodes (Continued)

I/O Code	Operation	File Type	Function
101 (Cont)			b) If the specified key does not exist, make a key-directory entry for it, and write the logical record to disc under the new key entry. Position file pointer to next sequential record.
			2) Key not specified: Return an error status (number 239) to the user. Do not reposition file pointer.
102	Replace	R.R.	Replace the logical record on the disc with the logical record from the user's buffer.
			1) Key specified: Replace the record specified by the key and position file pointer to next sequential records.
			2) Key not specified: Replace record where the file pointer is positioned and advance the pointer.
		K.I.	Replace the logical record on the disc with the logical record from the user's buffer.
			1) Key specified: Replace the record specified by the key and position file pointer to next sequential record. If key does not exist, return error and do not reposition file pointer.
			2) Key not specified: Replace record indicated by file pointer and advance pointer. If record to be replaced has a key, then return a logical error (number 240 or 241). Do not reposition file pointer.
103	Read	R.R.	Read the logical record from the disc into the buffer specified by the user.
			1) Key Specified: Transfer logical record associated with specified key and position pointer to next sequential record.

Table 4-3. Relative Record and Key Indexed File Management Opcodes (Continued)

I/O Code	Operation	File Type	Function
103 (Cont)			2) Key not specified: Read record indicated by the file pointer and advance pointer.
		K.I.	Read the logical record from the disc into the user's buffer. 1) Key Specified: a) Read the logical record as specified by the key from the disc into the user's buffer. Position file pointer to next sequential record. b) If the specified key is not in the key-directory, return an error status to the user. Do not change file pointer. 2) Key not specified: a) Read the record indicated by the file pointer and advance pointer. b) If a keyed record is encountered and the PRB specifies that key recovery is desired, return the key to the user in the area allocated for the key. If key recovery is not desired, do not return the key. 3) No data desired: By setting the "return record size only" bit in the PRB a user can issue a Read and no logical record data is transferred. The size of the logical record is returned in the Data Record Length field of the PRB. Functions 1 and 2 above apply.

Table 4-3. Relative Record and Key Indexed File Management Opcodes (Continued)

I/O Code	Operation	File Type	Function
104	Read High	R.R.	Fatal error
		K.I.	Read the logical record from the disc into the user's buffer. 1) Key Specified: a) If the exact specified key exists in the key-directory, read the logical record associated with it. Position pointer to next sequential record. b) If the specified key does not exist, find the next algebraically higher key in the key-directory and read the logical record associated with it. Return the key that was actually found to the key field addressed by word 4 of the PRB. Position the pointer to next sequential record beyond retrieved record. c) If the specified key does not exist and no algebraically higher key exists in the key-directory, return an error status (end of medium) to the user. Do not reposition pointer. 2) Key not specified: Use the forward sequential-access method to find the next key in the key-directory beyond the record indicated by the file pointer. a) Return the key to the user if a keyed record is found and if the key recovery bit is set in the PRB. Put the associated logical record in the user's buffer. Position file pointer to next sequential record. b) If no keyed record is found, return an error status (end of medium) to the user. Position pointer past last data record in file.

Table 4-3. Realtive Record and Key Indexed File Management Opcodes (Continued)

I/O Code	Operation	File Type	Function
105	Read Low	R.R.	Fatal error
		K.I.	Read the logical record from the disc into the user's buffer. 1) Key Specified: a) If the specified key exists in the key-directory, read the logical record associated with it. Position pointer to next sequential record. b) If the specified key does not exist, find the next algebraically lower key in the key-directory and read the logical record associated with it. Return the key that was actually found to the key field addressed by word 4 of the PRB. Position pointer to next sequential record. c) If the specific key does not exist, and no algebraically lower key entries are in the key-directory, return an error (beginning of medium) status to the user. Do not change file pointer. 2) Key not specified: Use the sequential access method to find the next key in the key-directory going backwards from the current file pointer position. a) If a keyed record is found, return the key to the user (if the key recovery bit is set in the PRB) and put the associated logical record in the user's buffer (if data is to be transferred). Position file pointer to record following the record used.

Table 4-3. Relative Record and Key Indexed File Management Opcodes (Continued)

I/O Code	Operation	File Type	Function
105 (Cont)			b) If no keyed record is found, return an error (beginning of medium) status to the user. Position file pointer to beginning of file.
106	Read Delete	R.R.	Fatal error
		K.I.	Read the logical record associated with the specified key into the user's buffer. Then mark both the key entry in the key-directory and its associated logical record as deleted. Same rules apply as for Read (I/O code 103) plus Delete (I/O code 107) for K.I. files.
107	Delete	R.R.	Fatal error
		K.I.	Mark the indicated record as deleted. 1) Key specified: a) If the key exists, mark the specified key entry in the key-directory and its associated logical record as deleted. Any sequentially linked record(s) that exist for the specified key remain, but the associated key and its record are specifically marked as deleted. Position pointer to next record. b) If the specified key entry does not exist in the key-directory, return an error status to the user. Do not change file pointer. 2) Key not specified: a) Mark the logical record at the current file pointer position as deleted. Increment the pointer to the next logical record. If the deleted record has a key, delete that key from the key-directory.

Table 4-3. Relative Record and Key Indexed File Management Opcodes (Continued)

I/O Code	Operation	File Type	Function
107 (Cont)			b) If the key recovery bit is set in the PRB and the next logical record contains a key, then return the deleted key to the user in the key field of the PRB. c) When the last logical record is deleted, the file pointer is incremented to point at the end of medium. Therefore, the appropriate status is returned to the user.
108	Delete Sequentially	R.R.	Fatal error
		K.I.	Mark the specified key entry, its associated logical record(s), and any sequentially linked logical record(s) up to, but not including, the next algebraically successive key entry as deleted. 1) Key Specified: a) If the key entry exists in the key-directory, mark it and any sequentially linked logical record(s) as deleted. If the key recovery bit is set, the next successive key is returned to the key field of the PRB. Position file pointer beyond deleted records. b) If the key entry does not exist in the key-directory, return an error status to the user. Do not change file pointer. 2) Key not specified: a) Begin at the current file pointer position and mark the sequentially linked logical record(s) up to, but not including, the next successive key entry as deleted. Position the file pointer beyond deleted records.

Table 4-3. Relative Record and Key Indexed File Management Opcodes (Continued)

I/O Code	Operation	File Type	Function
108 (Cont)			b) If the key recovery bit is set, return the next successive key to the user in the key field addressed by PRB Word 4. c) If an end of medium is detected, return the appropriate status to the user.
109	Delete All	R.R.	Fatal error
		K.I.	Mark all key-directory entries (key and non-key) and all logical data records in the file as deleted (delete all data contents of file to create an empty file). File allocation remains the same.

SECTION V
SUPERVISOR CALLS

5.1 GENERAL

DX980 supervisor calls (SVC) are requests to the operating system to perform a service for the user. This request is the only communication between a user program and the operating system. The computer interprets an SVC as an illegal machine instruction. When the computer encounters an illegal instruction, it automatically branches (traps) to the internal interrupt entry address. The operating system examines all instructions that cause a trap to determine if they are SVC opcodes. If not, the instruction actually is illegal and the system aborts the offending program. If the instruction is an SVC, the operating system decodes the remainder of the instruction and calls the appropriate system service routine to process the SVC.

For compatibility with the Basic Monitor System, DX980 recognizes two SVC instruction formats: C380 and F800. The C380 SVCs conform to SVCs for the Basic Monitor. However, the Basic Monitor call Set Control Status Flag (C384) is not recognized by DX980. The Basic System Use and Operation manual referenced in the preface to this manual explains Basic Monitor SVCs.

The standard DX980 SVC format is based on instructions in the form: F8XX (XX corresponds to the hexadecimal equivalent of the SVC number as described in this manual). To issue an F8XX SVC, construct an argument list that contains a word identifying the number of arguments described by the list, followed by successive words containing the addresses of each argument. Then set the M register to the address of the list and allow the program to execute the SVC. For example, a Wait for I/O SVC is implemented as follows:

```
        @LDM = ARGLST
        DATA  >F82B        (2B16 = 4310, Specifies SVC 43)
          .
          .
          .
ARGLST  DATA  1,PRB        Designates 1 Argument at Address PRB.
```

Supervisor calls can be specified in decimal form if the OPD (Operation Definition) assembler directive is first used to define a new instruction mnemonic, SVC, that maps the decimal number to its hexadecimal equivalent. Then the statements:

```
SVC 43
  and
DATA >F82B
```

are equivalent.

Table 5-1 summarizes the DX980 SVC. Table 5-2 lists and describes each SVC. The remaining paragraphs in this section provide examples of parameter setup and calling sequences for each SVC, plus a more detailed description.

Table 5-1. DX980 User Supervisor Calls

Base Ten SVC Number	Function
0	Input/Output
1	Terminate Job
2	Set Floating Point Address
3	Get Memory Limits
4	Terminate Job Abnormally
5	Terminate Task
6	Delete Task
7	Suspend Task
8	Post Event
29	Get Time and Date
30	Create Task
37	Load
38	Load and Relocate
41	Command Scanner
43	Wait for Input/Output
49	Allocate Resource
51	Deallocate Resource
98	Get Program Limits
129	Start Job

Table 5-2. DX980 Supervisor Call Description

Base Ten SVC Number	Function	Number of Arguments, Name(s)	Description
0	I/O	1, PRB	Performs all I/O for the user as specified by the PRB.
1	Terminate Job	0	Normal Job Termination
2	Set Floating Point Address	1, FLT980	Set floating point address to trap to for any floating point instructions.
3	Get Memory Limits	1, ARRAY	Get memory limits of user partition from job partition and return values in the supplied array.
4	Terminate Job Abnormally	1/2, ERRCOD, ERRID	Abnormal Job Termination. Number of arguments is either 1 or 2. First argument is ERRCOD, an error code supplied by the user (should be 1000 or greater). Second optional argument is ERRID, the address of a 6 character identifier that the user supplies to identify which error the ERRCOD applies to.
5	Terminate Task	0	Normal Termination of the Running Task.
6	Delete Task	1, TASKID	Delete all tasks under the user job having the same name as TASKID.
7	Suspend Task	1/2, WCL, RETEDB	Suspend the running task under the user job waiting for event(s) as specified by the Wait Criteria List (WCL). RETEDB is optional; if specified, the task returns to the last matched Event Description Block (EDB) on which the match occurred.
8	Post Event	1, EDB	Post the event specified in the EDB.

Table 5-2. DX980 Supervisor Call Description (Continued)

Base Ten SVC Number	Function	Number of Arguments, Name(s)	Description

29	Get Time & Date	4, BTIM, BDAT, CTIM, CDAT	Return the System Time and date in the following parameters:
			BTIM = System Time in Binary (2 words, milliseconds since midnight)
			BDAT = System Date in Binary (2 words, year & day)
			CTIM = System Time in Characters (3 words, HH:MM:SS)
			CDAT = System Date in Character (3 words, MM:DD:YY)
			The format of the date is year followed by day of the year (instead of the month).

30	Create Task	Variable	Create a task under the user job as specified by the following parameters: 1) TPRI - Relative Task Priority 2) TID - Task identifier (user specified binary number). 3) TSTART - Starting location of first instruction task is to execute. 4) TREG - Pointer to a register file. If the pointer is -1, the register file for the new task is to be the same as the creating task. If not -1, the parameter is a pointer to the register file containing values to be passed to the new task.

Table 5-2. DX980 Supervisor Call Description (Continued)

Base Ten SVC Number	Function	Number of Arguments, Name(s)	Description
30 (Cont)			5) TSS - TCB STACK SIZE; number of words to be allocated in the auxiliary stack attached to the TCB. 6) TWCL - Wait Criteria List - A linear array of one or more words. If the first word of this parameter is zero, the task is to be created in the Ready state. If the first word is non-zero, the parameter is the Wait Criteria List and the task is created in the Dormant state. The following input parameters are to be included only if the stack area for task creation is required. If no user supplied stack area setup is required, the parameter list ends here. 7) TSTK - Pointer to user supplied stack. This stack, within the user partition, is required if the reentrant task to be created requires work space. 8) TWRK - Work Area flag. If zero, no work area is required. If non-zero, a pointer to the work area within the user supplied stack area is to be supplied.

Table 5-2. DX980 Supervisor Call Description (Continued)

Base Ten SVC Number	Function	Number of Arguments, Name(s)	Description
30 (Cont)			9) TFLAG - Argument flag word - i.e., if any arguments are supplied, each bit of this word corresponds to an argument and specifies whether (= 1) or not (= 0) to move the first word of the argument list to the TCB. 10) ARG1 - Argument 1 address ARG2 - Argument 2 address ARGN - Argument N address
37	Load	3, MIP#, LOADR, EPA	Load MIP# (Memory Image Phase Number), at LOADR (a specified load address) and return the address to give control at EPA (Entry Point Address).
38	Load and Relocate	3, MIP#, LOADR, EPA	Same as SVC# 37 (LOAD) but perform the necessary relocation.
41	Command Scanner	5, CMDSTR, KEY, CTRL, PAKSTR, RESLAB	A free format input record - Descriptor array for output string - Controls SVC41 processing - Packed output string - Reserved labels for command operator
43	Wait (Suspend) for I/O	1, PRB	Suspend execution of this user program until the I/O (specified by the PRB) completes.

Table 5-2. DX980 Supervisor Call Description (Continued)

Base Ten SVC Number	Function	Number of Arguments, Name(s)	Description
49	Allocate Resource	2, JERR, JLDT	Allocate Resource at runtime. JERR - Error/Availability Code returned to caller. 0 means allocation made -5 means device offline -4 means device already assigned (unavailable) -3 means device already committed (unavailable) >0 an error encountered during assignment. JLDT - A user supplied "assign-time" LDT from which the "run-time" LDT is built.
51	De-allocate Resource	2, JERR, JLUNO	De-allocate resource at run-time. JERR - error code returned to caller. JLUNO - LUN number under users job of the resource to be de-allocated.
98	Get Program limits	1, ARRAY	Get program limits of Job partition and return values in the user supplied array.
129	Start Job	1, JSB	Starts an "independent" job from the user job. JSB (JOB START BLOCK) - a user supplied block, for starting a job. Includes any required JLDT's.

5.2 INPUT/OUTPUT - SVC NUMBER 0

```
        @LDM =ARGLST      Set M-Register to List Address
        SVC   0           Execute Call
          .
          .
          .
ARGLST  DATA  1           1 Argument
        DATA  PRB         Physical Record Block
```

This I/O call requests the operating system to perform input/output or file management action. Sections III and IV of this manual explain the system services available in response to this call.

5.3 TERMINATE JOB - SVC NUMBER 1

```
        @LDM =ARGLST      Set M-Register to List Address
        SVC   1           Execute Call
          .
          .
          .
ARGLST  DATA  0           No Arguments
```

This SVC terminates the job issuing the SVC. The Job Management system performs job termination, closes any files left open by the user, releases the files, devices and memory used by the job step, and prints a job step termination message on the system console. If the calling job step is the last or only step within a job, a job string termination message is also printed.

SVC 1 is the standard terminating method for jobs that have reached a normal conclusion. Although programs normally close all files and devices before termination, the operating system can also perform this function. Any incomplete input/output operation may be prematurely terminated.

5.4 SET FLOATING POINT ADDRESS - SVC NUMBER 2

```
        @LDM =ARGLST      Set M-Register to List Address
        SVC   2           Execute Call
          .
          .
          .
ARGLST  DATA  1           1 Argument
        DATA  FLT980      Package Entry Address
```

This call supplies the operating system with the address of the floating point package within the user's program. The Set Floating Point Address call (SVC 2) must be issued before performing any floating point operation. The

Floating Point Package in the FORTRAN Subroutine Library must have been previously combined with the user program by the link editor. The argument is the entry point address of the package.

5.5 GET MEMORY LIMITS - SVC NUMBER 3

```
        @LDM  =ARGLST   Set M-Register to List Address
        SVC   3         Execute Call
          .
          .
          .
ARGLST  DATA  1         1 Argument
        DATA  LIMITS    Limits Depository Address
          .
          .
          .
LIMITS  BSS   2         Lower and Upper
```

SVC 3 returns the memory limits of a user job area in the LIMITS argument of the SVC. Memory limits correspond to the lower and upper limit registers that surround a user's addressable memory area. LIMITS is a two element vector that contains the lower limit in LIMITS (0) and the upper limit in LIMITS (1) when control returns to the user program. During program execution, this SVC can determine the job area size (<jarea>) that was specified when the job was submitted. Since job area size varies with each job submission, this information tells a program the amount of memory supplied for a given submission. The lower limit is invariably returned as the value zero for protected programs. When using SVC 3, the following formula yields available memory for workspace:

$$\text{Workspace} = \underbrace{\text{Supplied Memory Size}}_{\substack{\text{Upper Limit minus}\\ \text{Lower Limit}}} - \underbrace{\text{Program Residence Requirements}}_{\substack{\text{Last Program Address minus}\\ \text{First Program Address}}}$$

SVC 98 performs a similar function and is simpler to use in certain cases.

5.6 TERMINATE JOB ABNORMALLY - SVC NUMBER 4

```
        @LDM  =ARGLST   Set M-Register to List Address
        SVC   4         Execute Call
          .
          .
          .
ARGLST  DATA  1 or 2    2 if Optional Argument Used
        DATA  ERRCOD    Address of Binary Error Code
        DATA  ERRID     Optional, Address of 6-character ID
          .
          .
          .
ERRCOD  DATA  value     System Console Prints Value + 1000
ERRID   DATA  'ABCDEF'  System Console Prints ABCDEF
```

The operating system terminates a job abnormally because of a fatal program error. Similarly, the program can terminate itself abnormally because of an abortive error made by the user. Since the termination message and error code are displayed on the system console, a user can notify the console operator of an abortive condition without assigning the system console to his program. To avoid confusion between user generated and system generated termination codes, the system adds $10,000_{10}$ to ERRCOD before printing. ERRCOD is a 16-bit number.

5.7 TERMINATE TASK - SVC NUMBER 5

```
              @LDM =ARGLST      Set M-Register to List Address
              SVC    5          Execute Call
                 .
                 .
                 .
       ARGLST DATA   0          No Arguments
```

SVC 5 invokes normal task termination. If the subject task is the last or only task for a job, the job also terminates normally.

5.8 DELETE TASK(S) - SVC NUMBER 6

```
              @LDM =ARGLST      Set M-Register to List Address
              SVC    6          Execute Call
                 .
                 .
                 .
       ARGLST DATA   1          1 Argument
              DATA   TASKID     Address of Task Number
                 .
                 .
                 .
       TASKID DATA   value      16-bit Value for TASKID
```

This call deletes all tasks within the job whose identifying number corresponds to that given in TASKID. TASKID is a 16-bit binary number that must correspond to the TASKID supplied when the subject task(s) was created. SVC 6 can delete more than one task at a time if they have the same task identification (TASKID). If several tasks are created to perform a similar function, the user can thereby cancel them all at once. This SVC is used by one task to delete another. Thus, a supervisory routine can maintain control by creating and deleting tasks as dictated by the environment.

5.9 SUSPEND TASK (WAIT FOR EVENT) - SVC NUMBER 7

```
         @LDM = ARGLST  Set M-Register to List Address
         SVC    7       Execute Call
          .
          .
          .
ARGLST   DATA   n       Number of Arguments
         DATA   WCL     Address of Wait Criteria List (WCL)
+-------------------------------------------------------------+
|        DATA   RETEDB  Address of a structure containing the |
|                       returned event descriptor block (EDB) |
|                                                             |
|                                                    optional |
+-------------------------------------------------------------+
          .
          .
          .
WCL      DATA   value   A List of Events
         DATA   value   Descriptors
          .      .
          .      .
          .      .
RETEDB   DATA    $-$    This field will contain upon activation
          .      .      the EDB on which the last matched
          .      .      occurred. It must be the size of the
          .      .      largest EDB identified in the WCL.
         DATA    $-$
          .      .
          .      .
          .      .
```

SVC 7 suspends the calling task until the occurence of the selected combination of events as described in the Wait Criteria List (WCL). The Suspend Task (SVC 7) and Post Event (SVC 8) calls are extremely useful in multitask environments. Together these two SVCs coordinate the activities of several tasks that are running asynchronously. The calls cooperate within a single job or between separate jobs. The suspend task SVC can also be used independently to wait for specified system events, such as time of day. A task may be suspended to wait for a single event or for several events to occur. Upon reactivation a user may determine on which EDB the last match occurred.

The returned event descriptor block argument is optional and, if specified, this argument is a pointer to a structure large enough to contain the largest EDB specified in the WCL. Upon reactivation of the task, this structure will contain the EDB on which the last match occurred.

5.9.1 WAIT CRITERIA LIST (WCL)

The Wait Criteria List is the only required argument for the Suspend Task SVC. A WCL contains one or more Event Descriptor Blocks (EDB) that define the parameters that must be satisfied before the task can resume execution. Figure 5-1 illustrates the components of an EDB. The Event Index (Word 1) is a numerical value that identifies the event to the operating system. These index values are given in the descriptions of system-wide and job-oriented event types later in this section. The index may require 0, 1 or 2 I.D. words within the EDB. The number and meaning of the I.D. words is different for each index.

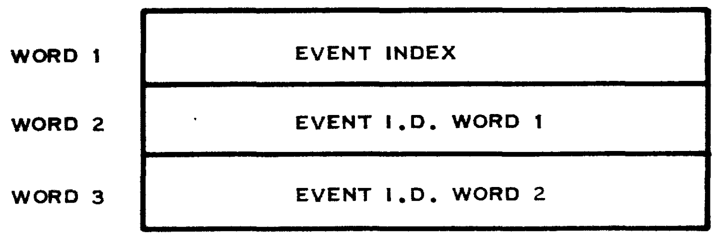
A WCL containing only one EDB requires no further information. If the WCL contains more than one EDB, the EDBs must be preceded in the WCL by two words that prescribe:

1. The two's complement of the number of EDBs in the WCL
2. How many of the events must be satisfied before the task is activated.

Figure 5-2 illustrates a WCL containing multiple EDBs.

5.9.2 SYSTEM-WIDE EVENTS

A system-wide event is an event that is beyond the scope of a single job. When tasks are suspended to wait for a system-wide event, the operating system places the associated EDB's in the job extension area, and links them together in separate lists for each event index. Each event index list may contain EDBs from several concurrent jobs. All system-wide events can be specified in an SVC 7 call as an event to reactivate a task. Some system-wide events can be specified in an SVC 8 call, while others can be posted only by the operating system. Those system-wide events that allow an SVC 8 call (user posting) provide the capability for synchronizing tasks



(A)129480

Figure 5-1. Event Descriptor Block Organization

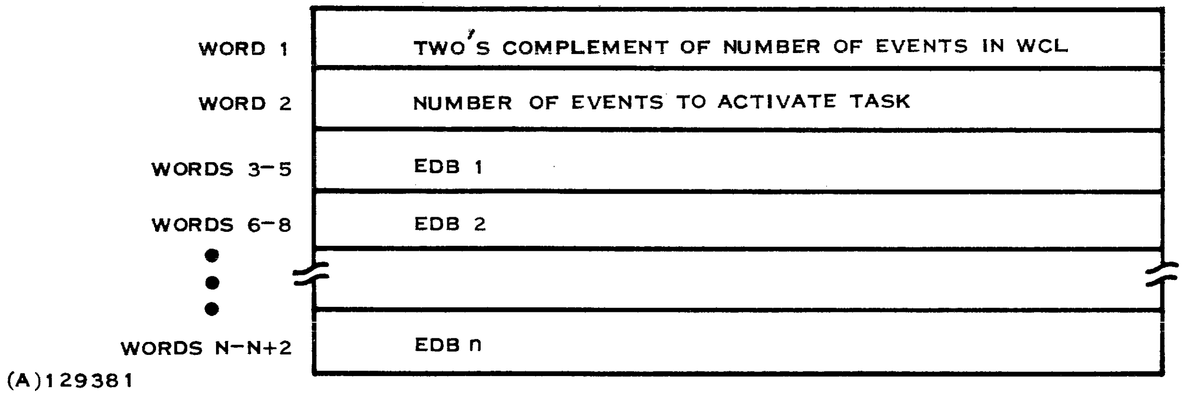

WORD 1
TWO'S COMPLEMENT OF NUMBER OF EVENTS IN WCL
WORD 2
NUMBER OF EVENTS TO ACTIVATE TASK
WORDS 3-5
EDB 1
WORDS 6-8
EDB 2
WORDS N-N+2
EDB n
(A)129381

Figure 5-2. WCL With Multiple EDB

across job boundaries. Table 5-3 lists the system-wide events and their attributes. The following paragraphs define the column headings of that table.

5.9.2.1 USER WAITABLE. This column indicates whether the event can be the object of an SVC 7 call (user wait) from a user program.

5.9.2.2 USER POSTABLE. This column indicates whether the event can be posted by the user with an SVC 8 call. User postable system-wide events are not remembered for matching with subsequent SVC 7 calls. Therefore the SVC 7 suspend must precede the SVC 8 post.

5.9.2.3 SINGLE OR MULTIPLE MATCH. When an event is posted, the operating system scans all the EDBs linked with the posted event index to determine if the posted event releases any of the waiting tasks. Similarly (but only for savable job-oriented events), when a task is suspended, the operating system examines previous postings to determine if the event specified in the Suspend Task call has already happened. User postable system-wide events are not remembered beyond a single scan. Therefore, a Suspend Task call awaiting a previously user-posted system-wide event must wait for the next posting to be released. The single or multiple match column specifies how far the operating system will search following the posting of an event. If the table indicates a single match, then the operating system scans waiting EDBs until it finds only one task to release. If the table indicates a multiple match, the operating system scans all waiting EDBs in the list and releases all tasks that are waiting for that event. When multiple tasks are released, processing will proceed in priority order. Tasks are released if the event I.D. words of both the SVC 7 and SVC 8 match as specified by the Relational Operator.

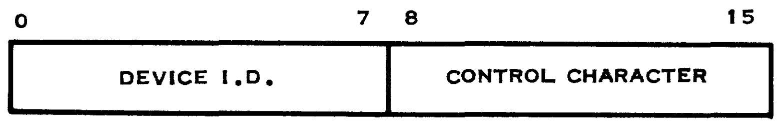
Table 5-3. User Accessible System-Wide Events

Index Number	User Waitable	User Postable	Single or Multiple Match	Relational Operator	Number I.D. Words	Event Description
22	Yes	No	Multiple	≥	2	Time-of-day (double prec. word in milliseconds)
23	Yes	No	Multiple	≥	2	Delta time (double prec. word in milliseconds; 100 ms min)
24	Yes	No	Multiple	=	0	Any job step termination
25	Yes	No	Multiple	=	1	Particular job string termination (bits 0-11 = job string no.)
29	Yes	No	Multiple	= ①	1	②
35	Yes	Yes	Single	=	1	Open to user
36	Yes	Yes	Single	=	1	Open to user
37	Yes	Yes	Multiple	=	1	Open to user
38	Yes	Yes	Multiple	=	1	Open to user
39	Yes	Yes	Multiple	=	1	Open to user

NOTES:

① If Bits 0-7 of the event I.D. are not all one's, then the compare relation operator "=" is done on all of the event I.D.; otherwise, it is only done on Bits 8-15 of the event I.D.

② The event I.D. is a 16-bit composite of an internal device I.D. and a control character, as follows:



The internal device I.D. is an 8-bit field that identifies the data terminal where the control character is generated. This field is set to a >FF if the match is to be made on any data terminal generating the control character. The control character is a 7-bit pattern generated on the data terminal. Refer to the table of USASCII Control Characters in Section III to determine which control characters are postable.

5.9.2.4 RELATIONAL OPERATOR. This column lists the criteria for determining if the I.D. words of a posted event match the I.D. words of a waiting task. The task is released if the I.D. words are equal (=), or if the I.D. words of the posted event are greater than or equal to (≥) the waiting task's I.D. words.

5.9.2.5 NUMBER I.D. WORDS. This column lists the number of I.D words that must be included in the EDB of an SVC 7 call that specifies the assocaited event index number.

5.9.3 JOB ORIENTED EVENTS

Job oriented events synchronize tasks within a single job. All job oriented events are User Postable and have a Relational Operator of "=". A job oriented event may be posted by a task even though no other task is currently waiting for that event. If the event is specified as savable, the post is preserved until a corresponding wait is issued, or until the job terminates. Once saved, the post is not retained past the first match. Table 5-4 lists the job oriented events and their attributes. Each job defines the functions of the events that it uses.

Table 5-4. Job-Oriented Events

Index Number	Savable	Single or Multiple Match	Number I.D. Words
40	Yes	Single	1
41	Yes	Single	1
42	Yes	Single	2
43	No	Single	1
44	No	Single	1
45	No	Single	2
46	No	Multiple	1
47	No	Multiple	1
48	No	Multiple	2
49	No	Multiple	1

5.10 POST AN EVENT - SVC NUMBER 8

```
        @LDM = ARGLST     Set M-Register to List Address
        SVC    8          Execute Call
         .
         .
         .
ARGLST  DATA   1          1 Argument Address
        DATA   EDB        Address of Event Descriptor Block
         .
         .
         .
EDB     DATA   Event index Event Descriptor Block
        DATA   I.D. Word 1
        DATA   I.D. Word 2
```

The Post Event SVC notifies the system that the specified event has occurred. The system then performs the necessary processing to either activate waiting tasks or queue the event posting. Queueing occurs only for savable job-oriented events that have no tasks waiting for them at the time of the posting call. System-wide events are posted according to the attributes defined for the corresponding event index. These postings may affect tasks throughout the system. Job oriented events are posted according to the corresponding job oriented event index. These postings can only affect tasks within the same job.

The Post Event SVC contains only one argument: an Event Descriptor Block (EDB). The format of the EDB for SVC 8 is the same as that for SVC 7.

5.11 GET TIME AND DATE - SVC NUMBER 29

```
        @LDM =ARGLST    Set M-Register to List Address
        SVC   29        Execute Call
         .
         .
         .
ARGLST  DATA  4         4 Arguments
        DATA  BTIM      Address for Binary Time (Milliseconds)
        DATA  BDAT      Address for Binary Date (Year, Day)
        DATA  CTIM      Address for Character Time (Hours,
                        Minutes, Seconds)
        DATA  CDAT      Address for Character Date (Month,
                        Day, Year)
         .
         .
         .
BTIM    BSS   2         Binary Milliseconds Since Midnight
BDAT    BSS   2         Binary Year, Binary Day in Year
CTIM    BSS   3         HHMMSS in USASCII
CDAT    BSS   3         MMDDYY in USASCII
```

This call gets the current time and date from the system. The operator supplies the system with time and date at IPL time and may change it from the operator's console while DX980 is running. An interval timer maintains the time for the system. There are 86,400,000 milliseconds in a day. The double length time word accomodates 1,073,741,823 milliseconds. Therefore, the timer does not overflow during a day. The interval timer for most installations interrupts each 100 milliseconds so that BTIM is truncated to the nearest 100 milliseconds.

5.12 CREATE TASK - SVC NUMBER 30

```
         @LDM  =ARGLST    Set M-Register to List Address
         SVC   30         Execute Call
           .
           .
           .
ARGLST   DATA  n          Number of Arguments
         DATA  TPRI       Address of Task Relative Priority
         DATA  TID        Address of Task Identifier
         DATA  TSTART     Address of Task Entry Point
         DATA  TREG       Address of Starting Register Values
         DATA  TSS        Address of Task System Stack Size
         DATA  TWCL       Address of Task Wait Criteria List
        ---------------------------------------------------------
        |DATA  TSTK       Address of User Stack Address         |
        |DATA  TWRKA      Address of Flag for Work Area         |
        |DATA  TFLAG      Address of Argument Flags             |
        |DATA  ARG1       User Argument Address 1               |
        |DATA  ARG2       User Argument Address 2               |
        |  .                                                    |
        |  .                                            optional|
        ---------------------------------------------------------
           .
           .
           .
TPRI     DATA             Priority Relative to Job
TID      DATA             Arbitrary 16-Bit Task I.D.
         REF   TSTART     Task Entry Point
TREG     DATA             -1=Undefined Register Values, Otherwise,
                             Points to Starting Values for A,E,X,M,
                             S,L and B.
TSS      DATA             System Stack Size, for New Task

TWCL     DATA             Zero, or Wait Criteria List if the Task
                             begins Suspended While Awaiting for
                             a Posted Event (See SVC 7).
TSTK     DATA  AREA       Address of Stack Area
AREA     BSS   size       Stack Area for Building Call Arguments
                             for Task
TWRKA    DATA  0          If Not Zero, a flag indicating User Work
                             Area in AREA

TFLAG    DATA  0          Bits that Equal 1 to Indicate Arguments
                             Transcribed into AREA
```

Once a job is running with the single task created by the system, additional tasks can be created from the user program with the SVC 30 call. The created tasks can have an equal or lower priority (high numerically) than the job under which the task runs. The priority for a new Task, TPRI, is stated relative to the basic job step priority specified in JCL (<jsprty>). TID supplies an identifier for the new task. The identifier may be referenced in a subsequent SVC 6. The starting address for program execution within the task is supplied as TSTART. TREG is a pointer to a set of values for the register file. If the pointer is to a -1, the register file values for the new task are undefined. If the location does not contain -1, the parameter points to values to be passed to the new task in A, E, X, M, S, L, and B registers. The new task requires allocation in the job extension area memory for system temporary storage (see Section II, < stksize>). TSS specifies this allocation. The argument TWCL, if zero, indicates that the new task is to be created active. If the first word of TWCL is not zero, then it and subsequent memory words constitute a wait criteria list as defined under SVC 7 and the task is created in a suspended state. The wait criteria list defines the event(s) that activate the new task.

5.12.1 OPTIONAL ARGUMENTS

The call arguments TSTK, TWRKA, TFLAG, and ARG_i are optional. If none of these are furnished, the new task receives a value defined by TREG in the M-Register. If these arguments are supplied, the M-Register points to AREA as specified by TSTK in the call. The operating system establishes the first word of AREA to specify the number of address arguments to be placed in AREA. Any address arguments that the operating system transcribes into AREA before activating the task appear in the following order:

1. The address of work space within AREA
2. ARG1
3. ARG2

   •
   •
   •

Each bit in the TFLAG argument corresponds to one of the supplied argument addresses; bit 0 represents the first address, bit 1 represents the second address, etc., to a maximum of 16 addresses. If the bit is a zero, the operating system transcribes the argument address (ARGi) into AREA. If the bit is a one, the operating system transcribes the argument value into AREA immediately following the argument addresses, and alters the corresponding argument address in AREA to point to the location in AREA containing the argument value. Figure 5-3 illustrates possible contents of AREA when the task is activated.

5.12.2 CREATE TASK EXAMPLES

The following paragraphs illustrate some uses of the Create Task SVC.

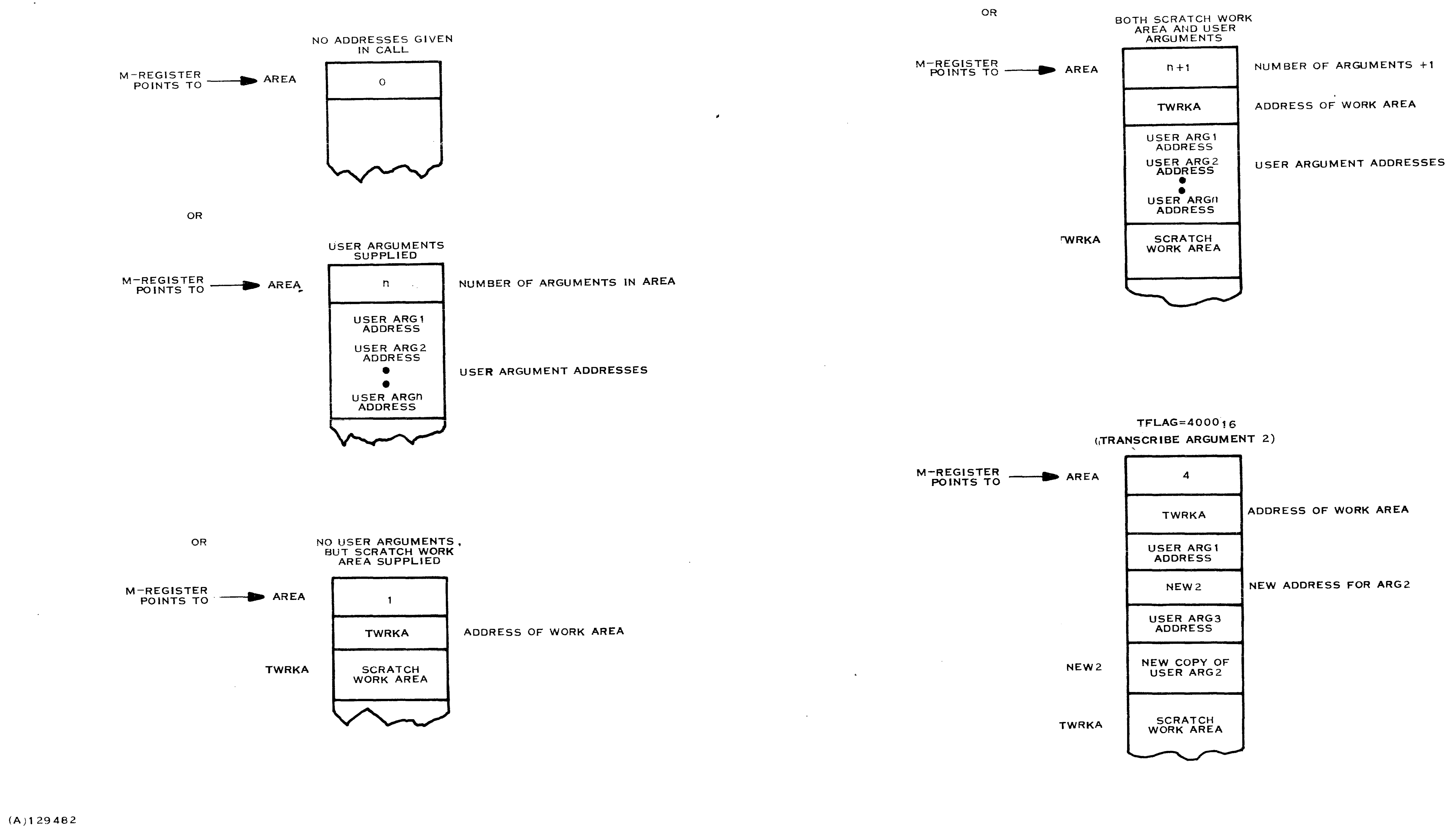


Figure 5-3. Sample TSTK Contents At Task Activation

5.12.2.1 NO ARGUMENTS. The following sample call creates a task that activates a non-reentrant subroutine, NEWTSK, with no arguments. NEWSTK performs non-file I/O, thus requiring a TCB stack of 110 words.

```
         @LDM   =LIST       Set List Address
         SVC    30          Create Task
         BRU    NEXT        Computation continues
          .
          .
          .
LIST     DATA   6           Six Arguments in List
         DATA   ZERO        New Task Priority Same as Job Priority
         DATA   TASKID      Numeric Task Identifier
         DATA   NEWTSK      Pointer to NEWTSK
         DATA   MINUS1      No Register Arguments
         DATA   TCBSTK      TCB Stack of 110 Words
         DATA   ZERO        Create Active Task
          .
          .
          .

                            Constants, Arguments, etc.
ZERO     DATA   0
TASKID   DATA   1
MINUS1   DATA   -1
TCBSTK   DATA   110
         REF    NEWTSK
```

5.12.2.2 DORMANT TASK. The following example creates a dormant task that activates in 10 seconds and then activates a non-reentrant subroutine, NEWTSK, with no arguments. NEWTSK performs non-file I/O, thus requiring a TCB stack of 110 words.

```
         @LDM   =LIST       Set List Address
         SVC    30          Create Task
         BRU    NEXT        Computation Continues
          .
          .
          .
LIST     DATA   6           Six Arguments in List
         DATA   ONE         New Task Priority One Lower Than
                               Job Priority
         DATA   TASKID      Numeric Task Identifier
         DATA   NEWTSK      Pointer to NEWTSK
         DATA   MINUS1      No Register Arguments
         DATA   TCBSTK      TCBSTK of 110 Words
         DATA   WCL         Pointer to Wait Criteria List
          .
          .
          .
```

(listing continued on next text page)

(listing continued from preceding text page)

```
                                 Constants, Arguments, etc.

ONE      DATA   1
TASKID   DATA   0
MINUS1   DATA   -1
TCBSTK   DATA   110
WCL      DATA   23
         DATA   0
         DATA   10000
         REF    NEWTSK
```

5.12.2.3 ARGUMENTS IN A AND X. The following example creates a task that activates a non-reentrant subroutine, NEWTSK, with one argument in the A register and one argument in the X register. NEWTSK performs non-file I/O, thus requiring a TCB stack of 110 words.

```
         @LDM   =LIST    Set List Address
         SVC    30       Create Task
         BRU    NEXT     Computation Continues
          .
          .
          .
LIST     DATA   6        Six Arguments in List
         DATA   ZERO     New Task Priority Same as Job Priority
         DATA   TASKID   Numeric Task Identifier
         DATA   NEWTSK   Pointer to Newtsk
         DATA   TREG     Pointer to Register File
         DATA   TCBSTK   TCB Stack of 110 Words
         DATA   ZERO     Create Active Task
          .
          .
          .
                         Constants, Arguments, etc.
TREG     DATA   RFILE
ZERO     DATA   0
TASKID   DATA   1
RFILE    DATA   I        A Register - Address of I
         BSS    1        E Register - Dummy
         DATA   10       X Register - Index of 10
         BSS    1        M Register - Not Specifiable
         BSS    3        S, L, B - Dummy
TCBSTK   DATA   110
         REF    NEWTSK
I        BSS    100      Data Array
```

5.12.2.4 TWO ARGUMENTS. The following example creates a task that activates a non-reentrant subroutine, NEWTSK, with two arguments, ARG1 and ARG2. NEWTSK performs non-file I/O thus requiring a TCB stack of 110 words.

```
          LDM     =LIST       Set List Address
          SVC     30          Create Task
          BRU     NEXT        Computation Continues
           .
           .
           .
LIST      DATA    11          Eleven Arguments in List
          DATA    ZERO        New Task Priority Same as Job Priority
          DATA    TASKID      TASKID of One
          DATA    NEWTSK      Pointer to NEWTSK
          DATA    MINUS1      No Register Arguments
          DATA    TCBSTK      TCB Stack of 110 Words
          DATA    ZERO        Create Active Task
          DATA    STKPTR      Stack for Argument List
          DATA    ZERO        No Work Space
          DATA    ZERO        No Volatile Arguments
          DATA    ARG1        Pointer to First Argument
          DATA    ARG2        Pointer to Second Argument
           .
           .
           .
                              Constants, Arguments, etc.

ZERO      DATA    0
TASKID    DATA    1
MINUS1    DATA    -1
TCBSTK    DATA    110
STKPTR    DATA    $+1
          BSS     3
ARG1      DATA    X,Y,Z
ARG2      DATA    I,J
          REG     NEWTSK
```

5.12.2.4 TWO ARGUMENTS AND WORKSPACE. The following example creates a task that activates a reentrant subroutine, RSUB, with two arguments, ARG1 and ARG2. RSUB performs file I/O, thus requiring 300 words of TCB stack. In addition, RSUB requires 10 words of remote data area for workspace.

```
          @LDM    =LIST       Set List Address
          SVC     30          Create Task
          BRU     NEXT        Computation Continues
           .
           .
           .
```

(listing continued on next text page)

(listing continued from preceding text page)

```
LIST     DATA   11          Eleven Arguments in List
         DATA   ZERO        New Task Priority Same as Job Priority
         DATA   TASKID      TASKID of Two
         DATA   RSUB        Pointer to RSUB
         DATA   MINUS1      No Register Arguments
         DATA   TCBSTK      TCB Stack of 300 Words
         DATA   ZERO        Create Active Task
         DATA   STKPTR      Stack for Argument List and Work Area
         DATA   MINUS1      Nonzero Signifies Work Area Supplied
         DATA   ZERO        No Volatile Arguments
         DATA   ARG1        Pointer to First Argument
         DATA   ARG2        Pointer to Second Argument
          •
          •
          •
                            Constants, Arguments, etc.

ZERO     DATA   0
TASKID   DATA   2
MINUS1   DATA   -1
TCBSTK   DATA   300
STKPTR   DATA   $+1
         BSS    14
ARG1     DATA   X,Y,Z
ARG2     DATA   I,J
         REF    RSUB
```

5.13 LOAD MEMORY IMAGE PHASE - SVC NUMBER 37

NOTE

SVC 37 is not normally used directly by a user program. Normally the overlay manager calls SVC 37 for the user as described in Section VIII for DXOLE.

```
         @LDM   =ARGLST     Set M-Register to List Address
         SVC    37          Execute Call
          •
          •
          •
ARGLST   DATA   3           3 Arguments
         DATA   MIPNUM      Address of Memory Image Phase Number
         DATA   LOADR       Address at Which to Load
         DATA   EPA         Entry Point Address
          •
          •
          •
MIPNUM   DATA   n           Memory Image Phase Number per DXOLE
                               Load Map
MODULE   BSS.   k           Overlay Area
LOADR    DATA   MODULE
EPA      BSS    1           Start Address for Execution
```

SVC 37 loads memory image phases directly from the load module file assigned to the running job. A memory image phase is a separate program segment that was produced by the DX980 Linkage Editor (DXOLE). DXOLE assigns a number to each memory image phase and outputs that number as part of the DXOLE load map. Refer to the DXOLE description in Section VIII of this manual for further details on load modules and memory image phases. The SVC 37 call applies to preplanned overlays. This means that the link editor must establish the load address relative to the main program (root). The overlay may only be loaded at that one relative address. On return from SVC 37 EPA contains the entry point address of the loaded phase.

5.14 LOAD AND RELOCATE MEMORY IMAGE PHASE - SVC NUMBER 38

```
        @LDM   =ARGLST   Set M-Register to List Address
        SVC    38        Execute Call
         .
         .
         .
ARGLST  DATA   3         3 Arguments
        DATA   MIPNUM    Address of Memory Image Phase Num-
                           ber
        DATA   LOADR     Address at Which to Load
        DATA   EPA       Entry Point Address
         .
         .
         .
MIPNUM  DATA   n         Memory Image Phase per DXOLE Load
                           Map
MODULE  BSS    k         Overlay Area
LOADR   DATA   MODULE
EPA     BSS    1         Start Address for Execution
```

SVC 38 transfers memory image phases to memory and relocates them within memory. The relocation map for the memory image phases is brought into the job extension area at the same time that a memory image phase is brought into the job area. The relocation map must, therefore, be considered when determining the size of the job extension area. The size of the relocation map can be determined by dividing the number of words in the memory image phase by sixteen (one map bit per phase word). This call applies to overlays that are not preplanned. Therefore, the link editor did not assign a fixed address to the overlay relative to the main program (root). The LOADR cell holds the load address of the module. This address may be determined dynamically. On return from SVC 38 EPA contains the entry point address of the loaded phase.

5.15 COMMAND SCANNER MODULE - SVC NUMBER 41

```
              CLDM = ARGLST  Set M-Register to List Address
              SVC      41     Execute Call
               .
               .
               .
    ARGLST    DATA     n      number of arguments
              DATA     CMDSTR command string
              DATA     KEY    key word area
              DATA     CTRL   control information
              DATA     PAKSTR packed string
              DATA     RESLAB reserved labels
 ------------------------------------------------------------
 |            DATA     MAXCH  number of characters to scan  |
 |                                                          |
 |                                             optional     |
 ------------------------------------------------------------
```

The Command Scanner SVC is a DX980 nucleus module that can be accessed through SVC 41. SVC 41 accepts free format command records and produces fixed format arrays. Table 5-5 describes the language syntax accepted by the command scanner.

5.15.1 EXTERNAL INTERFACE

The linking to SVC 41 is identical to the linkage for other SVC's. The M register points to an argument list. The first word of the list contains the number of arguments. Subsequent words contain the argument addresses.

5.15.1.1 INPUT. All arguments, except PAKSTR, must be initialized by the calling routine before issuing the SVC. The following parameters are necessary as input:

CMDSTR. The command string may be a variable length input record. If the MAXCH argument is specified, then the size of the command is specified via this argument; otherwise, the command string is assumed to be an 80 character input record, in which case if the actual string is less than 80 characters long, it should end with a period or semicolon, or have the remaining characters filled with blanks.

KEY. KEY is an array that holds the descriptors for the command plus all arguments in CMDSTR. The calling routine must zero the first word of the KEY array before invoking the SVC. If the Command Scanner later requests continuation records for a command, the first word contains a non-zero value. This value should not be changed until the calling routine wants to start a new command.

Table 5-5. Description of Command Language (Backus-Naur Format)

```
<command>::=<command string>.

<command string>::=<command identifier>|
                     <command identifier> <delimiter> <operand string>

<command identifier> ::= <label>| <blanks> <label>

<operand string> ::= <operand>
                   |<operand string> <delimiter> <operand>

<delimiter> ::=,|b̸ b̸<delimiter>| <delimiter> b̸|;

<blanks> ::=b̸<blanks>|b̸ | (no characters)
```

A command can extend over several input records. The first characters on the record will be ignored if CTRL word 5 is appropriately set. If a command extends to the next record, then the rightmost delimiter on the current record should be a semi-colon. If neither a period nor a semi-colon is present at the end of a record, then a period is assumed. Comments may appear between the period or semi-colon and the end of the record. The command identifier is compared against the labels in the Reserved Labels table. If the command identifier is found in the table, the identifier is treated as a Reserved Label; otherwise, the command identifier is treated as a label. Only the leftmost eight characters of the command identifier are significant.

```
<operand> ::= <label>| <number> | <expressions> | <range>
          |<empty operand>
          |<string> |<assignment>

<label> ::= <letter> | < label> <letter> | <label><decimal digit>

<letter> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
```

The length of labels is not restricted.

```
<number> ::= <sign><decimal integer>|<decimal integer>|<hex integer>

<decimal integer> ::= <decimal digit> |<decimal integer> <decimal digit>

<decimal digit> ::= 0|1|2|3|4|5|6|7|8|9

<sign> ::=+|-

<hex integer> ::=><hex digit>|<hex integer> <hex digit>
```

Table 5-5. Description of Command Language (Backus-Naur Format) (Continued)

<hex digit> ::= 0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F

Decimal integers must be in the range of -32768 to 32767. Hex integers must not contain more than four hex digits. Numbers larger than four hex digits must be described by strings.

<expressions> ::= <label> + <right side>

| <label> (<subscripts>)= <right side>

<right side> ::= <number> | <label> | <string> | (< subscripts>) |<range>

<subscripts> ::= <subscripts> , <script> | <script>

<script> ::= <number> | <label>

The number of subscripts allowed is not restricted.

<empty operand> ::= {the empty set}

An empty operand is generated for every occurrence of one of the following conditions:

- A pair of commas separated by no other characters or only by blanks.
- A comma and a period separated by no other characters or only by blanks.
- A comma and a semi-colon separated by no other character or only by blanks.

<subscripted expression> ::= <label> (< subscripts>)= <right side>

<subscripts> ::= <subscripts> , <script> | <script>

<script> ::= <number> | < labels>

The number of subscripts allowed is not restricted.

<string> ::= | <substring> |

<substring> ::= <character> | < substring> <character>

<character> ::= Any USASCII character. If a "|" is to appear in a <substring> , then "||" should appear in the input to the scanner.

The length of the strings is not restricted.

<assignment> ::= <label> : = <label>

<range> ::= <number> : <number>

CTRL. CTRL is a six word array, that must be initialized as follows:

- Word 0 - Number of Characters Reserved for PAKSTR
- Word 1 - Number of Words Reserved for KEY
- Word 2 - Not Initialized by the Calling Routine. Used by Command Scanner for Workspace
- Word 3 - Not Initialized by the Calling Routine. Used by Command Scanner for Workspace
- Word 4 - Number of Labels in RESLAB
- Word 5 - The Column Number in CMDSTR Where Scanning is to Start; the First Column in CMDSTR is Column Zero.

MAXCH. MAXCH is a one-word field containing the number of characters to scan. This argument is optional and if not specified the command string size is assumed to be 80 characters.

5.15.1.2 OUTPUT. SVC 41 changes PAKSTR and KEY, plus two words of workspace. The following paragraphs describe the effects on these two arguments.

PAKSTR. PAKSTR is used for storage of alphanumeric fields that were retrieved from CMDSTR. The fields are packed together under control of the KEY array.

KEY. The first word of KEY contains a completion code. The remaining words contain a translation of the command string. The value of the completion code indicates one of the following conditions:

- Normal End of Scan - A complete command has been successfully decoded.
- Continuation Requested - The command extends across more than one record. The value of the completion code minus one is the number of continuation records previously read. KEY array contains descriptors for all fields of CMDSTR that have been scanned.
- Error - Scanning of CMDSTR is terminated. The value of the completion code indicates which error has occurred. Table 5-6 describes the error numbers. The KEY array contains descriptors for all fields of CMDSTR that have been scanned. However, the last descriptor may be incomplete or erroneous.

Table 5-6. Error Codes

Code	Explanation
401	Overflow of the keyword area.
402	Overflow of packed strings character string.
403	The right-hand side of an expression or range is missing.
404	Unrecognizable or illegal subscript.
405	Missing delimiter after command identifier.
406	Number is larger than 16 bits.
407	Operand starts with illegal character.
408	Illegal digit in number.
409	Missing delimiter after operand.
410	Missing delimiter after subscript.
411	Illegal character precedes command.
412	ITS Run command does not contain a label or an expression.
413	Missing equal sign following colon in assignment.
414	The right-hand side of an assignment is missing.
415	Too many equal signs in expression.
416	Negative number of characters specified for PAKSTR by CTRL word 0.
417	Non-positive number of words specified for KEY by CTRL word 1.
418	Non-positive number of labels specified for reserved labels list (RESLAB) by CTRL word 4.
419	The starting column for the scan, specified by CTRL word 5, does not fall into the range of zero to seventy-nine.

Following the completion code is a description of the translation of CMDSTR. The second word of KEY contains the number of operands detected in CMDSTR. The remainder of KEY contains descriptors for the operands. The format for KEY is illustrated in figure 5-4. The operand descriptors are described in figure 5-5. The eight possible descriptor types are: Label, Number, Expression, Range, Reserved Label, Empty Operand, String, and Assignment.

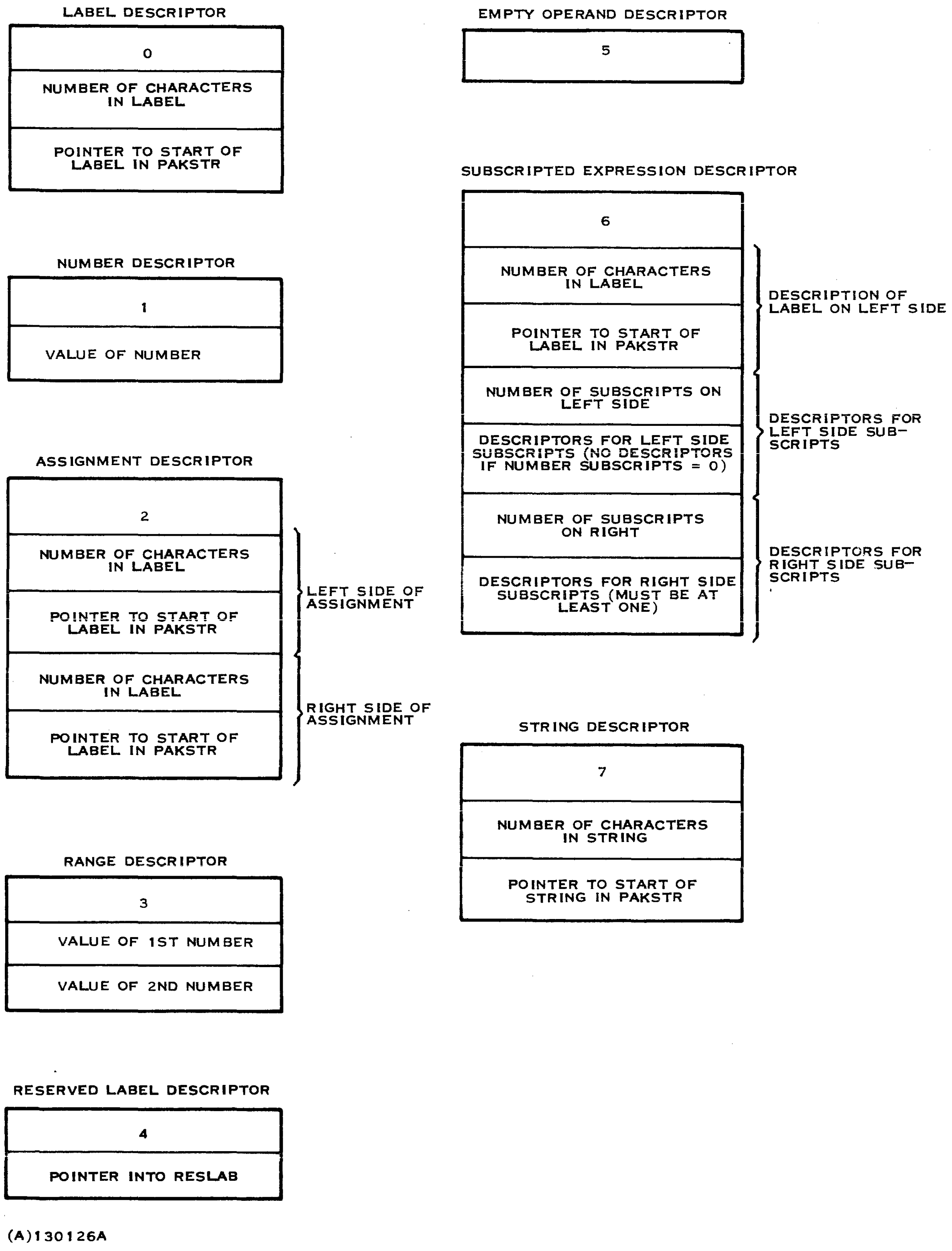

LABEL DESCRIPTOR
0
NUMBER OF CHARACTERS IN LABEL
POINTER TO START OF LABEL IN PAKSTR
NUMBER DESCRIPTOR
1
VALUE OF NUMBER
ASSIGNMENT DESCRIPTOR
2
NUMBER OF CHARACTERS IN LABEL
POINTER TO START OF LABEL IN PAKSTR
NUMBER OF CHARACTERS IN LABEL
POINTER TO START OF LABEL IN PAKSTR
LEFT SIDE OF ASSIGNMENT
RIGHT SIDE OF ASSIGNMENT
RANGE DESCRIPTOR
3
VALUE OF 1ST NUMBER
VALUE OF 2ND NUMBER
RESERVED LABEL DESCRIPTOR
4
POINTER INTO RESLAB
EMPTY OPERAND DESCRIPTOR
5
SUBSCRIPTED EXPRESSION DESCRIPTOR
6
NUMBER OF CHARACTERS IN LABEL
POINTER TO START OF LABEL IN PAKSTR
NUMBER OF SUBSCRIPTS ON LEFT SIDE
DESCRIPTORS FOR LEFT SIDE SUBSCRIPTS (NO DESCRIPTORS IF NUMBER SUBSCRIPTS = 0)
NUMBER OF SUBSCRIPTS ON RIGHT
DESCRIPTORS FOR RIGHT SIDE SUBSCRIPTS (MUST BE AT LEAST ONE)
DESCRIPTION OF LABEL ON LEFT SIDE
DESCRIPTORS FOR LEFT SIDE SUB-SCRIPTS
DESCRIPTORS FOR RIGHT SIDE SUB-SCRIPTS
STRING DESCRIPTOR
7
NUMBER OF CHARACTERS IN STRING
POINTER TO START OF STRING IN PAKSTR
(A)130126A

Figure 5-5. Templates for Descriptors in KEY Array

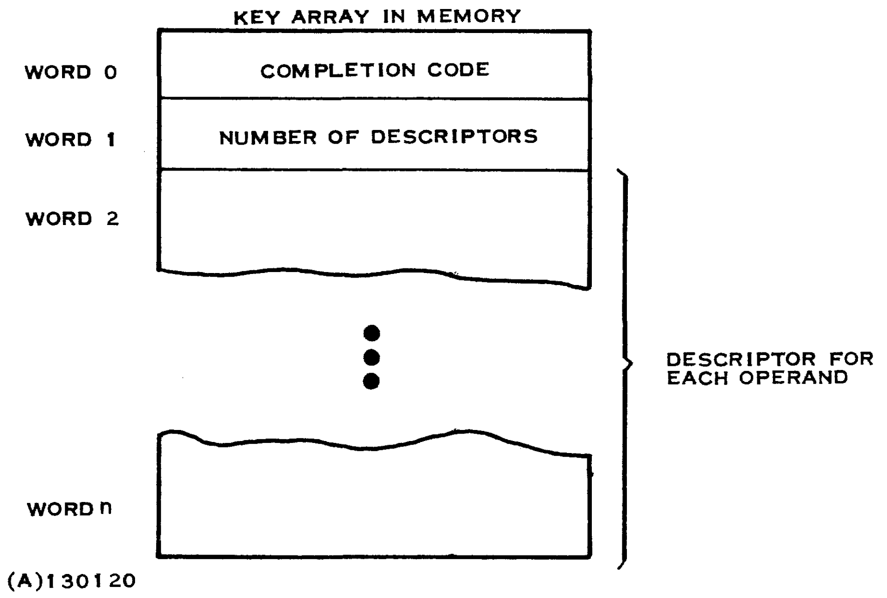


Figure 5-4. Format of Key Array After Return from SVC 41

5.15.2 SVC 41 EXAMPLE

The ITS Supervisor uses SVC 41 to decode commands from each terminal. The arguments could be declared as follows:

```
CMDSTR   BSS   80            Terminal command line
KEY      BSS   50            Key array
CTRL     DATA 80             Number of characters in PAKSTR
         DATA 50             Number of words in KEY
         BSS   2             Workspace for SVC 41
         DATA 12             Twelve valid commands
         DATA 0              START SCAN at 0
PAKSTR   BSS   80            Packed string
RESLAB   DATA 'LOGON '       Reserved labels
         DATA 'LOGOFF'
         DATA 'RUN   '
         DATA 'STATS '
         DATA 'EDIT  '
         DATA 'ENTER '
```

(Listing continued on next text page)

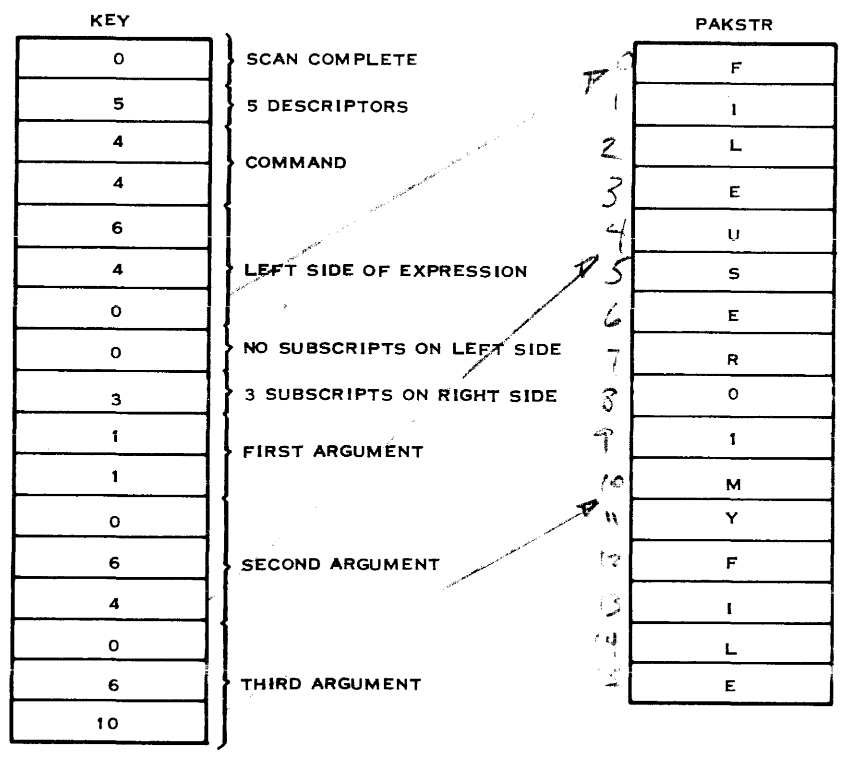
(Listing continued from preceding text page)

```
        DATA 'JOB     '
        DATA 'DELETE'
        DATA 'F1      '
        DATA 'F2      '
        DATA 'A1      '
        DATA 'A2      '
```

If a terminal user enters the following command:

    EDIT FILE=(1,USER01,MVFILE).

The command is stored in CMDSTR by an ITS I/O routine. The ITS supervisor issues SVC 41 with the standard argument linkage for SVC's. After control returns from the command scanner, PAKSTR and KEY contain the values illustrated in figure 5-6.



(A)130122

Figure 5-6. Resulting Contents of KEY and PAKSTR

5.16 WAIT FOR I/O - SVC NUMBER 43

```
        @LDM   =ARGLST    Set M-Register to List Address
        SVC    43         Execute Call
         .
         .
         .
ARGLST  DATA   1          1 Argument
        DATA   PRB        Address of PRB Used for Initiate Call
```

The Wait for I/O SVC is used in conjunction with an Initiate I/O data transfer (see Section III) or with a multitasking program that requires synchronization of I/O and processing tasks. For Initiate I/O calls the calling program continues execution during the actual I/O transfer. If processing must be discontinued at some point in the program until a requested I/O transfer is complete, the Wait for I/O SVC is issued. Program execution is then suspended until the I/O transfer is complete. If the I/O is already complete, processing proceeds without suspension.

The procedure is similar for multitasking programs except that a separate task issues an Execute I/O rather than an Initiate I/O call. Other tasks issue a Wait for I/O SVC to synchronize the I/O with processing portions of their program.

5.17 ALLOCATE RESOURCE - SVC NUMBER 49

```
              @LDM    =ARGLST    Set M-Register to List Address
              SVC     49         Execute Call
               .
               .
               .
     ARGLST   DATA    2          2 Arguments
              DATA    JERR       Address or Return of Error Code
              DATA    JLDT       Address or Resource Assignment Block
               .
               .
               .
     JERR     BSS     1          0=Ok, 39=Device Offline, 88=Device Un-
                                   available, > 0= Error
     JLDT     BSS     20         Assignment Block
```

The Allocate Resource SVC assigns a logical unit to a device or file from within user code at runtime. This feature permits extension of the job assignments that were made with JCL (refer to Section II for a discussion of JCL assignments). In response to SVC 49, the operating system checks the availability of the requested resource. If the resource is available, the operating system assigns that resource to the specified user job LUN. If the specified LUN matches a previous user LUN, the new assignment supercedes the old assignment.

CAUTION

When using SVC 49 with a high priority program, ensure that lower priority programs have not reserved the requested resource. Failure to observe this precaution may deadlock the system until the lower priority program releases the device.

Input parameters for SVC 49 are one word for error code return, plus another word group for the resource assignment block. The error code returned in the first parameter is one of the following quantities:

- 0 = Allocation Made
- 39 = Device Offline
- 88 = Device Already Assigned/Committed
- other (See Appendix A for error codes)

The resource assignment block (JLDT) describes the file or device to be assigned. The length of this block varies. It is four words long for a device assignment, 13 words long for an old file assignment, and 20 words long to define a new file or replace an old file. Table 5-7 lists the word and bit assignments for all fields of the resource assignment. Unused fields should contain zeros for compatability with future uses.

Table 5-7. Resource Assignment Block (JLDT) Format

Word	Bit	Field Description
0		Flag Word
	0	Not used
	1	Device(0)/file(1) flag
	2	Not used
	3	Exclusive(0)/shared(1) access flag
	4-5	Not used$(00)_2$
	6	No pass(0)/pass(1) resources flag
		Remainder of bits applicable to file assignments only:
	7	No delete(0)/delete(1) file flag
	8-9	Disposition flag:
		00_2 = Assign old file
		01_2 = Define new file
		10_2 = Replace old file
		11_2 = Illegal disposition
	10-11	File type flag
		00_2 = Illegal type
		01_2 = Linked sequential
		10_2 = Relative record
		11_2 = Key indexed
	12	Permanent(0)/temporary(1) file flag
	13-15	Not used $(000)_2$
1		
	0-7	Not used
	8-15	Logical unit number (LUN)
2		Device index (decimal 1-256 corresponding to device required). Actual device indexes for the physical devices can be obtained through the List Device (LD) command during IPL.
3		Reserved for future use
		Remainder of words applicable to file assignments only:
4		Number of buffers to be used for file I/O
5-7		File owner user ID (6 USASCII alphanumeric characters)
8-10		File name (6 USASCII alphanumeric characters)
11-12		File password (4 USASCII alphanumeric characters)

Table 5-7. Resource Assignment Block (JLDT) Format (Continued)

Word	Bit	Field Description
		Remainder of words applicable to file define/replace only:
13		Integrity code
	0-2	Read code
		100_2 = Creator only
		110_2 = Password owner
		111_2 = Any user
	3-5	Write code-same options as read code
	6-8	Delete code-same options as read code
	9-11	Execute code-same options as read code
	12-15	Not used
14		Initial file size (in tracks)
15		First disc address for allocation (in tracks)
16		Physical record length (in words)
17		Maximum file size (in tracks)
18		Logical record length (in characters) for relative record files only
19		Key length (in characters) for key indexed files only

5.18 DEALLOCATE RESOURCE - SVC NUMBER 51

```
        @LDM    =ARGLST   Set M-Register to List Address
        SVC     51        Execute Call
         .
         .
         .
ARGLST  DATA    2         2 Arguments
        DATA    JERR      Address for Return of Error Code
        DATA    JLUNO     Address for Logical Unit Number
         .
         .
         .
JERR    BSS     1         0=Ok, >0=error
JLUNO   DATA    n         Logical Unit for Released Resource
```

The Deallocate Resource SVC removes an assignment of a device or file from a user program at runtime. The released resource may have been allocated to the job step initially by JCL or may have been allocated during runtime with a SVC 49 call.

The input parameters consist of a word for return of error code from DX980 (0=Ok, >0=improper LUN), plus a word containing the LUN(1-254) to be deallocated.

5.19 GET PROGRAM LIMITS - SVC NUMBER 98

```
        @LDM    =ARGLST    Set M-Register to List Address
        SVC     98         Execute call
         .
         .
         .
ARGLST  DATA    1          1 Argument
        DATA    LIMITS     Address for Limits Depository
         .
         .
         .
LIMITS  BSS     2          Lower and Upper
```

SVC 98 is identical to SVC 3 (Get Memory Limits) except that the memory limits returned in LIMITS correspond to the area between the last word of the user program and the end of the job area. This SVC determines the amount of memory remaining for work area beyond the actual program code. When using this SVC in conjunction with preplanned overlays, LIMIT (0) contains the first word beyond the longest overlay; when used with non-preplanned overlays, LIMIT (0) contains the first word beyond the root segment.

5-20 USER START JOB - SVC NUMBER 129

```
        @LDM    =ARGLST    Set M-Register to List Address
        SVC     129        Execute call
         .
         .
         .
ARGLST  DATA    1          1 Argument
        DATA    JSB        Address for Job Structure Block
```

The User Start Job SVC presents independent job steps to the system for execution. The input to this SVC consists of a Job Structure Block (JSB). To initiate a multistep job string, the user must issue a separate User Start Job SVC for each job step of the string. The JSB for a single job step consists of a 26 word preamble plus one resource assignment block (JLDT) for each resource to be initially assigned to the job step. Refer to the Allocate Resource call (SVC 49) for a description of the JLDT. Table 5-8 lists the word and bit assignments for the JSB preamble. Jobs may be started from the system console or via batch or interactive input through the subsystems. All of these methods utilize JCL. Jobs should be initiated in those ways rather than through SVC 129. User Start Job should be used only when other alternatives are unsatisfactory. If the system detects an error in the structure of a job being submitted, it dismisses all previous job steps and aborts the job.

Table 5-8. Job Structure Block (JSB) Preamble Format

Word	Bit	Field Description
0		Total length of JSB (for one job step, including all JLDT's)
1		Flagword
	0	Last job step in sequence of steps within a job 0=No (More SVC 129's are forth coming in this job) 1=Yes (This is the last SVC 129 for this job)
	1	Privileged mode 0=Unprivileged mode <PROT> 1=Privileged mode <PRIV>
	2-15	Not used
2-4		User I.D. - <userid> (6 alphanumerics in USASCII)
5-7		Job name - <jsname> (6 alphanumerics in USASCII)
8		Job step number within job string (1 to 15)
9		Job priority - <jsprty> (1 to 31)
10		Number of task priority levels within the job - <nprty> (1 to 31)
11		User partition size - <jarea> (load module + user buffer)
12		Job size - <jearea> (I/O buffers + job internal system control area)
13		Stack size of initial task TCB and default size for subsequent tasks - <stksiz>
14		Time limit for job step (in seconds); -1 indicates no time limit
15-16		Two words, initially zero, for use by DX980
17		Load module volume ID - <volume>
18 19 20		Load module user I.D. (6 alphanumerics USASCII) - <fileid>
21 22 23		Load module file name (6 alphanumerics in USASCII) - <filnam>
24 25		Load module password (4 alphanumerics in USASCII) - <pswd>
26 ⋮		JLDT's

SECTION VI

BATCH PROCESSING SUBSYSTEMS

6.1 GENERAL

Three separate subsystems provide batch processing capabilities to the DX980 user. These subsystems are Batch Input Reader (BIR), Batch Input Spooler (BIS), and Batch Output Spooler (BOS). The two input subsystems allow submission of user programs through a card reader. These subsystems accept a data stream of intermixed data and control cards. However, the BIS subsystem stores the input information on disc prior to activation of the job, whereas the BIR subsystem allows the executing program to read directly from the input peripheral device. Similarly, the BOS subsystem reads program output from a disc and transfers it to a low speed output device such as a line printer. The subsystems may run concurrently as long as they each have separate peripheral devices. Normally, the operator starts one subsystem and allows it to run continuously. Any of the subsystems, however, may be stopped at any time.

The subsystems are structured and invoked in the same manner as a job. Each subsystem requires a JCS that identifies the peripheral devices and the load module file. The operator activates the subsystem with JCL Job and Run commands from the system console. The following paragraphs describe the operation of each subsystem and provide examples of batch processing sequences.

6.2 BATCH INPUT READER (BIR)

The BIR subsystem functions as a single program input stream. That is, the program reads any required input data directly from the input device. Therefore, the batch input device may be unavailable to other users until that program terminates. The BIR subsystem does, however, decrease the I/O overhead associated with spooling. Figure 6-1 illustrates a typical deck structure for BIR input. The following control commands govern BIR processing:

//JOB	JCL Job command
//RUN	JCL Run command
//DATA	BIR Data Control command
/$	BIR End of Job command

6.2.1 JOB COMMAND

The Job command used with BIR is identical to the JCL Job command and defines the start of the job. This command is described in Section II of this manual.

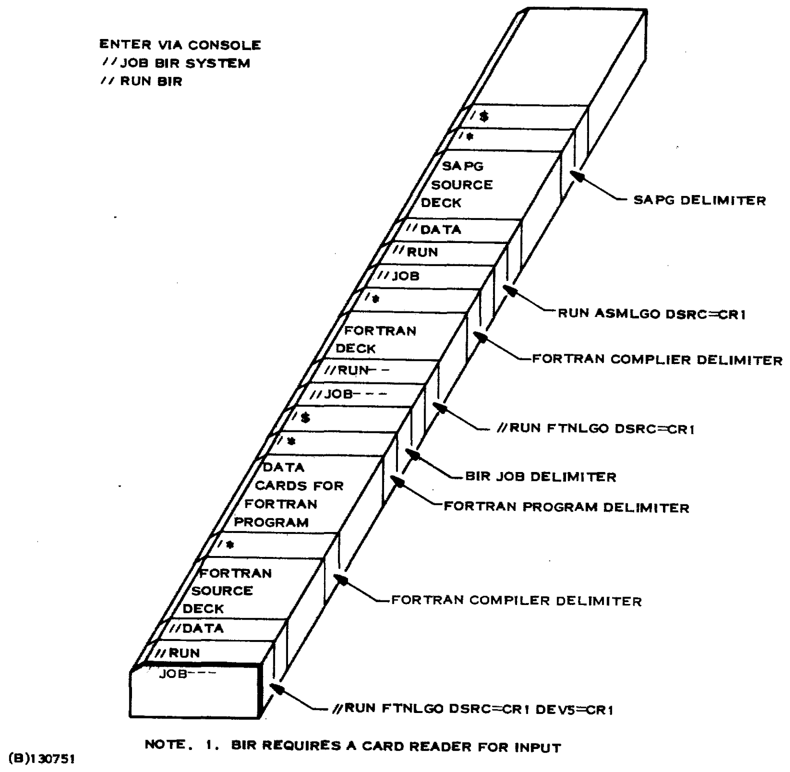


Figure 6-1. BIR Input Deck Structure

6.2.2 RUN COMMAND

The Run command used with BIR is identical to the JCL Run command and defines the JCS that loads and executes the requested job. This command is described in Section II of this manual.

6.2.3 DATA CONTROL COMMAND

The Data command (//DATA) identifies the start of the user data input. This command must be included following the Run command and before the data stream whenever data appears in the job input to BIR. The Data command for BIR contains only the six character symbol, //DATA. Other information may be included, but BIR ignores that input. Only one DATA command is allowed in a BIR job. The data deck must be organized in the order expected by the user program. The data deck cannot contain intermixed control cards.

6.2.3.1 SYSIN ASSIGNMENT. If the JCS identified in the Run command contains an assignment to the generic device, SYSIN (/ASSIGN 6 SYSIN.), then the input job must contain a Data command. When BIR reads the Data command, it deallocates the input device and suspends operation until the user program terminates. The user program then executes and is assigned the same input device. The program acquires all data from this peripheral as it is needed until processing terminates. At that point, BIR reactivates and scans the input from the peripheral device for a new Job command.

6.2.3.2 DEVICE ASSIGNMENT. If the user job contains an assignment to the same I/O device that BIR is using, BIR again deallocates that device and suspends processing until the user job completes. For example, if device CR1 is assigned as the input device for BIR and a user job desires input from that device, then the user job input must also contain a Data command following the Run command, plus the associated input data. BIR then processes the input as if the JCS for the job had assigned the input to SYSIN. The JCS may contain many assignments to either SYSIN or the BIR input device. All such assignments designate the same device.

6.2.4 END OF JOB COMMAND

The End of Job command (/$) follows the last data input of the user program and precedes the next Job command in the input stream. This command resets BIR for the next job and signals the end of the current job.

6.3 BATCH INPUT SPOOLER (BIS)

The BIS subsystem receives input from the assigned peripheral device and stores that input on a disc prior to the start of the related program. This operation is called spooling. When the program starts, it can access data from the high speed disc, rather than from the low speed input peripheral.

This system allows shorter program execution time and submission of other jobs while the first job is executing. Figure 6-2 illustrates a typical deck structure for BIS input. The following control commands govern BIS processing:

//JOB	JCL Job command
//RUN	JCL Run command
//DATA	BIS Data Control command
/$	BIS End of Job command

6.3.1 JOB COMMAND

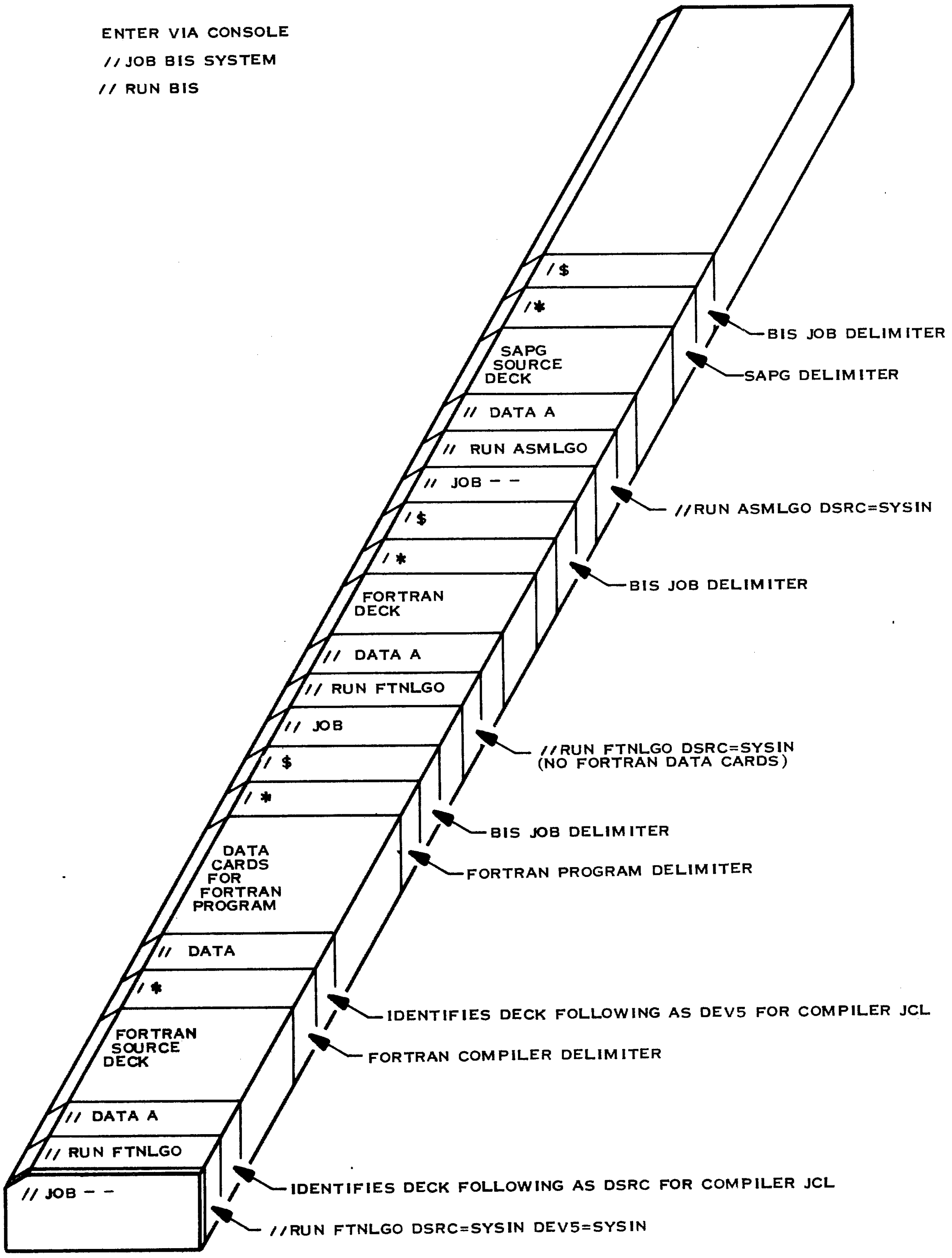
The Job command used with BIS is identical to the JCL Job command and defines the start of the job. This command is described in Section II of this manual.

6.3.2 RUN COMMAND

The Run command used with BIS is identical to the JCL Run command and defines the JCS that loads and executes the requested job. This command is described in Section II of this manual.

6.3.3 DATA CONTROL COMMAND

The Data command (//DATA) identifies the start of the user data input. This command must be included following the Run command and before the data



(B)130752

NOTES. 1. BASED ON A MINIMUM CONFIGURATION AND SUPPLIED FTNLGO SEQUENCE
2. BIS REQUIRES CARD READER FOR INPUT

Figure 6-2. BIS Input Deck Structure

stream whenever data appears in the job input to BIS. More than one Data command may appear in the input to BIS. Each Data command is in the form:

```
//DATA,[<p1>],[<p2>].
```

6.3.3.1 PARAMETER $<p_1>$. Parameter $<p_1>$ is optional. If supplied, this field contains the letter J. If BIS detects the letter J in this field, it spools all cards following the Data command onto the disc including any subsequent Data commands. In this mode the End of Job command (/$) is the only job delimiter. If the letter J is not supplied in this field, BIS interprets subsequent Data commands as well as End of Job commands as data deck delimiters. This allows the data input to be subdivided for use by different portions of the program. If the Data command contains a J entry, it must be the last Data command in the input stream since the remaining cards are not interpreted.

6.3.3.2 PARAMETER $<p_2>$. Parameter $<p_2>$ specifies the type of data conversion to be used during spooling. Three entries are possible for this parameter:

- A The letter A specifies conversion of the input data to USASCII before storing the data on disc.
- B The letter B specifies conversion of the input data to binary before storing the data on disc.
- D The letter D specifies storing the data directly on disc as it is read from the input device.

If this parameter is not included, BIS defaults to converting the input data to USASCII code for storage on the disc.

6.3.3.3 ASSIGNMENTS. All assignments for input data under BIS must be made to SYSIN. BIS does not allow an assignment to the device that it is using. Programs may have more than one assignment to SYSIN. Each assignment to SYSIN must have a separate data deck preceded by a Data command. The input data sets must be organized in the same order as their respective assignments. The first assignment to SYSIN in the JCS reads the first data set, the second assignment reads the second data set, etc.

6.3.4 END OF JOB COMMAND

The End of Job command (/$) follows the last data input of the user program and precedes the next Job command in the input stream. This command resets BIS for the next job and signals the end of the current job. If a //DATA J. Data command occurred in the job, the End of Job command is the only command that terminates data input.

6.4 BATCH OUTPUT SPOOLER (BOS)

The BOS subsystem allows the user program to store output data on a high speed disc file during program execution. The subsystem then retrieves the data from the disc when the program is complete and writes the data on the designated output device. This output spooling feature can be used only if a System Output Queue (SOQ) file is installed on the system disc. The operating system verifies the existence of SOQ at each initial program loading (IPL). If SOQ is available, then any job may use the output spooling feature by assigning the output LUN to SYSOUT. All data written to SYSOUT is stored on the disc until the job string terminates. At that time if BOS is running and not busy, it begins to output the data to the assigned device. If BOS is not running, it will print out the data as soon as it is activated with the Job and Run commands.

A job step may assign up to 26 LUNs to SYSOUT; however, the total number of LUNs assigned to SYSOUT in a job string is also limited to 26. When the data is spooled, it is separated on the disc according to LUN. Therefore, when the data is printed out following completion of the job, all data written to one LUN will be printed before any data is printed that was assigned to a different LUN. This system provides a degree of organization to the output data that cannot be achieved by assigning multiple LUNs to the same output device directly. In that case the data is printed chronologically as produced by the program. When the data is spooled, it can be divided into functional groups.

6.5 BIR AND BIS EXAMPLES

1.
```
//JOB     JBNAME    USER01
//RUN     JSBKEY
/$
```

The JOB command specifies a user job with jobname JBNAME under userid USER01. The JSB file defaults to the standard system JSB file under DX980.

The RUN command directs the system to retrieve the JSB with the name JSBKEY from the system JSB.

2.
```
//JOB     JBNAME    USER01     FILE=(1, USER2, FILEX)
//RUN     JSBKEY
/$
```

This example is identical to the previous example except that the JSB file is on disc unit number one with userid USER2 and filename FILEX.

3.
```
//JOB     JBNAME    USER01
//RUN     JSBKEY    INPUT=CR2   FILE=(USER3, FILE2)
```

The keyword INPUT, established at JCL translation time, is overridden with device assignment CR2 representing card reader number two. CR2 is not the batch input peripheral. Keyword FILE is overridden to specify a specific disc file.

4.
```
//JOB        JBNAME      USER01
//RUN        JSBKEY      INPUT=SYSIN
//DATA       A
   :
  data
   :
/*
/$
```

Input is from the device assigned to the batch subsystem. If executed under BIS, data is converted to USASCII before spooling. BIR ignores the letter A in the Data command.

5.
```
//JOB        JBNAME      USER01
//RUN        JSBKEY      INPUT=CR1
//DATA  ,,   A
   :
  data
   :
/*
/$
```

If CR1 is the system input device, BIS will not recognize this job. BIR ignores the letter A in the Data command.

6.
```
//JOB        JBNAME      USER01
//RUN        JSB002      INPUT=SYSIN
//DATA       J,A
   :
  data
   :
/*
//DATA  B
   :
```

(listing continued on next text page)

```
  data
   .
   .
   .
/*
/$
```

For BIS the //DATA B card is spooled to disc as data because of the 'J' on the first //DATA card. BIR also ignores the //DATA B card, because the first Data command suspended the system to wait for job termination (/$).

6.6 BIS EXAMPLE

```
//JOB       JBNAME     USER01
//RUN       JSB001     INPUT1=SYSIN   INPUT2=SYSIN
//DATA  A
    .
    .
    .
  data (a)
    .
    .
    .
/*
//DATA  B
    .
    .
    .
  data (b)
    .
    .
    .
/*
/$
```

Data (a) will be with INPUT1 and data (b) will be with INPUT 2.

SECTION VII

INTERACTIVE TERMINAL SUBSYSTEM

7.1 OVERVIEW

The Interactive Terminal Subsystem (ITS) is that portion of DX980 that supports interactive peripherals such as teleprinters and full duplex CRTs. In particular, a single version of ITS allows intermixing in any configuration of the following terminals:

- Model 733 ASR
- Model 733 KSR
- Model 33 ASR
- Full duplex 912 CRT

The ITS and the terminal user communicate with input commands and output data or messages. The primary difference in the type of terminal, from the terminal user's viewpoint, is the amount of information that can be displayed. Teleprinters provide a single display line whereas CRTs provide up to 24 display lines.

7.1.1 CRT DISPLAYS

Under ITS a full duplex CRT may be supported either as a single line or a multi-line device. Single line or multi-line support is determined when building the PDT for the CRT at IPL time. If single line mode is selected, the CRT functions exactly as a teleprinter. If multi-line support is specified, on-screen editing allows convenient operations not possible with a teleprinter. In this section references to a CRT applies specifically to a CRT operating in multi-line mode. The CRT should be regarded as a teleprinter if single line mode is to be used.

The ITS and the terminal user communicate using input commands and output data or messages. The first line of a CRT screen is dedicated to commands and error messages. The remainder of the screen displays data. The operator must enter requests to ITS in line 1. Each request (command) to ITS must be followed by a period or have blanks (spaces) for all remaining positions in line 1. Depending on the type of request, the operator may also enter lines of data prior to notifying ITS that the screen is ready for processing. The operator can modify data on the screen with the cursor positioning keys in conjunction with the data keys. Any data change is echoed to the screen and stored simultaneously in the CRT buffer in the main memory. The effective screen size for on-screen editing can be specified within a range of 2 to 24 lines. The size impacts the amount of main memory required to support the terminal since each terminal must have a main memory buffer of 40 words per display line. When the operator has completed the message on the screen, he transmits the message to the system by pressing carriage return (CR).

7.1.2 TELEPRINTERS

Communication between the ITS and a teleprinter is in a one-line format. Each input line must be entered completely before notifying ITS to accept the input. ITS responds with data one line at a time. The operator keys in a request to ITS at the terminal and terminates it by depressing carriage return (CR). If terminal data is required to complete the request, ITS returns control to the terminal and rings the bell. At that time the user enters a data line and presses CR. This procedure continues until the request is satisfied. Mistakes made during command or data entry can be corrected by pressing CRTL H repeatedly to backspace until the carriage is positioned over the character in error. Starting at that point, the remainder of the line can be reentered correctly. When the operator presses the first reentry key, the DX980 terminal handler issues a line feed before echoing the character. Thus, the corrected data appears immediately below the original data. If this technique is employed several times, the data line resembles a staircase. Pressing CTRL N displays the entire data line on a single line. If the line is uncorrectable, the terminal user can start over by pressing RUBOUT and reentering the line.

7.1.3 ITS TERMINAL ASSIGNMENTS

ITS runs as a privileged program and can be activated via the RUN command from the system console. (The system console can be any data terminal device.) The terminal assignments for ITS are normally incorporated into the JCL sequence for ITS, but can be overridden in the Run command at submission time by specification of Ti=<devnam>. For this notation, i is in the range of 1 through the number of terminals at the installation and <devnam> is the mnemonic representation for the subject terminal. Assignments may be made to the system console (SC) and to the dummy device (DUMMY). Any program, including ITS, can assign the system console as one of its devices. Thus it can serve a dual purpose both as a system console and as a user terminal. When assigned to a program, the console displays the message USER MODE to notify the operator that the console has switched modes. At that time the operator can use the console as any other terminal. Pressing CTRL O switches the console back to system mode. After completion of the system duties, the console can be switched back to user mode with CTRL U. Assigning a terminal to DUMMY is a useful technique for starting ITS when one of the normally active terminals is inoperative. This assignment affectively deletes the corresponding assignment in the original JCL sequence. The following sample entries illustrate initiation of ITS from the System Console:

(1)	//RUN ITS	Start ITS with default assignments. The "//" is displayed in response to pressing the escape key (ESC).

(2) //RUN ITS T1=SC	Start ITS with an assignment to the system console in addition to the default assignments to other terminals. T1 could have been given a default assignment of DUMMY in the expanded JCS which would require positive action for assignment to the system console.
(3) //RUN ITS T1=SC, T3=DUMMY	Same as example (2) except that the terminal normally assigned to T3 is inoperative.

7.1.4 ITS MEMORY REQUIREMENTS

The memory requirements of ITS vary depending upon the number of terminals, the type of terminals, and whether the terminals are going to use the Remote Job Entry capability of ITS. ITS requires approximately 6000 words of JAREA, plus 125 words of JAREA per (one-line) terminal and 150 words of JEA per terminal. If the terminal is a multi-line CRT, then 40 words per display line must be added to the JAREA. If the terminal is going to use the Remote Job Entry capability, then 500 words must be added to the JAREA and 350 words must be added to the JEA.

7.1.5 REMOTE TERMINALS

After ITS is activated, a read is issued to each assigned terminal. This command rings the terminal bell and activates the terminal for input. If the assigned terminal is connected through a telephone data set, the data set is conditioned so that the terminal is immediately activated when the telephone connection is made. The procedure for establishing communications between a telephone dataset and a terminal equipped with an acoustic coupler is as follows:

1) Dial the assigned number and wait for the data set tone (a high pitch squeal).
2) Plug the handset into the acoustic coupler (the acoustic coupler must have the Mode switch set on "Full Duplex"; the terminal Mode switch must be set to "Line").
3) Terminal is ready for use.

7.1.6 LOGON

When communication is established between ITS and a terminal, the terminal user must enter a Logon command to gain access to ITS facilities. The format of the Logon command is as follows:

LOGON <userid> <acctno>

The Logon command converts a terminal from the inactive state to the ready state. When the terminal is in the ready state, all the services of ITS become available to the user. The notation <userid> and <acctno> correspond to user identification and account number respectively. Depending on the installation, a future enhancement to ITS will use three parameters for validation against a file of acceptable combinations and recording along with the elapsed time of an ITS session for accounting purposes. The present version of ITS requires them for syntactic validation. The <userid> field is limited to six characters, the first of which must be alphabetic. The <acctno> field is a positive integer that must be less than 32,768. All ITS command operands may be separated by a single comma, one or more blanks, or a comma and one or more blanks.

7.1.7 OTHER ITS COMMANDS

After the Logon command has been validated, the message 'READY' is displayed on the terminal. At this point the following commands are valid (brackets denote optional fields):

- LOGOFF
- EDIT FILE=(<volume>, <fileid>, <filnam> [,<pswd>])
  [LRECL=<lrchar>]
- ENTER FILE=(<volume>, <fileid>, <filnam>[,<pswd>])
  [LRECL=<lrchar>] [EXTEND]
- JOB <jsname> <userid> [FILE=(<volume>,<fileid>,<filnam>[,<pswd>])]
- RUN <jcsnam> . . .
- STATUS [<jobnum>]
- DELETE FILE = (<volume>,<fileid>,<filnam>[,<pswd>])

The Edit and Enter commands gain access to the Interactive File Editor (IFE). The Job and Run commands enter the Remote Job Entry (RJE) facility. The Stat command accesses the Computer Status Display (CSD) facility. The Logoff command returns the terminal to the inactive state.

7.2 INTERACTIVE FILE EDITOR

The Interactive File Editor (IFE) is an integral part of the Interactive Terminal Subsystem (ITS). It supports teleprinters and full duplex CRTs. IFE allows the user to display, insert, delete or replace records from the file, as well as create an entirely new file from the editing terminal. Two utility programs, BLDEDT and DXCOPY, construct the file for editing, and transfer the edited file back to the user file. Two editing commands, ENTER and EDIT, add or delete records from the file. Figure 7-1 illustrates the data transitions that may occur within the scope of file editing. IFE operates only on key indexed files. The file must be constructed such that the keys are

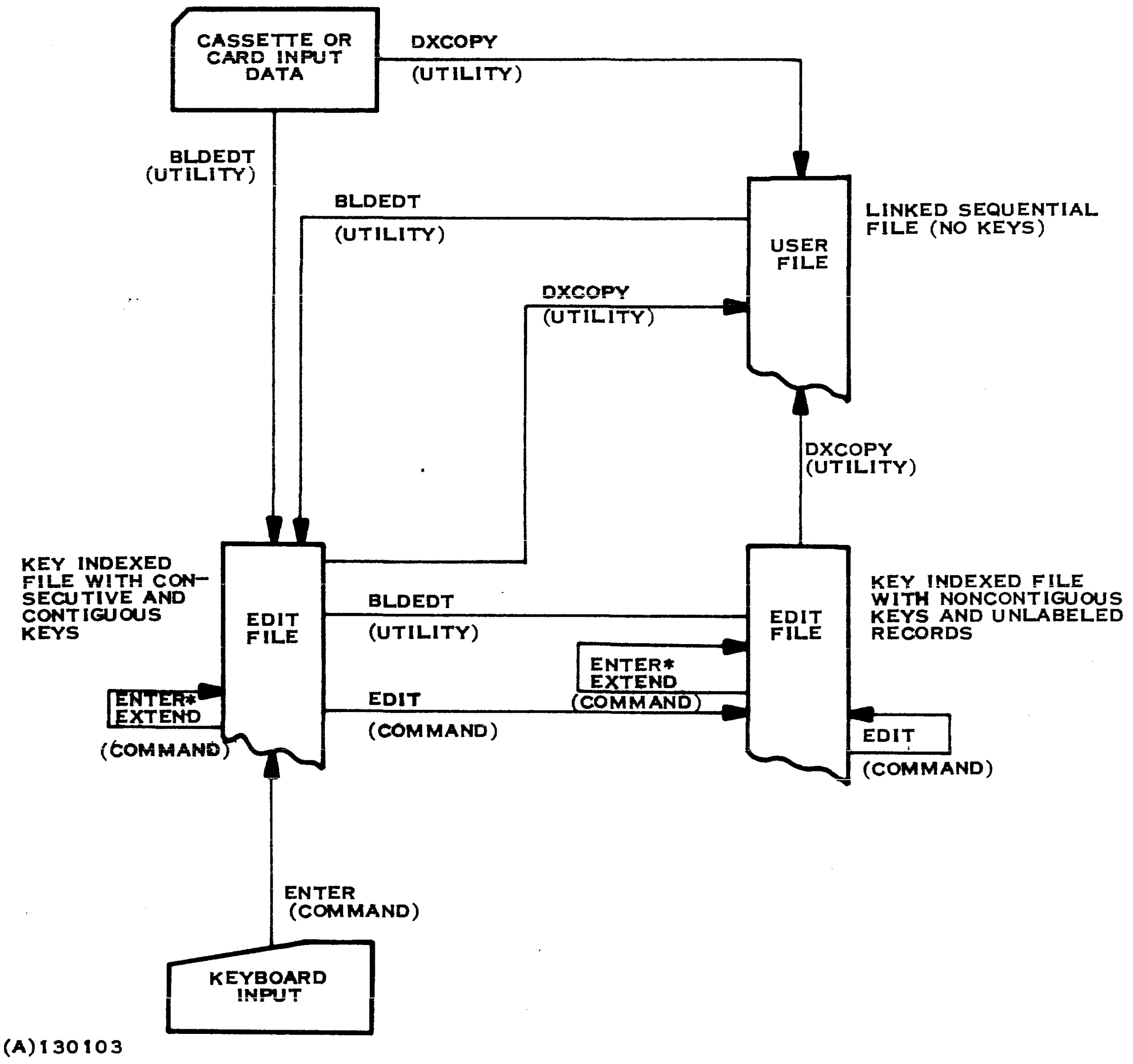


Figure 7-1. Interactive File Editor Transitions

the record numbers and the data is the source text. The first record must have a key of "1", the second record "2", etc.. Either of two methods can create a file in this form. The first method is a utility program name BLDEDT. BLDEDT accepts input from a sequential access source and copies the input data into a key indexed file suitable for use by IFE. The second method is to use IFE to create a suitable edit file. The Enter command causes IFE to create a new key indexed file and accept the new text from the user terminal record by record. Adding EXTEND to the Enter command causes IFE to add new input to an existing edit file. Once a suitable edit file is created, IFE may be used to modify it by inserting, replacing, or deleting specified records. This is accomplished with an Edit command. While editing, the record number keys in the Edit file become nonconsecutive due to deletions and record insertions without keys. The keys (or record numbers) may be cleaned up by recopying the file using the

BLDEDT utility. When editing is complete, the user may employ the DXCOPY utility with the "NOKEYS" option to transcribe the key indexed file into a linked sequential file. However, in most cases the edited material may be left in the key indexed file. The Assembler, the Compiler and other programs accept input from a key indexed file resulting from IFE.

When using IFE the escape key (ESC) stops input, output, or processing and returns control to the terminal.

The IFE command set can be separated into three basic catagories: file commands, edit commands, and state transition commands. Each command is independent of the type of terminal (teleprinter or CRT) that originated the command. Special exceptions are noted as they are encountered in the following command descriptions.

7.2.1 FILE COMMANDS

The IFE File commands assign, create or extend files for use by the IFE.

7.2.1.1 EDIT FILE.

The Edit File command (EDIT FILE) directs IFE to assign an edit file to the requesting terminal for interactive file editing. The file must have been previously created by the Build Edit File (BLDEDT) program or by an Enter File command. The Edit File command is permissable only when the terminal is in the Ready state. The format for the Edit File command is as follows. The interpretation of each parameter is identical to that specified in the JCL description in Section II of this manual. The default value for LRECL is 80 characters.

```
EDIT FILE=(<volume>, <fileid>, <filnam> [,<pswd>]) [LRECL=<lrchar>]
```

After IFE has processed the Edit File command, it converts the terminal to the Edit state and returns the first record(s) in the file to the terminal. At this point, the user may enter edit commands or state transition commands to control further processing.

7.2.1.2 ENTER FILE.

The Enter File command directs IFE to create a new file or extend an existing file using data from the terminal. The file is a key indexed file. If the file is being created by the Enter File command (EXTEND option not specified), then the maximum number of tracks allowed for the file is 25 which accommodates about 1200 lines of text. The access codes of a file created with the Enter File command are (Any, Any, Any, None) unless a password is specified in the command in which case the access codes are (Any, Pswd, Pswd, None). As each record is entered into the file, IFE assigns a record number to it that can be used in subsequent editing sessions. Enter File generally used for manual entry of data as with key-to-disc systems. Operation with this command is identical when using either a teleprinter or CRT as a terminal. Control returns to the terminal after each record is processed and awaits entry of the next record. For this command line 1 of a CRT contains data rather than commands.

The format for the Enter File command is as follows. The <fileid> parameter must have been previously defined via the CATLOG utility from the system console. The <filnam> parameter may be new if the command creates a new file. Default value for LRECL is 80 characters and defines the line-width for the terminal.

```
ENTER FILE=(<volume>, <fileid>, <filnam> [,<pswd>])
  [LRECL=<lrchar>] [EXTEND]
```

The Extend parameter of the Enter File command extends previously created files by adding records to the end. As each record is added, IFE assigns the next consecutive record number to it as with newly created files.

After IFE has processed an Enter command, it returns to the user and awaits entry of data records. The length of each input record, keyed in at the terminal, is controlled by the carriage return (CR) on a teleprinter or number of characters per line on a CRT. The record is padded out with blanks if the record length is less than LRECL, and is truncated if record length is greater than LRECL. A zero length record (CR only) causes IFE to stop accepting records to put in the file, and IFE returns to the Ready state.

7.2.1.3 DELETE FILE. The Delete File command directs IFE to delete a file. The file is deleted if the integrity code which ITS is running under allows the deletion.

7.2.2 EDIT COMMANDS

The basic format for all Edit commands plus the first two operands is as follows:

```
{command} [RN=][±]M [, N]
```

The braces, { }, indicate a mandatory field containing the command or operator. The brackets, [ ], indicate options. The parameter, M, is either an absolute or relative record number depending upon the presence or absence of a preceeding [RN=] field. The absolute record number indicator, [RN=], directs IFE to select record number M from the record numbers that were originally recorded in the edit file. Since the record numbers start at 1 and are all positive, a minus sign for M does not have a general interpretation for absolute record numbers. However, two special cases exist for starting in front of the first record in the file (Beginning of File or BOF), or after the last record (End of File or EOF). The two cases are:

RN=0	Positions file at BOF
RN=-1	Positions file at EOF

Any other absolute record number for M must correspond to a record number that exists in the file. If not, the message "RECORD NOT FOUND" is displayed following the command.

If the [RN=] field is not used, then M specifies a relative record position. That is, it uses the current record position as a point of reference within the edit file to access other records. The current record is the first data line displayed on a CRT or the last data line that was printed on a teleprinter. Specification of relative record position within a command directs IFE to move forward (+) or backward (-) M records. If the sign field is not present, + is assumed. The current record is defined as relative record number 1, the next record is record 2, etc. If the sign field specifies a minus number, -1 refers to the record immediately preceeding the current record, -2 to the record preceeding -1, etc. If 0 is specified as a relative record number, IFE converts it to +1. The one exception to this rule is for the Insert Record command for CRTs. See examples for that command. The parameter [N] specifies the number of records to be processed. Processing occurs after the new file position has been established. If not specified, [N] defaults to 1. The following examples illustrate the use of the M parameter and its associated fields:

• RN=1	Position at record number 1 in the edit file
• RN=-3	Error: An edit file cannot contain a record number of -3
• +1	Position at the current record: On a teleprinter this specifies the last record printed. On a CRT it specifies the first data line displayed (the first data line is actually line 2 on the screen, since line 1 is reserved for commands)
• 1	Same as +1
• -3	Position at the record that is three records before the current record.
• RN=5, 2	Position at record number 5, then process two records.
• 3, 1	Position at the record that is two records beyond the current record, then process one record
• 3	Same as 3, 1

7.2.2.1 FIND RECORD: F[RN=][±]M. This command establishes a new current record and displays it at the terminal. If the terminal is a CRT, it fills the screen with records from the file starting at the new current record.

7.2.2.2 REPLACE RECORD: R[RN=][±]M[,N]. This command establishes a new current record and replaces N records in the file with new records entered from the terminal. If the new replacement record has a pound sign (#) in column 1, it deletes the corresponding file record but does not replace it with a new record. This character allows simultaneous deleting and modifying of replacement records. After the replacement process is complete, the record immediately following the last record replaced becomes the current record.

When this command is entered from a teleprinter, IFE returns control to the keyboard to accept N records (one at a time). Each time a record is entered, IFE deletes the corresponding record from the file and, unless a # is in column 1, replaces the old record with the new replacement record, as entered. If a null length record is transmitted (user enters only a carriage return), then IFE terminates the replacement process and returns control to the keyboard to accept a new command. The Replace command has a slightly different interpretation when entered from a CRT. The records displayed on the screen become the replacement records even if they originally came from the file. This feature allows multiline editing from a CRT. The records displayed on the screen by any previous command can be modified on the screen and then used as replacements for the corresponding records in the file. Since the new replacement records come from the screen image, this command should not be used immediately following a List command which displays information other than just the records from the file.

The absolute record number option, [RN=], is not valid from a CRT.

The following examples illustrate the use of this command:

- R 1,3 — Replace the current record and the two successive records. When entered from a CRT, the replacement records come from data lines 1, 2, and 3; when entered from a teleprinter, control returns to the terminal three times to accept three replacement records. On any type of terminal, a replacement record with # in column 1 deletes the corresponding file record and does not add the replacement record.
- R RN=1,3 — Same as R 1,3 except that the records are the file record with record number 1 and the two successive file records. This command is not valid from a CRT.
- R 5,4 — Replace the file record which is 4 records beyond the current record and the three following records. When entered from a CRT, the replacement records come from data lines 5, 6, 7, and 8; when entered from a teleprinter, control returns to the terminal four times to accept four replacement records.

7.2.2.3 INSERT RECORD: I [RN=][±]M [,N]. This command establishes a new position and inserts N records following the new current record. The insertions can be terminated short of N records from a teleprinter by transmitting a null line. After processing is complete for the Insert Records command, control returns to a teleprinter with the print head in position one and the bell rings. When executed on a CRT, the displayed data opens at the insertion point to make room for inserting records. A # appears in column one of each insertion record and the remainder of the record contains blanks. Insertion of records from a CRT is accomplished with a user entered Replace Record command after the new records have been keyed into the data lines that were opened for insertion. Each # must be eliminated before executing the Replace Record command, or the record will not be inserted. The absolute record number option is not valid from a CRT. The following examples illustrate the use of this command:

• I 1,3	Insert three records after the current record. When entered from a CRT, the current record is maintained on data line 1; data lines 2, 3, and 4 are blanked out except for a # in column 1. The record that was previously displayed on data line 2 is in data line 5, data line 3 moves to data line 6, etc. The user-entered Replace Record command that would normally follow this command is R 2,3. This accomplishes the actual insertion of the records (that the user has keyed into data lines 2,3, and 4) over the top of the lines which IFE displayed with the # in column 1. When entered from a teleprinter, IFE returns control to the terminal three times to accept three insertion records. If the user enters a null line (enters only a carriage return), then the insertion process is terminated and IFE requests a new command.
• I RN=5,3	Insert three records after record number 5. Not valid from a CRT.
• I RN=0,2	Insert two records <u>before</u> the first record in the file. Not valid from a CRT.
• I 0,2	Insert two records after the current record. If entered from a CRT, data lines 1 and 2 are blanked out except for a # in column 1; and if the normal Replace Record command of R 1,2 is then entered, the records actually get inserted <u>before</u> the record that is considered the current record at the time this I 0,2 command is entered. In this way, records may be inserted before the first record in the file.
• I -1,4	Insert four records after the record which precedes the current record. Not valid from a CRT.

NOTE

Since, on a CRT, this command is normally followed by a Replace Record command (which gets replacement records from the lines on the CRT exactly as they are displayed), the Insert Record command should not be used immediately following a List Record command.

7.2.2.4 DELETE RECORD: D[RN=][±]M[,N]. This command finds a new current record and deletes N records starting with the new current record. After the appropriate records are deleted, the current record is then changed to be the record which follows the last deleted record. If records are deleted from a CRT screen, the displayed data closes to reflect the new contents of the edit file. The following examples illustrate the use of this command.

• D 1,2	Delete the current record and the record which follows it. When complete, display the new current record.
• D RN=5,3	Delete record number 5 and the following two records.
• D RN=0,4	Delete the first four records of the file.

7.2.2.5 FIND STRING: FS [RN=][±]M [,N][C1:C2]/STRING/. This command finds a new current record and searches N records for the subject string. The search for the string occurs between record columns C1 and C2. A slash within a string is denoted by a double slash, i.e. //=/. The Find String command is used in the same manner as Find Record except that IFE searches for an occurrence of the specified string within the record limits specified. If C1 and C2 are not specified, IFE searches each record between column 1 and LRECL. The limits C1 and C2 restrict the search to fewer than LRECL characters to eliminate unwanted matches and to decrease the search time. The following examples illustrate the use of this command:

• FS RN=1, 10 /ABC/	Search through ten records starting with record number 1 for the string ABC. Establish the record containing ABC as the new current record. If the string is not found, display "STRING NOT FOUND" message and maintain the original current record.
• FS RN=10, 50 1:10 /XY//Z/	Search through fifty records starting with record number 10 for the string XY/Z. Limit the search to record columns one through ten.

7.2.2.6 REPLACE STRING: RS [RN=][±]M [,N][C1:C2]/STRING1/ /STRING2/. The Replace String command is identical to Find String except that STRING2 replaces the first occurrence of STRING1. If the lengths of STRING1 and STRING2 are different, the insertion is made and the record length is adjusted accordingly. If the adjusted record length is greater than LRECL, the record is truncated to LRECL. If the new record length is shorter than LRECL, the record closes and fills with blanks from the right. The use of consecutive slashes for STRING2 denotes a null string and deletes

STRING1 from the record. The following examples illustrate the use of this command:

• RS 1, 10 1:20 /ABC/ /DEF/	Search through 10 records starting with the current record for the string ABC. If ABC is found, substitute the string DEF, otherwise display the message: "STRING NOT FOUND". Limit the search to columns 1 through 20.
• RS RN=1, 10 /ABC/ //	Search through 10 records starting with record number 1 for the string ABC. If ABC is found, substitute a null string, close up the record and fill with blanks from the right. Otherwise, display the message "STRING NOT FOUND".

7.2.2.7 REPLACE ALL STRINGS: RA [RN=][±]M [,N][C1:C2]/STRING1/ /STRING2/. Replace All Strings is identical to Replace String except that STRING2 replaces every occurrence of STRING1 in the subject records. The current line is relocated to the record which follows the N lines searched.

7.2.2.8 LIST RECORD: L [RN=][±]M[,N]. This command establishes a new current record and lists N records starting at that point. When the listing is complete, the current record remains at the first of the N records listed. The format of the listed records is as follows:

1. If the record has a record number associated with it (the record was not inserted with an Insert Record command since the last time the file was built), then the record number (maximum of four digits) followed by a separator blank and the record are displayed. The record is truncated on the right to fit the display line.

2. If the record has no associated record number, then five blanks and the record are displayed. Again, the record is truncated on the right to fit the display line.

The List Record command is intended for low volume listing and is, therefore, a relatively slow operation. For high volume listing, use the LSTEDT utility.

7.2.3 STATE TRANSITION COMMANDS

The state transition commands permissable in the edit state are Stop and Logoff. Stop returns the terminal to the ready state. Logoff returns the terminal to the inactive state. No state transition commands are permissible from the enter state. The only option available to change states is a null transmission which switches the state from enter to ready.

7.3 REMOTE JOB ENTRY

The Remote Job Entry (RJE) facility in ITS provides interactive access to the job management system of DX980 through a user terminal. RJE accepts two basic commands: Job command and Run command. Both commands are valid from the ready state only.

7.3.1 JOB COMMAND

The Job command is optional when submitting a job via ITS. However, if it is specified, it must immediately precede a Run command. The format of the Job command from ITS is identical to that from any other system source. If the Job command is not supplied, the default values are as follows:

- <jsname> - <userid> from Logon command
- <userid> - <userid> from Logon command
- <devnum> - 1, corresponds to DISC1
- <fileid> - SYSTEM
- <filnam> - SJCBFL, the system JCL file
- <pswd> - none

7.3.2 RUN COMMAND

The Run command is required to start a job from ITS. The format of the Run command is identical to that from any other source. One additional option available to ITS users is the assignment of the user's terminal to the job being started. This can be very useful if the terminal is located at a remote site. The user can assign the terminal through a device assignment of TERMIO. Expanded JCL sequences allow changing device assignments through the use of keywords. Any device keyword can be assigned to the psuedo-physical device name, TERMIO. ITS substitutes the true device name for TERMIO before submitting the job. For example the expanded JCL may specify: DEVICE:=INDEV. In the Run command the user may state: INDEV=TERMIO. In this case ITS substitutes the true device name for TERMIO. That device name may be CRT3 or any other device previously assigned to TERMIO. The pseudo-physical device name, TERMIO, must be specified in the Run command. It may not be specified in the expanded JCL. As many device names as desired can be assigned to TERMIO and each name is converted to the subject terminal. When the job is started, ITS releases the terminal until the job is complete. At that time ITS reacquires the terminal and activates it for further communication. The following example illustrates the device assignment option of the Run command:

```
RUN XA SRCDEV=DISC1, SRCFIL=(USER01, MYPROG);
LSTDEV=TERMIO, ERRDEV=TERMIO, OBJDEV=PTP1
```

Run the assembler with input from a file with file name "MYPROG" (possibly created and edited using IFE). Assign listings and error messages to my terminal (TERMIO) and assign the object output to the paper tape punch.

7.4 COMPUTER STATUS DISPLAY

The Computer Status Display facility in ITS determines system status either before or after submitting a job through the RJE facility. Entering the following command at the terminal displays the status of all programs in the system:

```
STATUS
```

The statistics for all steps within a specific job can be displayed by entering the command in the following form:

```
STATUS <jobnum>
```

7.5 ITS ERROR MESSAGES.

ITS error messages fall generally into three catagories.

A. Messages of the form:

ERROR NNNN where NNNN is a four digit error number. These error messages are given only by the Remote Job Entry and Computer Status Display facilities. The error numbers are passed to ITS by DX980 and are documented in Appendix A.

B. The message:

INSUFFICIENT MEMORY, TRY AGAIN LATER

This message is given only by the Remote Job Entry and Computer Status Display facilities and indicates that there is not enough available memory in the ITS area to perform the request. If several terminals are currently using ITS, then more memory may be available later as operations for other terminals are completed.

C. IFE Error Messages:

(1) .INVALID COMMAND
(2) .DISC OFFLINE
(3) .INVALID FILE TYPE
(4) .INVALID ARGUMENT
(5) .ARG COUNT ERROR
(6) .ARG RANGE ERROR
(7) .RECORD NOT FOUND
(8) .STRING NOT FOUND
(9) .STRING TOO LONG
(10) .INSUFFICIENT MEMORY
(11) .FILE ASSIGN ERROR
(12) .FILE BUSY
(13) .FILE FULL
(14) .FILE READ ERROR
(15) .FILE OPEN ERROR
(16) .FILE I/O ERROR

In most cases the user can readily discover the problem causing the error and then reenter the command correctly to continue. However, messages (15) and (16) may be given in response to the user submission of the EDIT or ENTER commands and if this occurs the user must enter STOP or LOGOFF before re-submitting the desired EDIT or ENTER command.

SECTION VIII
UTILITY PROGRAMS

8.1 GENERAL

A comprehensive set of programming support language processors and utility programs is available for use with the operating system. Sequences of expanded JCL supplied with a system reside on the system disc in the file addressed as FILE=(1, SYSTEM, JCLSRC). Appendix B contains a sample listing of some expanded JCL sequences. When specified in a Run command, these standard JCL sequences perform the language processors and utilities. Some of these programs, such as the FORTRAN compiler, are not unique to the DX980 operating system. These programs are documented in separate manuals. This section provides information concerning the use of these programs as they apply to the DX980 operating system.

8.2 JCL TRANSLATOR (JCLTRN)

The JCL translator is a utility program that runs in user memory (as opposed to running in memory that is allocated to the operating system). This program translates expanded JCL statements into an internal binary representation for later processing. Input to the translator can be supplied from any of the standard 980 input devices (card reader, cassette, magnetic tape, disc files, etc.). Output from the translator is stored on a disc file for later submission. Since the translator is a utility program, it can be invoked as any other user program by using abbreviated JCL form. The saved JCL sequence for the translator is stored on the system disc together with several other sequences that are supplied with each DX980 installation. The name assigned to the sequence for the translator is JCL.

8.2.1 STANDARD JCL PROCEDURE

The following JCL is a standard procedure for invoking the JCL translator (JCLTRN).

```
.# CREATE JCL    ,COMMENT,"CREATE JCL PROCEDURE          "
/REPLACE JCL          . CREATE JCL PROCEDURE .
/EXEC OBJ=(1,SYSTEM,JCLTRN) MEM=(300,7550,1000) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI
/ASSIGN  1 SC    DEVICE:=DSRC FILE:=FSRC BUFFERS=1             , SOURCE INPUT
/ASSIGN  2 SC    DEVICE:=DERR SHARE:=SERR                      . ERROR MESSAGE
/ASSIGN  3 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.SOURCE LISTING
/ASSIGN  4 DISC; DEVICE:=DOBJ FILE=(SYSTEM,SJCBFL,AB);
/          FILE:=FOBJ REPLACE:=ROBJ BUFFERS=2 INDEXED;
/          ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=COBJ;
/          ALLOCATE=(1,0,96,20) ALLOCATE:=LOBJ KEYLEN=6        . OBJECT OUT FILE
/END
```

8.2.2 MEMORY PARTITION REQUIREMENTS

The memory allocation parameters for the JCL translator are:

MEM=(300, 7550, 650)

8.2.3 LUN ASSIGNMENTS

The LUN assignments required by the JCL translator are listed in table 8-1.

Table 8-1. JCLTRN Logical Unit Assignments

LUN	Description	Comments
1	Expanded JCL input	Any sequential access input device file
2	Error Message	Normally assigned to DUMMY when input is from a Data Terminal; otherwise assigned to SYSOUT or LP1.
3	JCL echo print	
4	Binary Format JCL output	Key indexed file with key length of 6 characters and a physical record length of 96 words.

8.2.4 OPERATION

Refer to Section II of this manual for a description of the JCL translator.

8.2.5 ERROR CODES

Refer to Appendix A of this manual for JCL error codes.

8.2.6 SAMPLE INPUT

Refer to Section II and Appendix B of this manual for sample input sequences.

8.3 MASTER FILE DIRECTORY EDITOR (CATLOG)

The CATLOG utility allows the operator to list, create, or delete users in the Master File Directory of any specified disc volume.

8.3.1 STANDARD JCL PROCEDURE

The following JCL is a standard procedure for the Master File Directory Editor, CATLOG:

```
.# CREATE CATLOG,COMMENT,"LIST, CREATE,OR DELETE USERS"
/REPLACE CATLOG      . LIST, CREATE, OR DELETE USERS .
/EXEC OBJ=(1,SYSTEM,CATLOG) MEM=(300,4000,670) PRTY=(1,5);
/          PRIV TIME=-1 PRTY:=PRI
/ASSIGN  0 SC    DEVICE:=DCON SHARE:=SCON                    . CONTROL
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.USER LISTING
/END
```

8.3.2 MEMORY PARTITION REQUIREMENTS

The job area required to run CATLOG must only be large enough to contain the load module since CATLOG requires no work space. Sufficient job area is 4000 words; sufficient job extension area is 670 words.

8.3.3 LUN ASSIGNMENTS

The logical unit assignments needed for CATLOG are listed in table 8-2.

Table 8-2. CATLOG Logical Unit Assignments

LUN	Description	Comments
0	Message Control	Any input/output device
6	Listing of Users	Any printing device

8.3.4 OPERATION

The Master File Directory Editor, CATLOG, is a single task program that runs under DX980 in the privileged mode. Since it accesses portions of the system which are inaccessible to the user, the CATLOG modules must be linked with the system external definition file (SYSTEM, DXEXTD).

The program, CATLOG, should be run from the system console only. It first types the message:

ENTER COMMAND TYPE--LIST, CREATE, DELETE--

on the message and control device (LUN 0) which is usually assigned to the system console. The operator then responds by entering the first letter of any of the three commands (i.e. L, C or D), followed by a carriage return to select a list, create, or delete user operation, respectively. CATLOG then requests the disc volume identification by printing to LUN 0:

DISC VOLUME ID =

The user responds by entering a number between 1 and 20 followed by a carriage return. This number selects the disc pack contained in the disc drive with that number. The default for this input parameter is disc drive 1 and can be invoked by entering carriage return without selecting a number.

Beyond this point, CATLOG's response is dependent on the function (list, create, or delete) selected. Table 8-3 summarizes the input/output of CATLOG for the three functions. The three functions are described separately in the next three paragraphs.

8.3.4.1 LIST. If the user responds with an L input to select the list users function, the utility prints on LUN 6 an alphabetically ordered list of all users on the disc volume selected. The utility also prints the date and time of request.

CATLOG provides the following information for each user:

- User ID - the name of the user associated with the directory.
- Integrity code - A three-position code that defines allowed operation to the user directory for three separate operations. The output format is X, X, X, for read, write, delete operations respectively. The letter X represents either N, C, or A (NONE, CREAT, ANY, the file level integrity codes used as input to the JCL translator/ interpreter). Table 8-4 provides an interpretation of the three integrity codes as a function of operation at the user level.
- Current user directory size - the number of tracks currently assigned to the user directory.
- Maximum user directory size - the maximum number of tracks that can be assigned to the user directory.

In addition to the above tabular data for each user on the disc volume, CATLOG prints the following information for the entire disc volume.

- Number of users
- Number of tracks available
- Number of tracks used

Table 8-3. CATLOG User Interaction

CONSOLE OUTPUT	User Response		
	LIST	CREATE	DELETE
ENTER COMMAND TYPE--LIST, CREATE, DELETE--	L	C	D
DISC VOLUME ID =	any disc drive number 1 through 20 (default = 1)		
USER ID =		A 1 to 6 character alphanumeric name of the user ID. (no default)	
INTEGRITY CODE =		4 alphabetic characters, separated by commas, designate the access codes for read, write, delete and execute accesses (default = A, A, A, A)*	
MAXIMUM DIRECTORY SIZE =		Number of tracks to designate maximum size of user directory (default = 1)	

*N = None
C = Creator
P = Password
A = Any

Table 8-4. User ID Integrity Codes

Operation			
Integrity Code	Read	Write	Delete
N (NONE)	N, X, X no one can access the files under the user ID	X, N, X no one can define new files under the user ID	X, X, N no one can delete or replace files under the user ID
C (CREAT)	C, C, C ▷1 only the creator can access the files under the user ID	X, C, X ▷1 only the creater can define new files under the user ID	X, X, C ▷1 only the creator can delete or replace files under the user ID
A (ANY)	A, X, X any one can access the files under the user ID	X, A, X any one can define new files under the user ID	X, X, A any one can delete or replace files under the user ID

▷1 Creator - that user whose < userid > parameter in the JOB command matches the <userid> of the directory.

8.3.4.2 CREATE. If the user responds with a C input to select the create user function, the utility requests the following additional information (in addition to the disc volume ID):

- User ID - A 1 to 6 character alphanumeric name of the user ID to be created, followed by a carriage return.
- Integrity code - A 3-position code that defines allowed operation to the user directory for three separate operations. The input format is X, X, X for read, write, delete operations, respectively. The letter X is N, C, or A. See table 8-4 for an interpretation of the three integrity codes as a function of operation at the user level. The default for this input parameter is A, A, A for unlimited user operations.

- Maximum user size - A decimal number that designates the maximum number of tracks to be assigned to the user directory. The default is 1 track.

The program then creates the designated user by entering a user file directory (UFD) keyed on the user ID into the master file directory.

8.3.4.3 DELETE. If the user responds with a D input to select the delete user function, CATLOG checks for other jobs running in the system before continuing. If other jobs are running, an error message is printed to LUN 0 and the CATLOG job is terminated. If the job does not terminate at this point, the utility requests (in addition to the disc volume ID) only the user ID to be deleted. The utility then deletes the user file directory (UFD) from the master file directory as well as all files created under that user ID. The CATLOG delete function should not be used to delete individual files under a user, but only to delete an entire user directory, including all files residing within that directory. The delete function of CATLOG may be used only when no other jobs are in the system.

8.3.5 ERROR CODES

Table 8-5 lists the possible error/termination message printed by CATLOG to LUN 0. Those that are possible only when a specific function is requested are indicated accordingly in the table.

Table 8-5. CATLOG Error Messages

Message	Current Function	Meaning
NORMAL TERMINATION		Process performed correctly no errors
ILLEGAL COMMAND		First character of Command input not an L, C, or D.
TOO MANY JOBS IN THE SYSTEM	D	Delete command-cannot be run if there are other jobs running under DX980.
ILLEGAL INTEGRITY CODE	C	Create command-user integrity code is not of format X, X, X where X is A, C, N.
BAD DATA		Numerical input not a valid decimal integer.

Table 8-5. CATLOG Error Messages (Continued)

Message	Current Function	Meaning
ILLEGAL DEVICE ID		Device ID not in the Physical Device Table chain.
DISC OFFLINE		Disc is not enabled.
PREV. DEFINED USER ID	C	Create command-user ID already defined in the Master File Directory.
ILLEGAL USER ID	D	Delete command-user ID not found in the Master File Directory.
USER ID MISSING	D	Delete command-user directory file control block is not in the file.

8.4 LIST USER FILE DIRECTORY (CATFIL)

The CATFIL utility lists all files in the User File Directory for a specified <userid>. It also tabulates all pertinent information for each file.

8.4.1 STANDARD JCL PROCEDURE

The following JCL is a standard procedure for List User File Directory CATFIL:

```
.* CREATE CATFIL,COMMENT,"LIST FILES UNDER A USER     "
/REPLACE CATFIL      . LIST FILES UNDER A USER .
/EXEC OBJ=(1,SYSTEM,CATFIL) MEM=(300,1850,670) PRTY=(1,5);
/           PRIV TIME=-1 PRTY:=PRI
/ASSIGN  0 SC    DEVICE:=DCON SHARE:=SCON                 . CONTROL
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.FILE LISTING
/END
```

8.4.2 MEMORY PARTITION REQUIREMENTS

The job area required to run CATFIL must only be large enough to contain the load module since CATFIL requires no work space. Sufficient job area is 1850 words; sufficient job extension area is 670 words.

8.4.3 LUN ASSIGNMENTS

The logical unit assignments needed for CATFIL are listed in table 8-6.

Table 8-6. CATFIL Logical Unit Assignments

LUN	Description	Comments
0	Message and control	Any input/output device
6	Listing of files	Any printing device or a file

8.4.4 OPERATION

CATFIL is a single task program that runs under DX980 in the privileged mode. Since it accesses portions of the system that are user inaccessible, it is necessary to link the CATFIL modules with the system external definition file (SYSTEM, DXEXTD).

The program, CATFIL, may be run from the system console or from the Batch Processing or Interactive Terminal Subsystems. When run from other than the system console the printout of the password for each file is suppressed. CATFIL types the following request on the message and control device (LUN 0):

```
USERID =
```

The user responds with a 1 to 6 character identifier for the <userid> of the User File Directory to be listed. The utility then requests the volume number (i.e. disc drive number) of the disc containing the User File Directory:

```
DISC VOLUME ID =
```

The user may respond by entering a specific integer value from 1 to 20 followed by a carriage return, or by invoking the default volume number of 1 by selecting carriage return only. The utility then prints the date and time of request, and an alphabetically ordered list of all files within the selected User File Directory on LUN 6. The information provided by CATFIL for each file is as follows:

- File Name - 1 to 6 alphanumeric characters identifying the file.
- Disc Address - hexadecimal track address of initial track allocated to the file.
- Password - 0 to 4 alphanumeric characters identifying the password for the file. If CATFIL has been invoked from other than the system console any passwords for files will be replaced with asterisks.

- Integrity code - four alphabetic file access codes in the form X_1, X_2, X_3, X_4 where

  X_1 = read access code

  X_2 = write access code

  X_3 = delete access code

  X_4 = execute access code

  In general X_i can be:

  N = No one can access the file.

  C = The creator can access the file.

  P = The user having the correct password can access the file.

  A = Anyone can access the file.

- Current File Size = Number of tracks currently allocated to the file. If the file is completely full, this current file size exceeds the maximum file size by 1 track for linked sequential and key indexed files only.
- Maximum File Size - Maximum number of tracks that may be allocated to the file. One additional track may be allocated to a file at the time the file becomes full.
- File Type - Indicates the type of file from the three supported types; linked sequential, relative record, and key indexed.
- Logical Record/Key Length - Number of characters in the fixed length logical record for relative record files. Number of characters in the key for key indexed files.
- Physical Record Length - Number of words in the physical record for any type file. This establishes the length of the system accessible buffers used for input/output to the file.

In addition to this tabular data for each file within the User File Directory, CATFIL prints the following information for the specified User ID:

- The number of files
- The number of tracks allocated

8.4.5 ERROR CODES

The possible error/termination messages printed by CATFIL to LUN 0 are listed in table 8-7.

Table 8-7. CATFIL Termination Messages

Message	Definition
NORMAL TERMINATION	Process executed correctly
INVALID DEVICE ID	Device ID is not in the Physical Device Table List
ILLEGAL USER ID	On INPUT from the CONSOLE-User ID is greater than 6 characters OR first character is not a letter
DISC OFF LINE	
BAD NUM	Input from the console - not a valid decimal integer
UNDEFINED USER ID	User ID not found in the Master File Directory

8.5 DX980 OVERLAY LINK EDITOR (DXOLE)

The DX980 Overlay Link Editor (DXOLE) must be used to format input object records into a load module for execution under DX980. Three program structures are allowed:

1. A memory resident program with no overlays
2. An overlay structure defined at link edit time (preplanned overlay)
3. An overlay structure that is dynamic and can change during execution (unplanned overlay).

DXOLE functions in three different modes to allow maximum flexibility to the user. These modes are:

- Compact Mode - The Compact mode combines object modules to create new object modules. All references for defined symbols between the object modules are resolved. All references to entry points not found in the object modules are left unresolved. The object module produced is relinkable. The new object module defines all the entry points defined in the linked modules as entry points and contains references to the unresolved entry points.
- Normal Mode - The Normal mode combines one or more object modules to create a load module for execution. This mode is the default mode for the link editor. The load module created may or may not contain an overlay structure, depending upon parameters supplied by the user. The Normal mode can produce an External Definition File. This file contains the symbol names and addresses of the entry points defined in the root segment of the created load module.

- Subsystem Mode - The Subsystem mode combines one or more object modules to create a load module containing a subsystem for execution. This load module may contain a preplanned overlay, but cannot contain an unplanned overlay. Subsystems execute as absolute programs. Therefore, they can address into the root segment of the operating system. In the Subsystem mode an External Definition File produced during a previous Normal mode link of the operating system resolves external references.

8.5.1 STANDARD JCL PROCEDURE

The following JCL listing is a standard procedure for DXOLE:

```
/REPLACE DXOLE      . LINK EDITOR .
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(300,12000,2000) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  1 DUMMY DEVICE:=DOB1 FILE:=FOB1 BUFFERS=2              . ALT 1 OBJECT IN
/ASSIGN  2 DUMMY DEVICE:=DOB2 FILE:=FOB2 BUFFERS=2              , ALT 2 OBJECT IN
/ASSIGN  5 DISC1 DEVICE:=DIN  FILE:=FIN  BUFFERS=1              , PRIMARY INPUT/CON
/ASSIGN  6 SC    DEVICE:=DLST                                   , LOADMAP LIST/ERR
/ASSIGN  7 DUMMY DEVICE:=DOBJ FILE:=FOBJ BUFFERS=1              . COMPACT OBJ OUT
/ASSIGN  8 DISC1 DEVICE:=DLM  FILE:=FLM  REPLACE:=RLM;
/          BUFFERS=1 RELREC ACCESS=(ANY,CREAT,CREAT,ANY);
/          ACCESS:=CLM ALLOCATE=(1,0,32,1) ALLOCATE:=LLM;
/          LRECL=64                                             . LOAD MOD OUTPUT
/ASSIGN  9 DISC1 FILE=(SYSTEM,USRFTN) DEVICE:=DLIB;
/          FILE:=FLIB BUFFERS=2                                 . LIBRARY FILE
/ASSIGN 10 DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,256,30)    , LINKSEQ SCRATCH
/ASSIGN 11 DISC1 FILE=(SYSTEM,DXEXTD) FILE:=FEXT BUFFERS=1      . SYS EXT DEFS OPT
/ASSIGN 13 DISC1 FILE=(TEMP,SCRR) NEW BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10);
/          LRECL=100                                            . RELREC SCRATCH
/END
```

8.5.2 MEMORY PARTITION REQUIREMENTS

DXOLE memory requirements for linking an average program are defined with the following expression:

MEM=(300, 12000, 2000)

DXOLE is an overlay program with variable memory requirements.

DXOLE requires 7950 words of program space plus a 2000 word Job Extension Area (<jearea>). A variable amount of table space is required depending upon the number of references, definitions, overlays, and identifications in the input object decks. The memory requirements do not depend upon the size of the load module being produced. The following guidelines aid in determining the size of the variable tables:

- 17 words per overlay to be generated
- 12 words per DEF
- 12 words for the first REF and 3 words for each subsequent REF to a label

- 9 words per common definition
- 3 words per common reference
- 13 words per object module
- 5 words per OBJNAM parameter (unplanned overlays)
- 3 words per Library LUN specified

8.5.3 LUN ASSIGNMENTS

Table 8-8 provides a summary of the LUN assignment requirements for DXOLE as a function of mode. As can be seen in the table, not all assignments are required for each mode. In addition to these standard LUN's any other LUN's can be assigned to DXOLE and accessed by means of Include, Search, and Library commands.

8.5.4 OPERATION

DXOLE allows two different overlay structures, preplanned and unplanned. Each structure has advantages and the user should decide which, if any, overlay technique to use. The control cards describe the program structure to DXOLE.

8.5.4.1 PREPLANNED OVERLAYS. A program that is link edited in the Normal mode or Subsystem mode with the overlay (OVLY) option is called a preplanned overlay. The programmer describes the overlay structures for the program with the Root and Segment control cards. During execution of a preplanned overlay structure, a Runtime Overlay Manager (OLM$#$) loads all overlays into memory. The link editor includes the overlay manager as part of the root segment. The Runtime Overlay Manager determines the relationships between overlay segments and the load addresses of the overlays by inspecting overlay transfer vectors built by the link editor. These runtime vectors, called the Overlay Segment Vector (OSV$#$) and Overlay Transfer Vector (OTV$#$), are also included in the root segment of the load module produced.

The link editor uses the call-by-name technique for loading overlays in a preplanned overlay structure. The user program references the name of the definition resolved in the next level of the overlay. The overlay manager then loads the overlay and transfers control to the desired entry point. The overlay is only loaded when necessary. Following calls to the same overlay will not reload the overlay unless it has been replaced in memory by another overlay at the same overlay level. The overlay manager uses the S and X registers and does not preserve their values. The calling program must save these registers if the contents should not be destroyed. On the preplanned overlay load map all forward references in the overlay are identified by the message "**OTV**". The address associated with the reference on

Table 8-8. DXOLE Logical Unit Assignment

LUN	Description	Comments	Mode Requirement		
			Compact	Normal	Subsystem
5	Primary input	Sequential input device/file with 80 character record length. Input control commands and object modules may be intermixed.	X	X	X
6	Diagnostics and Memory Maps	Sequential output device/file with 80 character record length.	X	X	X
7	Object Module Output	Sequential output device/file with 80 character record length.	X		
8	Load Module Output File	Relative record file with 64 character record length.		X	X
9	Subroutine Library File	Dummy or key indexed file with 80 character record length and 8 character key length. This file should be built using the Library Builder (LIBBLD) utility.	X	X	X
10	Intermediate Scratch File	Rewindable device or linked sequential file with 80 character record length.	X	X	X
11	External Definition Input File	Linked sequential file with 8 character record length. This file contains the external definition table for the currently running versions of the operating system produced under the Normal mode during a previous link edit of the operating system.	X		X
12	External Definition Output File	Linked sequential file with 8 character record length.		X [1]	
13	Intermediate Scratch File	Relative record file with 100 character record length.	X	X	X
any unused	Alternate Input	Accessed with Include control command	X [2]	X [2]	X [2]
any unused	Alternate Library File	Accessed with Library control command	X [2]	X [2]	X [2]

NOTES:

1 The External Definition Output File is not required for any mode, but is optional for the Normal mode. It is assigned only when an external definition table is to be produced.

2 Optional alternate LUN.

the load map points to a table entry that the Overlay Manager will use to load the overlay. External references in the overlays are resolved in the following manner.

1) A reference to a label defined in the same overlay is resolved as the address of the label.

2) A reference to a label defined in a lower level overlay which is in the path of this overlay is resolved as the address of the label.

3) A reference to a label which is defined in the next overlay level is resolved as an address in the Overlay Transfer Vector. At runtime the entries in the Overlay Transfer Vector are managed by the Overlay Manager. When an overlay is currently in memory, the Overlay Transfer Vector causes control to pass directly to the overlay. When an overlay is not in memory, the Overlay Transfer Vector causes control to transfer to the Overlay Manager to load the overlay and then transfer control to the overlay. Forward data references may not be used.

4) No forward references can be made to definitions more than one overlay level away. Any such reference is marked as undefined ("**UN-DEF**" on the load map) and is assigned the absolute value FFFF_{16}.

8.5.4.2 UNPLANNED OVERLAYS. An unplanned overlay is a program that is link edited in the Normal mode with the No Overlay (NOVLY) option and that defines overlays with Object control cards. The programmer describes the overlay structure with the Root and Object control cards. The overlay is called an unplanned overlay since the location of the overlay in memory is not specified when the link edit is performed. During execution the overlay is loaded using the load and relocate SVC. DXOLE does not include a Runtime Overlay Manager as part of the load module. The second parameter of the Object control card defines a name. This name can be an external reference in the root or any overlay, and is resolved as the overlay number. The link editor does not resolve any other references between overlays. The order for resolving external references is:

1) A reference to a label defined in the same overlay is resolved as the address of the label.

2) A reference in an overlay to a label defined in the root is resolved as the address of the label.

3) A reference having the same name as the second parameter of the Object control card is resolved as the overlay number. These items are identified on the load map by the message, **MIP**.

4) No references can be made to definitions in any overlay. Any such reference is unresolved and is flagged as "**UN-DEF**" on the load map. Hexadecimal FFFF is assigned as the value of the unresolved reference.

8.5.4.3 CONTROL CARDS. Figure 8-1 illustrates the format of the three DXOLE control cards (Type I, Type II and Type III). Control cards must have a blank in column one, a card name starting in column two and one or more blanks preceding the first argument. Commas separate each argument. The argument list cannot contain imbedded blanks. Arguments may be specified up to and including column 72. Comments can be placed on the control cards following the first blank after the last argument specified. These control cards relate three kinds of information to the link editor:

- The mode of the link edit to produce either a relinkable object module or a load module.
- The relationship between the memory image phases in the load module.
- The input LUNs for the object modules to be included with each Type II control card.

Table 8-9 indicates the Type II and Type III cards that are valid for each Type I control card and its options. Figure 8-2 shows a sample control deck and a block diagram of the associated program.

TYPE I:

a) COMPACT $\left[\begin{array}{l}\text{PAGE ①}\\ \hline \text{NO PAGE}\end{array}\right]$

b) NORMAL $\left[\begin{array}{l}\text{NEXTD ①}\\ \hline \text{EXTD}\end{array}\right]$ $\left[\begin{array}{l}\text{NOVLY ①}\\ \hline \text{, OVLY}\end{array}\right]$ $\left[\text{, MAXENT=}n_1\ ②\right]$ $\left[\text{, MAXSEG=}n_2\ ③\right]$ $\left[\begin{array}{l}\text{PAGE ①}\\ \hline \text{NO PAGE}\end{array}\right]$

c) SUBSYSTEM $\left[\begin{array}{l}\text{NOVLY ①}\\ \hline \text{OVLY}\end{array}\right]$ $\left[\text{, MAXENT=}n_1\ ②\right]$ $\left[\text{, MAXSEG=}n_2\ ③\right]$ $\left[\begin{array}{l}\text{PAGE ①}\\ \hline \text{NO PAGE}\end{array}\right]$

TYPE II:

OBJECT	IDTNAM, OBJNAM	[, HEXCON]
ROOT	IDTNAM	
SEGMENT	LEVEL	

TYPE III:

SEARCH	1,2(Key1,Key2,...,Key$_n$),3
INCLUDE	1,2(Key1,Key2,...,Key$_n$),3
LIBRARY	20,21,22

① Default option

② Default = 44

③ Default = 22

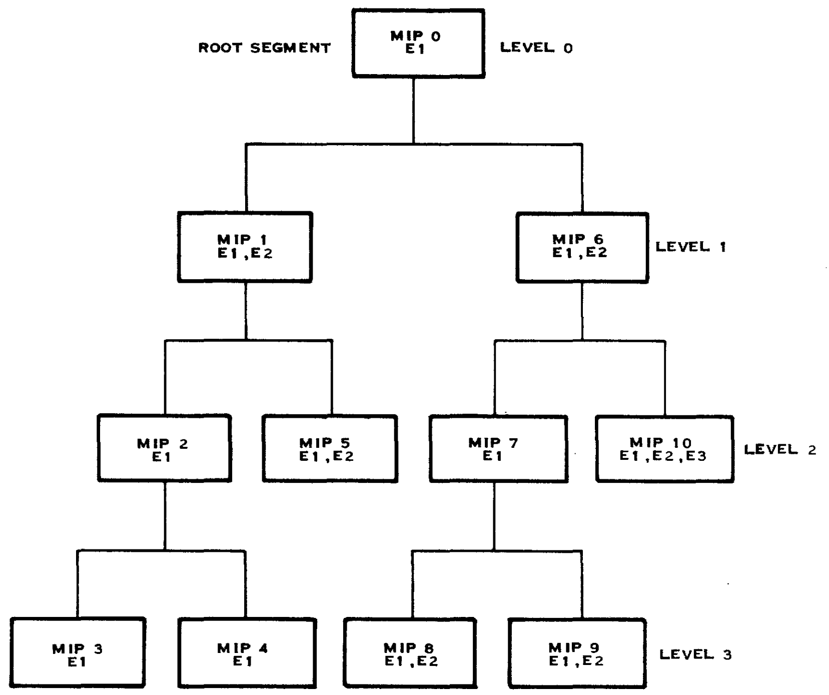
Figure 8-1. Control Card Formats

Table 8-9. DXOLE Control Card Sequence and Options

Type I Control Card	Valid Type II Cards			Valid Type III Cards			
	Object	Root	Segment	Object Deck	Search	Include	Library
Compact	X [1]			X	X	X	X
Normal No Overlay	X	X		X	X	X	X
Normal With Overlay		X [1]	X	X	X	X	X
Subsystem No Overlay		X [1]		X	X	X	X
Subsystem With Overlay		X [1]	X	X	X	X	X

Note:

1. Required card



```
CARD
TYPE

  I    NORMAL OVLY,
       MAXSEG=10
 II    ROOT IDT1
      {'1700' IDT1
III   {   ~              OBJECT DECK
      {'1706'
       INCLUDE 16(S0)
       LIBRARY 17,18
 II    SEGMENT 1
III    INCLUDE 16(S1)
 II    SEGMENT 2
      {1700 IDTX
III   {   ~
      {1706
       SEARCH 18(KEY1)
 II    SEGMENT 3
III    INCLUDE 16(S3)
 II    SEGMENT 3
      {1700
III   {   ~
      {1706
       LIBRARY 17
 II    SEGMENT 2
III    INCLUDE 16(S5,S4),17(S$5)
 II    SEGMENT 1
      {1700
III   {   ~
      {1706
 II    SEGMENT 2
III    INCLUDE 18(S7)
 II    SEGMENT 3
III    INCLUDE 16(S8)
 II    SEGMENT 3
III    INCLUDE 9 (QQCOSINE)
       SEARCH 17
 II    SEGMENT 2
      {1700
III   {   ~
      {1706

       /*
```

(A)130110

Figure 8-2. Overlay Structure and Control Deck

Type I Control Cards. A Type I control card is optional. If present it must be the first control card. The Compact card directs DXOLE to link object decks together to create new relinkable object decks. The only valid parameter for this mode is to specify the load map use page ejects or double spacing (PAGE and NOPAGE, respectively). This Normal control card directs DXOLE to create a load module for execution. If no Type I card is specified, the Normal mode is the default Type I control card. Four optional parameters may be specified on this card. The NEXTD/EXTD parameter specifies production (EXTD) or no production (NEXTD) of an External Definition File. The External Definition File is a sequential file that contains the names and addresses of all the definition statements in the root segment of the object to be linked. This file then links subsystems to the operating system root segment. The NOVLY/OVLY parameter specifies production (OVLY) or no production (NOVLY) of a preplanned overlay structure. If this parameter is not present no overlay will be produced. The MAXENT and MAXSEG parameters specify the maximum numbers of forward references in the overlay structure and the maximum number of overlays, respectively. The Subsystem control card also directs DXOLE to create a load module for execution. The arguments valid for Subsystem cards have the same meaning as for the Normal card. The Normal and the Subsystem modes handle the External Definition File differently. The Subsystem mode automatically references the EXTD and creates definition entries for all the names in the EXTD. This allows privileged programs to use subroutines that are part of the root of the operating system.

Type II Control Cards. A Type II control card is required unless the program to be linked is a nonoverlaid and nonprivileged program. For this exception, DXOLE will take the program entry point from the first object deck in the root segment. This default condition allows the control input of DXOLE (LUN 5) to be assigned directly to a linked sequential file output from the 980 Assembler or FORTRAN compiler. For all other DXOLE modes at least one control card is required. In the Compact mode an Object card is required. In the Normal and Subsystem modes a Root card is required. Segment cards can also be specified for the OVLY option and Object cards for the NOVLY option.

A special comment is necessary to explain the specification of a FORTRAN main program as the entry point in the DXOLE output. DXOLE uses blanks as delimiters between the control card type (i.e., Normal, Root, Include) and the arguments. This convention makes the specification of 'ƀMAINƀ' (The FORTRAN main program IDT) impossible. Therefore, DXOLE handles the name 'MAIN' as a reserved word. The IDT specification of 'MAIN' on either the Root card or the Object card as the object deck with the entry point is interpreted by DXOLE to be any one of the following names:

1. ƀMAINƀ - The FORTRAN main program name.
2. ****** - The default IDT name assigned to assembler output decks
3. MAIN. - The default IDT name assigned to an object deck by the PL/EXUS compiler.

In the Compact mode the Object card has two arguments. The first argument is IDTNAM. This argument is the name of the object deck which has an end vector to be used to define the control entry point of the linked object deck. The second parameter is OBJNAM. This argument is the name to assign to the new linked object deck. No other arguments are valid.

Multiple compact mode object decks can be output by entering multiple object control cards. The object decks are output to LUN 7 and terminated with a single end of file.

In the Normal and Subsystem modes the Root card defines the memory resident code. Segment and Object cards define preplanned and unplanned overlays, respectively. The Root card has one argument, IDTNAM. This argument specifies the name of the object deck which has an end vector to be used to define the control entry point in the root segment of the load module. The Segment card has 1 argument, the level number. This argument defines the overlay's relationship to the other overlays and the root. The root is at level 0 of the overlay. The first level below the root is level 1, the next logical level is level 2 and so forth. Level numbers must be sequential. An Object card also defines an overlay. However, all Object overlays are at level 1. The Object card is valid only for the unplanned overlays (NOVLY option). The first parameter of the card, IDTNAM, identifies the object deck which has an end vector to be used to define the control entry point for this overlay. The second required parameter, OBJNAM, associates a name with the overlay being created. This parameter is entered in the symbol table of DXOLE with a value equal to the overlay number (i.e. 1 for the first, 2 for second, etc.). This entry name may be referenced from any other segment and is resolved by DXOLE as the overlay number. The third parameter, HEXCON, is optional. If specified, this parameter indicates the load address of the newly created overlay.

Type III Control Cards. Type III control cards specify input LUNs. The link editor uses the LUNs to obtain the object programs to be included. Type III cards follow the Type II card defining the beginning of a root or overlay, and precede the Type II card that defines the beginning of the next overlay.

The Search and Include cards have identical formats. Both have one or more parameters. Each parameter is either a LUN or a LUN followed by one or more indexed record keys grouped inside parentheses. Specified keys are evaluated as an USASCII string. Keys are limited to a maximum of 8 characters. Keys of less than 8 characters are extended on the right with blank characters. LUNs accessed with keys must be assigned to an indexed file. The Include card identifies particular object modules which DXOLE includes in the output phase being created. Search cards identify object modules to be searched for a definition that resolves an undefined reference in the same phase. Only those modules that resolve external references are included. Search and Include cards are processed when encountered in the input stream and are scanned from left to right. If the LUNs do not have keys, the link editor processes object modules until it finds an end of file. If a key is specified, the link editor processes all object modules until it finds an end of file or another keyed record. DXOLE processes object modules in the input stream as if they were specified by an Include statement. The object module is included in the output phase regardless of whether it resolves external references. A keyname of 'b̸MAINb̸' cannot be specified because DXOLE does not allow imbedded blanks within the list of keys. To include a FORTRAN main program, specify a keyname of 'MAIN'. 'MAIN' will be interpreted as 'b̸MAINb̸' for this program.

The Library card can only specify LUNs. Each LUN must have been assigned to an indexed file which has been formatted like the system library. The link editor does not search libraries until it processes all other Type III control cards. DXOLE then searches the indexed file(s) in the order specified by the Library card before searching the system library file. The system library, LUN9, is not accessed in the Compact mode unless explitcly specified on a Library control card.

8.5.4.4 USING DXOLE WITH BATCH INPUT READER. Executing DXOLE under Batch Input Reader (BIR) requires careful preparation of the input stream if LUNs other than LUN 5 are assigned to the card reader. When DXOLE processes an Include or Search card, the referenced LUN is opened and read until an end of file is found. The input stream for DXOLE must be structured as shown on the following page.

```
ROOT          IDT1
1700          IDT1
 .
 .
 .
1706
INCLUDE        17
1700           IDT2
 .
 .
 .
1706
1700           IDT3
 .
 .
 .
1706
/*                       End of file for LUN 17
SEARCH 18
/*                       End of file for input stream LUN 5
```

The included file is physically defined inside the data for LUN 5. When DXOLE encounters the Include for LUN 17, it opens and reads the file. When the end of file is found, LUN 17 is closed and the next control card is read from LUN 5. This structure also works for the Batch Input Spooler (BIS). However, a more conventional input stream for BIS is as follows:

```
//DATA       5
  ROOT       IDT1
  1700       IDT1
    .
    .
    .
  1706
  INCLUDE    17
  SEARCH     18
/*                        End of file for LUN 5
//DATA       17
  1700       IDT2
    .
    .
    .
  1706
  1700
    .
    .
    .
  1706                    End of file for LUN 17
/*
```

8.5.5 ERROR CODES

Table 8-10 lists the error codes and their corresponding meanings for the DXOLE utility.

Table 8-10. DXOLE Error Messages

Error Number	Message
1	Missing Type 1 Control Card
2	Improper Control Card Format
3	Unidentifiable Control Card
4	Maximum Number of Entries Exceeded
5	Invalid Control Card Argument
6	Unexpected End Of File On LUN
7	Missing Type 2 Control Card
8	Invalid Control Card For DXOLE Mode
9	Multiply Defined Root Segment
10	Missing Argument Following Delimeter
11	Too Many Arguments
12	Undefined Program Name
13	Invalid Delimiter
15	Invalid LUN
16	Invalid Key
17	Missing 1700 Object Record
21	Level Number Not Sequential In Overlay
24	Maximum Number Of Segments Exceeded
25	Maximum Number Of Entries Too Large
26	Invalid Number Specified
27	Numeric Overflow >16 Bit Signed Number
28	Missing Root Segment
30	Multiply Defined Name
31	Invalid Level Number
32	Table Requirements Exceeded Available Memory
34	Object Decks Must Be Relocatable
35	Unidentifiable Object Record
36	Checksum Error
37	Missing 1706 Object Record
38	Common or External Reference Not Defined Before Referenced
39	File Organization Does Not Allow Keyed Access
40	Library File Not Properly Structured
41	Block Data STMTS Must Be In MIP Defining the Common

8.5.6 INPUT FORMAT

The following paragraphs describe the structure and format for input accessed by DXOLE. Refer to the description of the JCL translator in this section of the manual for the structures of:

- Library files
- Data sets
- Object records in the files

8.5.6.1 DATA SET STRUCTURE FOR SEARCH AND INCLUDE. LUNs referenced by Search and Include commands can be accessed either sequentially or randomly. The user specifies the access method by supplying either a LUN only, or a LUN and a key. When supplied with a LUN only, DXOLE processes the LUN sequentially until it finds an end of file. Data sets or devices assigned to these LUN's must allow sequential access of data. All records read from that LUN must be in DXOLE input object format. By specifying a LUN number and the value of a key, the user can specify processing of any part of a key indexed file. (The LPFBLD utility can be used to build a key indexed file with keynames equal to the object deck IDTNAM.) The key is an index for positioning to the first record to be read. Processing is complete when DXOLE encounters either an end of file or another record with a key. The key lengths are restricted to eight characters. If less than eight characters are specified, the editor adds USASCII blanks on the right of the key to make eight characters. The physical structure of the data set is as follows:

```
Record
 Keys      Data Associated with Key (Object Record)

PGM01b     1700 PGM01
           1702
             .
             .
             .
           1706

TABLEb     1700 TABLE
             .
             .
             .
           1706
```

To include object modules from this data set, the programmer specifies the name of the program(s) to be accessed. For example:

```
INCLUDE     1(TABLE, PGM01)
```

A key indexed file with this structure can also be accessed without specifying a key since DX980 file management allows sequential access of records for key indexed files.

8.5.6.2 LIBRARY AND SYSTEM LIBRARY STRUCTURE. DXOLE assumes that LUN 9 has been assigned to either a dummy file or a key indexed file that was created with the DX980 library builder (LIBBLD) utility program. If assigned to DUMMY, no processing other than syntax checking occurs. If assigned to a library file, that file has the characteristics described in the following paragraphs.

Library files must reside on disc and must be in the following format:

Record Key	Data Associated with Key	
QQSINE	1700 SINE	
	1702 SIND	Records defining
	1702 SINE	entry points.
	1702 ARCSIN	
	⋮	
	1706	
SIND	QQSINE	
SINE	QQSINE	
ARCSIN	QQSINE	
QQCOSINE	1700 COSINE	
	1702 DCOS	
	1702 COSINE	
	1702 ARCCOS	
	⋮	
	1706	
DCOS	QQCOSINE	
COSINE	QQCOSINE	
ARCCOS	QQCOSINE	

The 1700 object record of all programs in the library have a key equal to the characters 'QQ' concatenated with the identification name of the program. All other object records for this object deck are sequentially linked to the keyed 1700 record. The utility program that creates the library also makes a key equal to each entry point in the object module (i.e., SINE, SIND, ARCSIN for object deck SINE). The data associated with each key is another key that positions the file at the 1700 object record of the program defining the entry point. DXOLE accesses all library files in the following manner for each unresolved external reference:

(1) The name of the symbol referenced is used as a key to read the library.

(2) If the key does not exist, the symbol is not defined by any object module in the library.

(3) If the key exists, the first 8 characters of the record returned are used as the key of a second read operation.

(4) The data returned with the second read must be the 1700 object record of the module defining the external reference. Also, the second key must exist or the library is improperly structured.

(5) Processing of the object module terminates with the 1706 record.

Library files may also be referenced with Search and Include control cards by specifying the key that positions the file at the 1700 record. That is, the letters 'QQ' followed by the identification name of the program to be accessed:

INCLUDE 9(QQSINE, QQCOSINE)

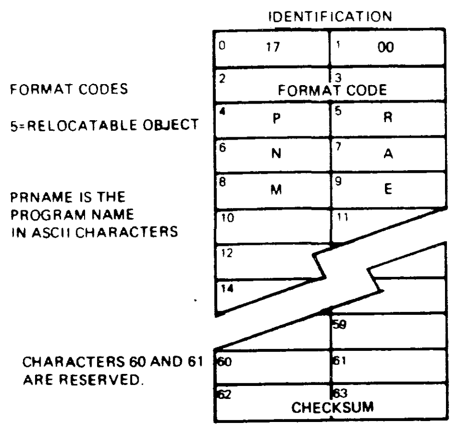
A library file must have a key specified if accessed with an Include or Search card since some of the records contain keys rather than object records. Using an Include on a library might be necessary if the order of the object modules in the load module is important. Otherwise, normal library processing at the end of a program segment will include those program necessary to define external references.

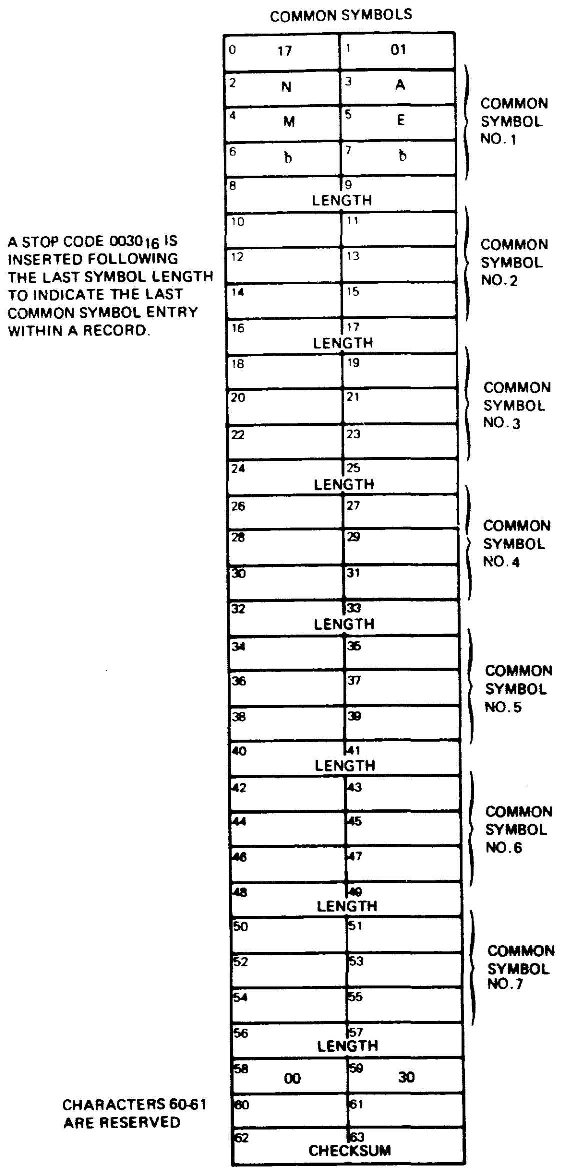
8.5.6.3 OBJECT RECORD FORMAT. DXOLE accepts object from linked sequential files, key indexed files, and devices. Several different data set organizations are allowed. The DXOLE control cards describe the organization of the data set and therefore the processing method used. All input data sets must define at least one object module. Seven standard object records are used by all object modules. These object records are illustrated in figure 8-3. Note that each record contains 32 words. All numbers are hexadecimal, and the object records are independent of the object media. The first word of each record specifies the record type (0-6). The last word in each record is a checksum, which is the two's complement of the sum of the first 31 words.

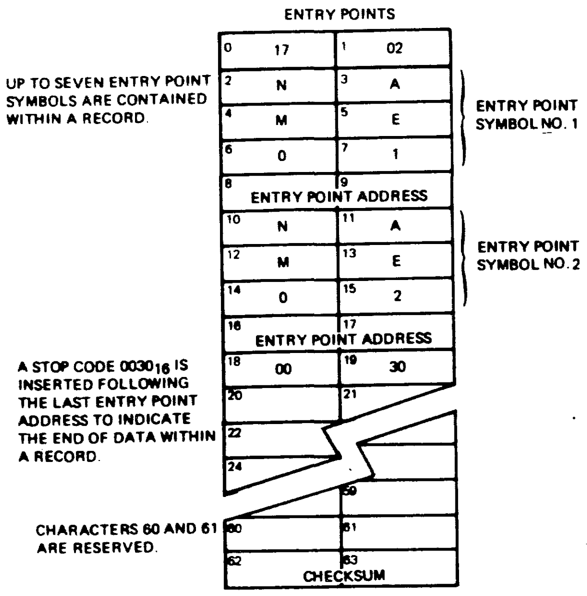
Identification Record. The identification record contains the program name which may be specified, for example, in an IDT assembler directive. If the program name is greater than six characters, it is truncated; if it is less than six characters, trailing blanks are inserted. Leading zeros are inserted in the format code.

Common Symbols. Each record of common symbols contains a maximum of seven symbols. Each symbol contains six characters, and trailing blanks are inserted as required. A stop code is required following the last symbol in each record.

Entry Points. Each record of entry points contains a maximum of seven entries as indicated, for example, by a DEF assembler directive.

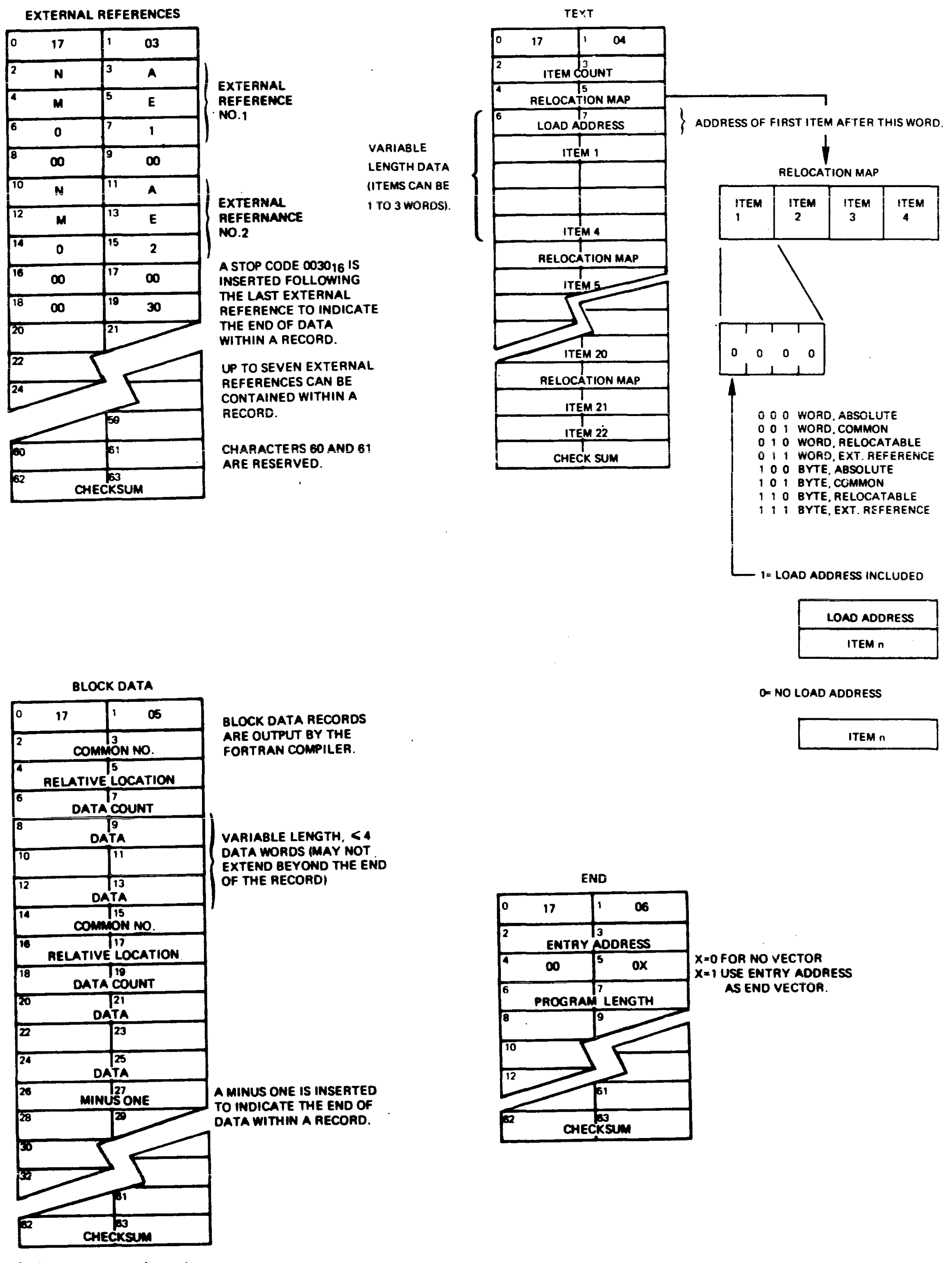





ENTRY POINTS
0 17
1 02
2 N
3 A
4 M
5 E
6 0
7 1
8
9 ENTRY POINT ADDRESS
ENTRY POINT SYMBOL NO. 1
10 N
11 A
12 M
13 E
14 0
15 2
16
17 ENTRY POINT ADDRESS
ENTRY POINT SYMBOL NO. 2
18 00
19 30
20
21
22
24
59
60
61
62
63 CHECKSUM
UP TO SEVEN ENTRY POINT SYMBOLS ARE CONTAINED WITHIN A RECORD.
A STOP CODE 0030₁₆ IS INSERTED FOLLOWING THE LAST ENTRY POINT ADDRESS TO INDICATE THE END OF DATA WITHIN A RECORD.
CHARACTERS 60 AND 61 ARE RESERVED.

(A)130394 (1/2)

Figure 8-3. Object Records (Sheet 1 of 2)



(A)130394 (2/2)

Figure 8-3. Object Records (Sheet 2 of 2)

External Reference Points. Each record of external reference points contains a maximum of seven references as specified, for example, by a REF assembler directive.

Text Records. All text records include an item count between 1 and 22 for the current record. Each item is between one and three words long, and all words in an item are contained within one text record. The number of object record words per item in memory, and the relocation character printed in the first column of an assembly listing appear in table 8-11. The final relocation map in each record applies only to items within the current record, not to any items in the next record.

Table 8-11. Text Record Parameters

Item Type in Relocation Map	Assembly Listing Character	Words per Item in Memory	Words per Item in Text Record
000 - Absolute word	(Blank)	1	1
001 - Common word	C	1	2
010 - Relocatable word	P	1	1
011 - External reference word	X	1	2
100 - Absolute byte	(Blank)	2	2
101 - Common byte	C	2	3
110 - Relocatable byte	P	2	2
111 - External reference byte	X	2	3

Block Data. The block data records contain common numbers which are used to order the symbols in the common symbols records for easy reference. For example, common number one refers to the common names in the first entry of the first common symbols record. Relative location refers to the start of the appropriate common, and data count is the number of words, not items, in the specified common.

End Record. The end record indicates the end of the object program, and contains the program length in words.

8.5.7 DXOLE OUTPUT FORMATS

DXOLE has two output formats: object modules and load modules. Object modules, output by the Compact mode, have the same record formats as previously described for object records. The format of a load module is shown in figure 8-4. Table 8-12 describes the fields of the load module. All programs submitted for execution under DX980 must be in load module format; object modules cannot be executed directly.

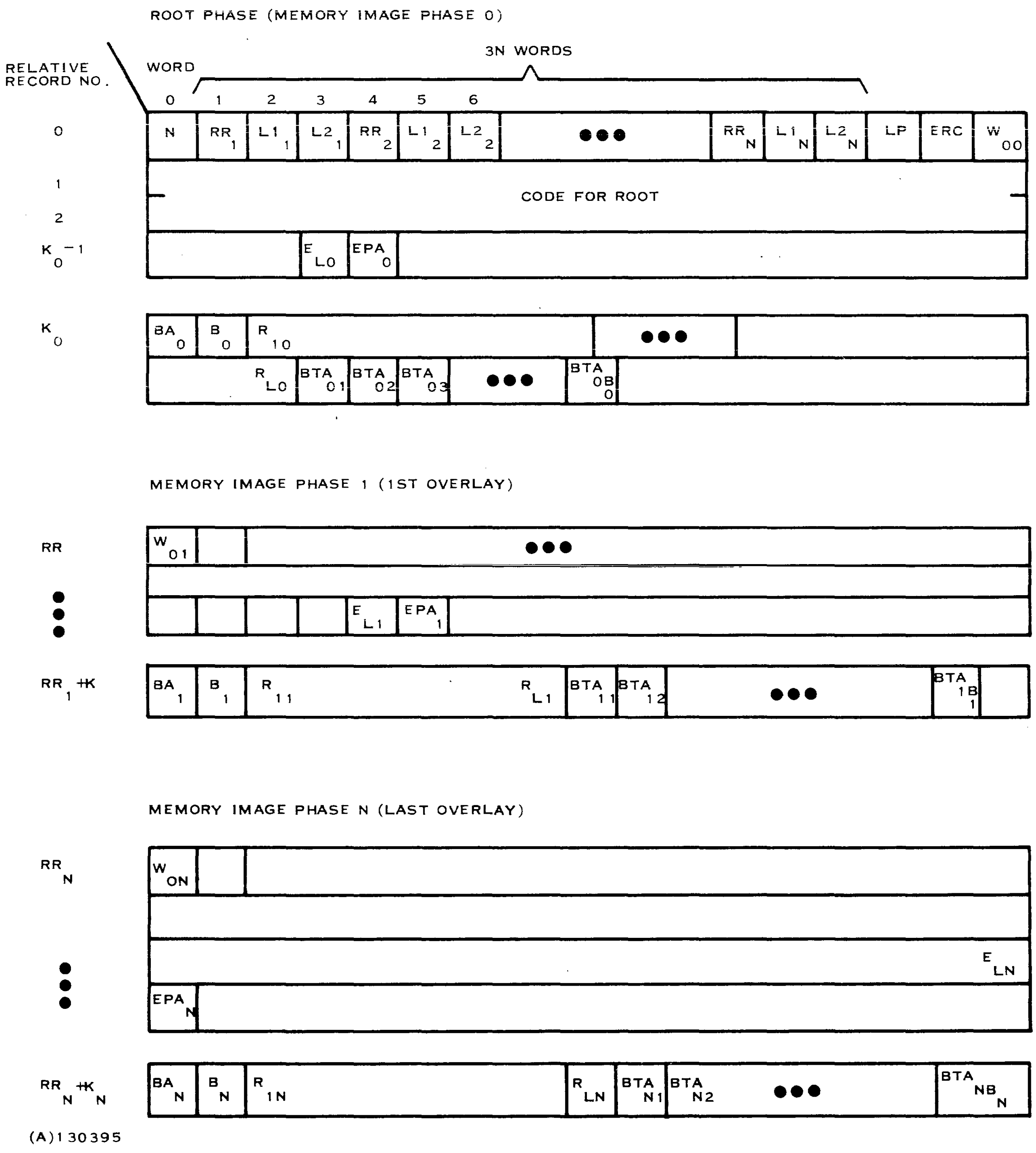


Figure 8-4. DX980 Load Module Records

Table 8-12. Load Module Field Definitions

Field	Definition
N	Number of overlays.
RR_i	Relative record number of the first word of overlay. The relative record number of the root phase (RR_0) is always 0.
$L1_i$	Number of words in the code portion of the overlay including the entry point word. The length of the root phase ($L1_0$) is in the File Control Block (FCB) of the load module file and is not part of the load module proper. $L1_0$ includes the length of the overhead words for the root segment (3n +3 words long).
$L2_i$	Length of the relocation map in words. $L2_0$ is in the FCB of the load module and is not part of the load module proper.
LP	The address of the first word beyond the longest overlay path. On the memory image phase file, this word is numerically equal to the number of words used by the longest overlay path. The word is marked for relocation in the relocate flag table. If the load module is executed in the unprivileged mode, no relocation will be performed. If the load module is executed in the privileged mode, a relocation constant equal to the absolute address of W_{00} will be added. In either case the value in memory is the address of the first word beyond the longest overlay path in the address space of the executing Job.
ERC	In the memory image phase file this word is numerically equal to 3*N+3 and is marked for no relocation in the relocate flag table. The correct value is error checked on the load of the root phase. After the root phase has been loaded the cell in memory is changed to have the absolute address of N.
W_{0i}	Word 0 of the code for overlay i.
E_{Li}	Last word of code for overlay i.
EPA_i	Entry point address for overlay i.
K_i	Number of sectors required to hold the code portion of overlay i. K_i may be calculated from $L1_i$ or, $K_i = ((L1_i+31)/32)*32$, using integer arithmetic.

Table 8-12. Load Module Field Definitions (Continued)

Field	Definition
BA_i	BA is a value set by DXOLE to calculate the relocation bias to be applied to all relocatable words in the code area from the formula, BIAS = FWLA-BA, where FWLA = The address of the first word loaded (ADDR(N) if i = 0 or ADDR(W_{0i}) if $i \neq 0$). For the root segment BA_0 is set to -(3*N+3) so the bias will evaluate to the address of W_{00} when loading a privileged program. For preplanned overlays, BA is set to the preplanned address relative to W_{00}. For unplanned overlays, BA is set to 0.
B_i	Number of byte relocatable items in the code portion of the overlay.
R_{1i}	First word of the relocation bits. There is 1 bit for each word in the overlay. In the root, the first relocation bit corresponds to the N, the number of secondary overlays. A 1 indicates a relocatable item.
R_{Li}	Last word of the relocation bits. Includes a bit for EPA_i.
BTA_{i1}	Address of first byte relocatable item in this overlay.
BTA_{iB_i}	Address of last byte relocatable item in this overlay.

8.6 LIBRARY BUILDER (LIBBLD)

The LIBBLD utility program builds library files for use by the DX980 Overlay Link Editor (DXOLE).

8.6.1 STANDARD JCL PROCEDURE

The following listing is a standard procedure for LIBBLD:

```
.# CREATE LIBBLD,COMMENT,"BUILD LIBRARY FILE           "
/REPLACE LIBBLD     . BUILD LIBRARY FILE .
/EXEC OBJ=(1,SYSTEM,LIBBLD) MEM=(300,2000,1000) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  5 MT1   DEVICE:=DOBJ FILE:=FOBJ BUFFERS=1             . OBJECT INPUT
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.IDT/DEF LISTING
/ASSIGN  9 DISC1 DEVICE:=DLIB FILE:=FLIB REPLACE:=RLIB;
/          BUFFERS=2 INDEXED ACCESS=(ANY,ANY,ANY,ANY);
/          ACCESS:=CLIB ALLOCATE=(1,0,128,20) ALLOCATE:=LLIB;
/          KEYLEN=8                                            . OUTPUT LIB FILE
/END
```

8.6.2 MEMORY PARTITION REQUIREMENTS

Memory requirements of LIBBLD are:

MEM = (300, 2000, 1000)

8.6.3 LUN ASSIGNMENTS

The LIBBLD utility uses three LUN assignments. Table 8-13 outlines the functions of each unit.

Table 8-13. LIBBLD Logical Unit Assignments

LUN	Description
5	Provides object input for addition to the library file. The LIBBLD utility terminates when it finds an end of file in this input.
6	Provides a hard copy listing of the modules (IDT names) and of the defined entry points within each module in the library.
9	Assigned to a key indexed file with a key length of 8. This unit contains the library file to which the object modules will be added.

8.6.4 OPERATION

When using LIBBLD, all names must be unique. No two definitions and no two IDT names can be the same. The object modules in the library can all be added at one time or by several executions of LIBBLD. No object modules can be replaced. LIBBLD checks for duplicate IDT names and duplicate entry point names. To redefine an object module in the library, the library must be entirely rebuilt.

8.6.5 ERROR CODES

LIBBLD error messages are literal and self-explanatory. These messages are as follows:

DUPLICATE DEF NAME, FIRST DEF USED <def name>

DUPLICATE KEY GENERATED FOR IDT

LIBRARY IS FULL. NO MORE RECORDS CAN BE ADDED.

8.6.6 SAMPLE OUTPUT

The resulting library file is an indexed data set that has the following logical organization of keys and associated data:

```
QQIDT1   1700     IDT1
         1702     DEF1  DEF2
           .
           .
           .
         1706

DEF1     QQIDT1
DEF2     QQIDT1
QQIDT2   1700     IDT2
         1702     DEF3
           .
           .
           .
         1706

DEF3     QQIDT2
```

8.7 FILE COPY UTILITY (DXCOPY)

The DXCOPY utility copies and saves data from any type of file or device to any other type of file or device. In addition DXCOPY can list the contents of a file on a printer. Control parameters for the utility specify or override optional functions of the utility. All of these parameters may be omitted to specify their default values. DXCOPY also generates keys for key indexed files.

8.7.1 STANDARD JCL PROCEDURE

The following listing is a standard procedure for DXCOPY:

```
.# CREATE DXCOPY,COMMENT,"GENERAL PURPOSE COPY        "
/REPLACE DXCOPY      . GENERAL PURPOSE COPY .
/EXEC OBJ=(1,SYSTEM,DXCOPY) MEM=(300,3700,2000) PRTY=(1,15);
/           TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  5 DUMMY DEVICE:=DCON                                 . CONTROL/MESSAGE
/ASSIGN  6 DUMMY DEVICE:=DLST FILE:=FLST BUFFERS=1            . LISTING
/ASSIGN  7 DISC1 DEVICE:=DOUT FILE:=FOUT REPLACE:=ROUT;
/           BUFFERS=2 BUFFERS:=BOUT LINKSEQ:=LIN RELREC:=REL;
/           INDEXED:=IND ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=COUT;
/           ALLOCATE=(1,0,128,10) ALLOCATE:=LOUT KEYLEN=6;
/           KEYLEN:=KOUT LRECL=64  LRECL:=GOUT                . OUTPUT
/ASSIGN  8 DISC1 DEVICE:=DIN FILE:=FIN DELETE:=TIN BUFFERS=1   . INPUT
/END
```

8.7.2 MEMORY PARTITION REQUIREMENTS

The job area consists of space for the DXCOPY load module. Since DXCOPY is an overlaid program, this space contains the root segment plus the longest overlay. If there is no listing (i.e. LUN 6 is assigned to DUMMY), the job area size may be specified as 2400. If listings are possible (i.e. LUN 6 assigned to a printing device or file), then the job area size must be specified as 3700.

The job extension area can be calculated according to the following formula:

$$\text{<jearea>} = \sum_{i=1}^{NF} \sum_{j=1}^{NB} (BS_i+j) + 17(NT) + 7(NL) + SS + 11$$

where

NF = Number of files assigned ($0 \leq NF \leq 3$)

NB = Number of buffers per file (1 NB≤x depending on amount of blocking)

BS = Physical record length in words of the individual files

NT = 2 co-resident tasks if the listing LUN (6) is assigned to DUMMY; 3 co-resident tasks if the listing LUN (6) is assigned a printing device or file

NL = 4 LUNs assigned

SS = 600 for dynamic TCB stack requirement if LUN 6 is assigned to DUMMY; 900 if LUN6 is assigned to a printing device or file

Therefore,

<jearea> = 673 + BB if LUN 6 assigned to DUMMY or,
<jearea> = 990 + BB otherwise

where:

BB = total blocking buffer requirements for all files.

A default of MEM=(300, 3700, 2000) has been used in the standard procedure which allows the listing option and 1000 words of file buffer space.

8.7.3 LUN ASSIGNMENTS

The utility program uses four logical unit number (LUN) assignments, as described in table 8-14.

Table 8-14. DXCOPY Logical Unit Assignments

LUN	Description
5	Control input
6	Listing output
7	Data output
8	Data input

8.7.4 OPERATION

The DXCOPY utility is a multitask program that runs under DX980 in the protected mode. A minimum of two tasks run concurrently. A third task is created if a listing is possible (i.e. LUN 6 is assigned to other than DUMMY).

The user enters any of the control parameters on LUN 5. The entry is in free format with one or more blanks between each parameter. All parameters must be within the same record. For many cases, however, the default values of the parameters produce the desired results so that none of the parameters need to be specified. If all of the control characters are entered, the input record is of the following form:

<type>,<key specification>,<definition>,<rewind option>, TRIM,<list>.

The following paragraphs describe each of these parameters and their default values.

8.7.4.1 DATA TYPES AND CONVERSIONS. DXCOPY performs the required file conversion and copying according to the type of input and output files without the data type (<type>) being given explicitly. However, certain types of copies do require explicit definition in order to perform the desired conversion. Data type specifies the format of the input. The output file data is dictated by the input data type and what kind of files are being used (key indexed, relative record, linked sequential, or other device). The following file types may be used for the <type> control parameter:

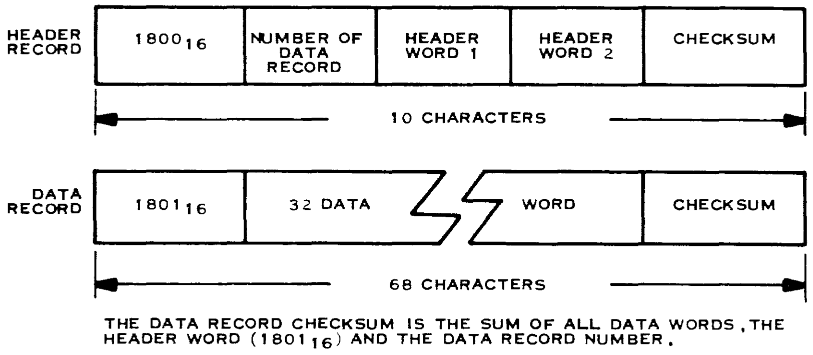
- SOURCE: Source files may exist on any file type. This data type is the default value for linked sequential files and non-disc devices.
- DATA: Data is the default data type when the input file is key indexed. The Data parameter must be stated if the input or output media is binary format paper tape, card, or cassette in order to obtain correct input conversion.
- RELOBJ: Relocatable object is never the default data type. This parameter must be specified to verify checksum, or if the input or output media is cards, paper tape, or cassette.

- MEMI: Memory image is the default data type if either the input file or output file is a relative record file. Memory image files exist in relative records files as two header words in the File Control Block and a group of 32 word records. When DXCOPY transfers the file to any other media, it generates a header record followed by data records and an end of file. Figure 8-5 illustrates the final format. The user must specify either Data or Source for copying relative record files that are not load module files.

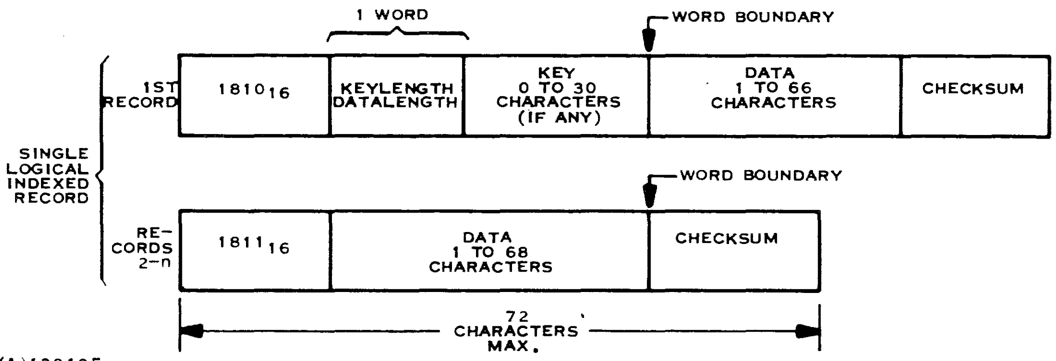
- SAVED: Saved data is created by copying a (key indexed) file to a file that is not a key indexed file. SAVED is the default data type when the input file is indexed and the output file is not indexed. A key specification input of NOKEYS overrides the creation of Saved files. Saved files consist of physical records of 72 characters or less. This organization allows large logical records to be transmitted on card media. (Figure 8-6 illustrates the format of a Saved file.

8.7.4.2 KEY SPECIFICATIONS. Key specification parameters allow the user to set the characteristics of the keys or to delete input keys in the output file. DXCOPY accepts the following words for key specification inputs:

- KEYLEN=<nchar>: This parameter specifies the length of keys in characters. Default length key is 30 characters.



(A)130104 Figure 8-5. Memory Image Format for Files Other than Relative Record



(A)130105

Figure 8-6. Saved File Format

- KEYPOS=<mchar>: This parameter specifies that a key should be generated for each record of input, starting at the character number given to the right of the equal sign (<mchar>).
- NOKEYS: This parameter specifies that any keys found in the input file should not be transferred to the output file. This parameter overrides the Saved data specification.

8.7.4.3 DEFINITION. The user can enter:

FILES=<nfiles>

to specify copying more than one set of data (delimited by an end of file) to the output file. The argument, <nfiles>, is either a number specifying how many data sets to copy, or an asterisk (*) to specify copying all data sets until an end of media is detected. All end of files are transmitted to the output file unless the output file is a key indexed or relative record file. These output file types cause DXCOPY to declare an error and abort when more than one file is transferred.

8.7.4.4 REWIND OPTIONS. DXCOPY normally rewinds the input and output files before beginning the copy. However, the user can prevent rewinding the input file by specifying NOINREW. Similarly, entering NOOUTREW prevents rewinding of the output file.

8.7.4.5 TRIM OPTION. If the user enters TRIM for a control input, DXCOPY trims trailing blanks before copying to the output file.

8.7.4.6 LISTING OPTIONS. If LUN 6 is not assigned to Dummy, DXCOPY generates two types of listings by default. The utility produces a source list if the data type is Source, or an identification listing if the data type is relocatable object (RELOBJ). In addition, the user can enter the following parameters to select printing of optional data:

- RECNO: This parameter specifies printing of record numbers.
- KEYS: This parameter causes the keys to be listed. The key is assumed to be valid USASCII. This listing is printed in hexadecimal if the HEX parameter is also specified.
- HEX: This parameter results in a hexadecimal listing.
- ASCII: This parameter instructs DXCOPY to print the USASCII equivalent to the right of the hexadecimal listing.
- WIDE: This parameter specifies that a 132 column printer is available on LUN 6 instead of an 80 column printer. Hexadecimal dumps are then printed at 16 words per line instead of 8 words per line.

8.7.5 ERROR CODES

When an error occurs during processing, DXCOPY prints an error code on the terminal. Table 8-15 defines the error codes.

Table 8-15. DXCOPY Error Messages

Code	Definition
11001	Unrecognizable parameter specified
11002	Invalid number
11003	More than one data specification
11004	Key length not ≥ 1 and ≤ 32
11005	Invalid key position specified
11006	Trim cannot be specified with given (or default) data type
11007	When copying memory image data from a non-relative record file, the first record was not the expected header record.
11008	When copying memory image data from a non-relative record file, a non-memory image data record was detected
11009	When copying saved data, a record was found that was not an 1810 record when an 1810 was expected
11010	When copying saved data, a record was found that was not an 1811 record when an 1811 record was expected
11011	An unexpected end of file was detected
11012	An end of file was not found when an end of file was expected
11013	A logical error was detected when writing to an indexed file
11014	An embedded end of file was detected, but the output file is either relative record or indexed
11016	Checksum error

8.8 PROGRAM DEBUG (DEB980)

DEB980 is a standalone utility package that aids in debugging user programs without endangering system operation. The package includes commands that permit the user to modify or display contents of memory and registers, and control execution of the program being debugged. Control of execution may be either interactive or batch, and includes loading, setting of breakpoints, and specifying traces. Program instructions may be executed one at a time to allow examination of results following each instruction. DEB980 is designed for debugging single tasking jobs and does not support the following SVCs:

- 6 - Delete Task
- 7 - Suspend Task
- 8 - Post Event
- 30 - Create Task
- 37 - Load
- 38 - Load and Relocate
- 49 - Allocate Resource
- 51 - De-allocate Resource
- 129 - Start Job

This program is completely documented in a separate manual: Model 960 Computer and Model 980 Computer Debug User's Guide and Operating Instructions, Part Number 942760-9701.

8.8.1 STANDARD JCL PROCEDURE

The following listing is a standard procedure for DEB980:

```
.# CREATE DEB980,COMMENT,"PROGRAM DEBUG AID          "
/REPLACE DEB980     . PROGRAM DEBUG AID .
/EXEC OBJ=(1,SYSTEM,DEB980) MEM=(300,6000,600) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN >F0 SC    DEVICE:=DCIN                              , CONTROL INPUT
/ASSIGN >F1 SC    DEVICE:=DMSG                              , SYSTEM MESSAGE
/ASSIGN >F2 DUMMY DEVICE:=DCLST                             , CONTROL LISTING
/ASSIGN >F3 SC    DEVICE:=DUMP FILE:=FUMP BUFFERS=1         , MEMORY DUMP
/ASSIGN >F4 DISC1 DEVICE:=DOBJ FILE:=FOBJ BUFFERS=1         , RELOC OBJECT IN
/ASSIGN   0 DUMMY DEVICE:=DEV0                              , USER PROG LUN 0
/ASSIGN   4 DUMMY DEVICE:=DEV4 FILE:=FIL4 BUFFERS=1         , USER PROG LUN 4
/ASSIGN   5 DUMMY DEVICE:=DEV5 FILE:=FIL5 BUFFERS=1         , USER PROG LUN 5
/ASSIGN   6 DUMMY DEVICE:=DEV6 FILE:=FIL6 BUFFERS=1         , USER PROG LUN 6
/ASSIGN   7 DUMMY DEVICE:=DEV7                              , USER PROG LUN 7
/ASSIGN   8 DUMMY LUNO:=LUN8 DEVICE:=DEV8                   . USER PROG LUN 8
/END
```

8.8.2 MEMORY PARTITION REQUIREMENTS

The job area consists of space for the DEB980 load module plus space dependent on the program being debugged. The following formula specifies the job area size requirements:

<jarea> = 3000 + PS + WA + ST

where

3000	= DEB980 load module size
PS	= Size of user program object module being debugged
WA	= Work area for program being debugged
ST	= Storage for symbol table and commands

The job extension area can be calculated according to the following formula:

$$<\text{jearea}> = \sum_{i=1}^{NF} \sum_{j=1}^{NB} (BS_i+1) + 17(NT) + 7(NL) + SS + 11$$

where

NF = Number of files assigned

NB = Number of buffers per file ($1 \leq NB \leq x$ depending on amount of blocking)

BS = Physical record length in words of the individual files

NT = 1 co-resident task

NL = 5 + number of user program LUNs

SS = 300 for dynamic TCB stack requirement

Therefore

<jearea> = 363 + 5(UL) + BB

where UL = Number of user program LUN's; BB = total blocking buffer requirements for all files.

A default of MEM=(300, 6000, 600) has been used in the standard procedure.

8.8.3 LUN ASSIGNMENTS

NOTE

Unlike most utility programs, DEB980 LUN assignments are hexadecimal numbers.

Table 8-16 lists the logical unit (LUN) assignments for the DEB980 program. In addition, all LUN assignments used in the user program being tested must also be assigned to the DEB980 program.

8.8.4 OPERATION

Refer to the Debug User's Guide for a description of program operation.

8.8.5 ERROR CODES

Refer to Section VI of the Debug User's Guide for a description of error codes and messages generated by DEB980.

8.8.6 SAMPLE INPUT

Refer to the Debug User's Guide for sample input format.

Table 8-16. DEB980 Logical Unit Assignments

LUN	Function	Description
$F0_{16}$	Control input	Inputs debug commands
$F1_{16}$	Control log	Displays operation messages, error messages, simulation termination messages, and command requests
$F2_{16}$	Input log	Lists commands entered on control input device
$F3_{16}$	Dump and trace output	Prints dump and trace messages
$F4_{16}$	Program input	Inputs object program being debugged. This file must be direct output either from the assembler or from DXOLE in the Compact mode.

8.8.7 SAMPLE OUTPUT

Refer to the Debug User's Guide for sample output format.

8.9 SYMBOLIC ASSEMBLER (SAPG)

The general assembler, SAPG, translates 980 symbolic assembly language into object language acceptable to the Model 980 Computer. SAPG is a two pass assembler. During Pass 1, a symbol table is generated as the source program is read. Pass 2 generates the object output and program listing using both the source program and the generated symbol table. More detailed characteristics of the 980 assembler are described in the Model 980 Computer Assembly Language Programmer's Reference Manual, Part Number 943013-9701.

8.9.1 STANDARD JCL PROCEDURE

Standard JCL procedures are listed below for the single job step assembly job and for the 3-step assemble, link and go job sequence.

```
.# CREATE ASMBLR,COMMENT,"ASSEMBLE
/REPLACE ASMBLR     . ASSEMBLE .
/EXEC OBJ=(1,SYSTEM,ASMBLR) MEM=(300,5000,1000) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  0 DUMMY DEVICE:=DMSG SHARE                          ; SYSTEM MESSAGE
/ASSIGN  4 DUMMY DEVICE:=DCON SHARE:=SCON                    ; CONTROL/MESSAGE
/ASSIGN  5 DISC1 DEVICE:=DSRC FILE:=FSRC BUFFERS=1           . SOURCE INPUT
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.SOURCE LIST/ERROR
/ASSIGN  7 DISC1 DEVICE:=DOBJ FILE:=FOBJ NEW:=NOBJ;
/          REPLACE:=ROBJ BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=COBJ;
/          ALLOCATE=(1,0,64,10) ALLOCATE:=LOBJ               : OBJECT OUTPUT
/ASSIGN 16 DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,256,30) : SOURCE SCRATCH
/END
```

(Continued on next text page)

(Continued from previous text page)

```
.# CREATE ASMLGO,COMMENT,"ASSEMBLE, LINK, AND GO      "
/REPLACE ASMLGO     . ASSEMBLE, LINK, AND GO .
/EXEC OBJ=(1,SYSTEM,ASMBLR) MEM=(300,5000,1000) PRTY=(1,15);
/         TIME=-1 MEM:=MEMA
/ASSIGN  0 DUMMY DEVICE:=DMSG SHARE                            . SYSTEM MESSAGE
/ASSIGN  4 DUMMY DEVICE:=DCON SHARE:=SCON                      . CONTROL/MESSAGE
/ASSIGN  5 DISC1 DEVICE:=DSRC FILE:=FSRC BUFFERS=1             . SOURCE INPUT
/ASSIGN  6 SC    DEVICE:=DLSTA FILE:=FLSTA BUFFERS=1           . SOURCE LIST/ERROR
/ASSIGN  7 DISC1 FILE=(TEMP,OBJECT) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,64,10)    . OBJECT OUTPUT
/ASSIGN 16 DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,256,30)   . SOURCE SCRATCH
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(300,12000,3000) PRTY=(1,15);
/          TIME=-1 MEM:=MEML
/ASSIGN  5 DISC1 FILE=(TEMP,OBJECT) DELETE BUFFERS=1           . PRIMARY INPUT/CON
/ASSIGN  6 SC    DEVICE:=DLSTL FILE:=FLSTL BUFFERS=1           . LOADMAP LIST/ERR
/ASSIGN  8 DISC1 FILE=(TEMP,LM) NEW BUFFERS=1 RELREC LRECL=64;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,32,10)    . LOAD MOD OUTPUT
/ASSIGN  9 DUMMY                                               . LIBRARY FILE
/ASSIGN 10 DISC1 FILE=(TEMP,SCRL) DELETE BUFFERS=1             . LINKSEQ SCRATCH
/ASSIGN 13 DISC1 FILE=(TEMP,SCRR) NEW DELETE BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10);
/          LRECL=100                                           . RELREC SCRATCH
/EXEC OBJ=(1,TEMP,LM) MEM=(300,4000,1000) PRTY=(1,15);
/          TIME=100 MEM:=MEMG TIME:=TIMG
/ASSIGN  4 DUMMY DEVICE:=DEV4 FILE:=FIL4 BUFFERS=2             . USER PROG LUN 4
/ASSIGN  5 SC    DEVICE:=DEV5 FILE:=FIL5 BUFFERS=2             . USER PROG LUN 5
/ASSIGN  6 SC    DEVICE:=DEV6 FILE:=FIL6 BUFFERS=2             . USER PROG LUN 6
/ASSIGN  7 DUMMY DEVICE:=DEV7                                  . USER PROG LUN 7
/END
```

8.9.2 MEMORY PARTITION REQUIREMENTS

The job area consists of space for the load module of the assembler and any work space required by the assembler for symbol table storage. The number of words required for a single symbol table entry is a function of the number of characters in the symbol as shown in table 8-17. The job area requirement may be calculated using the following formula:

<jarea> = 3650 + 3(S12) + 4(S34) + 5(S56)

where:

3650	= SAPG load module size
S12	= Number of symbols in table with a 1 or 2 character length
S34	= Number of symbols in table with a 3 or 4 character length
S56	= Number of symbols in table with a 5 or 6 character length

A job area size of 5000 words will allow the assembly of a program with as many as 270 symbols.

Table 8-17. Symbol Table Memory Allocation

Symbol Length in Characters	Words Required in Symbol Table
1-2	3
3-4	4
5-6	5

The job extension area can be calculated according to the following formula:

$$\text{<jearea>} = \sum_{i=1}^{NF} \sum_{j=1}^{NB} (BS_i+1) + 17(NT) + 7(NL) + SS + 11$$

where

NF = Number of files assigned ($0 \leq NF \leq 3$)

NB = Number of buffers per file ($1 \leq NB \leq x$, depending on amount of blocking)

BS = Physical record length in words of the individual files

NT = 1 co-resident task

NL = 6 LUNs assigned

SS = 300 for Dynamic TCB stack requirement

Therefore,

<jearea> = 370 + BB

where BB is the total blocking buffer requirements for all files.

To assemble a program using the standard JCL procedure provided previously in this section using input and output files having physical record lengths of 256 words or less, a job extension area of 1000 words is adequate.

8.9.3 LUN ASSIGNMENTS

Table 8-18 lists the logical unit assignments for the assembler.

Table 8-18. SAPG Assembler Logical Unit Assignments

LUN	Description	Comments
0	System error messages	Any printing device
4	Messages and control	Any input/output device
5	Source input	Any input device/file. If not rewindable, LUN 16 must be assigned to a rewindable device/file, or input must be manually input twice.
6	Output listing and error messages	Any printing device/file
7	Relocatable object output	Output device/file
16	Temporary storage of source input	Any input device/file. If rewindable, it is used as input to Pass 2; if not rewindable, LUN 5 provides input to Pass 2.

8.9.4 OPERATION

The SAPG assembler is a single-task program that runs under DX980 in the protected mode. Figure 8-7 illustrates the operation and LUN assignments for SAPG. The assembler first determines if the source input (LUN 5) is rewindable. If it is not rewindable, SAPG prints the message:

```
READY SOURCE, HIT C/R
```

on LUN 4. When this message appears, the operator should ready the source device, if necessary, and then select a carriage return on LUN 4. This control interaction can be eliminated by assigning the message and control (LUN 4) to DUMMY. Pass 1 reads the source from LUN 5 until it reads a record containing an END directive. After the END is processed, Pass 1 terminates and Pass 2 begins.

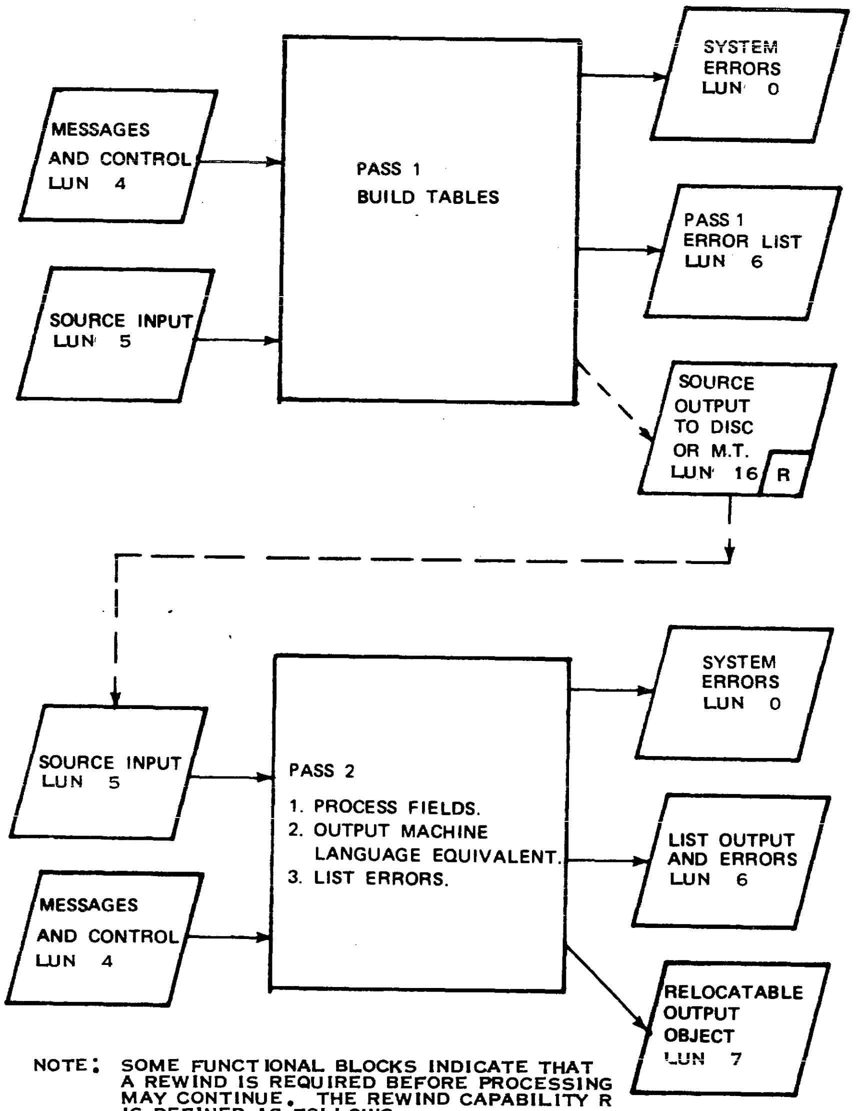
Pass 2 obtains the source input from one of two sources:

1) If LUN 16 has been assigned to a rewindable device or to a file, Pass 1 copies the source input to LUN 16 and Pass 2 uses LUN 16 for its input.

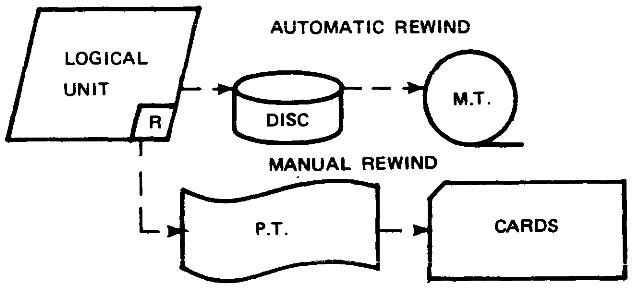
2) If LUN 16 has been assigned to a non-rewindable device, Pass 2 uses the primary source input, (LUN 5) for input. If LUN 5 is assigned to a rewindable device or to a file, Pass 2 automatically rewinds LUN 5. If LUN 5 is not rewindable, the source input control message:

   ```
   READY SOURCE, HIT C/R
   ```

   is again printed to LUN 4. After repositioning the source in LUN 5, select carriage return.


MESSAGES AND CONTROL LUN 4
SOURCE INPUT LUN 5
PASS 1 BUILD TABLES
SYSTEM ERRORS LUN 0
PASS 1 ERROR LIST LUN 6
SOURCE OUTPUT TO DISC OR M.T. LUN 16
R
SOURCE INPUT LUN 5
MESSAGES AND CONTROL LUN 4
PASS 2
1. PROCESS FIELDS.
2. OUTPUT MACHINE LANGUAGE EQUIVALENT.
3. LIST ERRORS.
SYSTEM ERRORS LUN 0
LIST OUTPUT AND ERRORS LUN 6
RELOCATABLE OUTPUT OBJECT LUN 7

NOTE: SOME FUNCTIONAL BLOCKS INDICATE THAT A REWIND IS REQUIRED BEFORE PROCESSING MAY CONTINUE. THE REWIND CAPABILITY R IS DEFINED AS FOLLOWS.



(A)130097

Figure 8-7. SAPG Assembler Functional Diagram

Pass 2 produces a listing and the bulk of the object output, and terminates processing when it encounters an END directive. At that time Pass 1 restarts. The SAPG assembler continues to process source inputs terminated by an END directive and generates corresponding output object modules until it reads an End of File (EOF) record (/*) to terminate the assembly. It then outputs an EOF record and trailer at the end of the object output to indicate assembler termination. To execute any object program under DX980, the module must be linked using the link editor, DXOLE.

8.9.5 ERROR CODES

When SAPG reads the source program, it may detect format errors. Detection of an error prints a diagnostic message on LUN 6. If the error is detected in Pass 1, the message appears before the listing. If the error is detected in Pass 2, the message is printed adjacent to the source line in question. A total number of errors encountered in the assembly is given at the end of the listing. Table 8-19 lists the possible error messages printed by SAPG.

8.9.6 SAMPLE INPUT

Figure 8-8 illustrates the source input to the SAPG assembler.

8.9.7 SAMPLE OUTPUT

Figure 8-9 illustrates the object module output from the SAPG assembler.

8.10 FORTRAN IV COMPILER

The FORTRAN compiler furnished with the DX980 operating system exceeds the specifications set forth in the American National Standards Institute publication number USAS X3.9-1966. The FORTRAN compiler is a 1-pass, 2-phase compiler that outputs an intermediate pseudo object of tables and linkage information at the end of Phase 1. Using the mass storage capability of the DX980 system, the operator need not handle the pseudo object output. More detailed characteristics of this compiler are described in the FORTRAN manual referenced in the Preface to this manual.

8.10.1 STANDARD JCL PROCEDURE

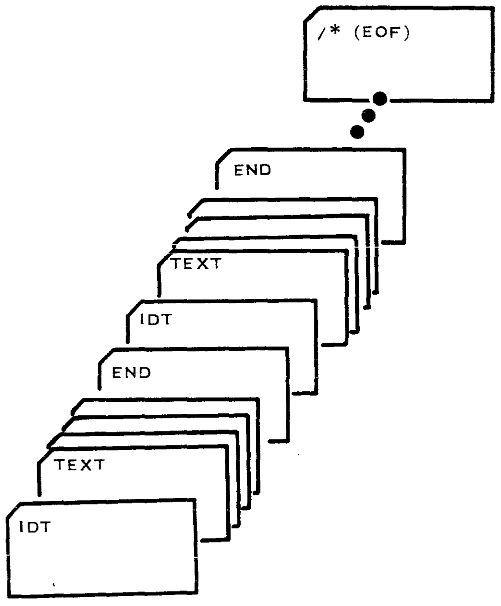
Standard JCL procedures are listed below for the 2-step FORTRAN compile job sequence (Phase 1 and Phase 2) and for the 4-step FORTRAN compile, link and go job sequence. This procedure does not permit searching of any alternate, user-supplied libraries other than the standard FORTRAN library. To enable additional searches, a separate JCL procedure must be written with modifications to the job step portion of DXOLE. These modifications include assignment of the object module (TEMP,OBJECT) to an unused LUN, assignment of the alternate library file to another unused LUN, and assignment of the required DXOLE control commands (either in a file or from an input device) to LUN 5.

```
.# CREATE FTNPS1,COMMENT,"FORTRAN PHASE 1 COMPILE      "
/REPLACE FTNPS1      . FORTRAN PHASE 1 COMPILE .
/EXEC OBJ=(1,SYSTEM,FTN) MEM=(300,8000,1000) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  0 DUMMY DEVICE:=DMSG SHARE                          . SYSTEM MESSAGE
/ASSIGN  5 DISC1 DEVICE:=DSRC FILE:=FSRC BUFFERS=1           . SOURCE INPUT
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.SOURCE LIST/ERROR
/ASSIGN  7 DISC1 DEVICE:=DINT FILE:=FINT BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(1,0,64,30);    . SOURCE SCRATCH
/          NEW:=NINT REPLACE:=RINT ACCESS:=CINT ALLOCATE:=LINT
/END
.# CREATE FTNPS2,COMMENT,"FORTRAN PHASE 2 COMPILE      "
/REPLACE FTNPS2      . FORTRAN PHASE 2 COMPILE .
/EXEC OBJ=(1,SYSTEM,FTNPS2) MEM=(300,8000,1000) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  0 DUMMY DEVICE:=DMSG SHARE                          . SYSTEM MESSAGE
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.ERROR MMSSG&5
/ASSIGN  7 DISC1 DEVICE:=DOBJ FILE:=FOBJ BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(1,0,64,10);    . OBJECT OUTPUT
/          NEW:=NOBJ REPLACE:=ROBJ ACCESS:=COBJ ALLOCATE:=LOBJ
/ASSIGN  8 DISC1 DEVICE:=DINT FILE:=FINT BUFFERS=1            . INTERMED OBJECT
/END


.# CREATE FTNLGO,COMMENT,"FORTRAN COMPILE, LINK,AND GO"
/REPLACE FTNLGO      . FORTRAN COMPILE, LINK, AND GO .
/EXEC OBJ=(1,SYSTEM,FTN) MEM=(300,10000,1000) PRTY=(1,15);
/          TIME=-1, MEM:=MEMC
/ASSIGN  0 DUMMY DEVICE:=DMSG SHARE                           . SYSTEM MESSAGE
/ASSIGN  5 DISC1 DEVICE:=DSRC FILE:=FSRC BUFFERS=1            . SOURCE INPUT
/ASSIGN  6 SC    DEVICE:=DLST1 FILE:=FLST BUFFERS=1           . SOURCE LIST/ERROR
/ASSIGN  7 DISC1 FILE=(TEMP,PHASE1) NEW BUFFERS=1 LINKSEQ;    . INTERMED OBJECT
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,64,30)   . SOURCE SCRATCH
/EXEC OBJ=(1,SYSTEM,FTNPS2) MEM=(300,8000,1000) PRTY=(1,15);
/          TIME=-1 MEM:=MEMC
/ASSIGN  0 DUMMY                                              . SYSTEM MESSAGE
/ASSIGN  6 SC    DEVICE:=DLST2                                . ERROR MESSAGE
/ASSIGN  7 DISC1 FILE=(TEMP,OBJECT) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,64,10)   . OBJECT OUTPUT
/ASSIGN  8 DISC1 FILE=(TEMP,PHASE1) BUFFERS=1                 . INTERMED OBJECT
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(300,12000,3000) PRTY=(1,15);
/          TIME=-1 MEM:=MEML
/ASSIGN  5 DISC1 FILE=(TEMP,OBJECT) DELETE BUFFERS=1          . PRIMARY INPUT/CON
/ASSIGN  6 SC    DEVICE:=DLSTL FILE:=FLST BUFFERS=1           . LOADMAP LIST/ERR
/ASSIGN  8 DISC1 FILE=(TEMP,LM) NEW BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,32,10);  . LOAD MOD OUTPUT
/          LRECL=64
/ASSIGN  9 DISC1 FILE=(SYSTEM,USRFTN) BUFFERS=2               . LIBRARY
/ASSIGN 10 DISC1 FILE=(TEMP,PHASE1) DELETE BUFFERS=1          . LINKSEQ SCRATCH
/ASSIGN 13 DISC1 FILE=(TEMP,SCRR) NEW DELETE BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10);. RELREC SCRATCH
/          LRECL=100
/EXEC OBJ=(1,TEMP,LM) MEM=(300,8000,1000) PRTY=(1,15);
/          TIME=100 MEM:=MEMG TIME:=TIMG
/ASSIGN   0 SC    DEVICE:=DMSG                                . SYSTEM MESSAGE
/ASSIGN >B0 SC    DEVICE:=DEV0                                . USER LUN 0
/ASSIGN >B1 DUMMY DEVICE:=DEV1                                . USER LUN 1
/ASSIGN >B5 SC    DEVICE:=DEV5 FILE:=FIL5 BUFFERS=2           . USER LUN 5=INPUT
/ASSIGN >B6 SC    DEVICE:=DEV6 FILE:=FIL6 BUFFERS=2           . USER LUN 6=OUTPUT
/ASSIGN >B8 DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300.32,10)  . USER SCRATCH FILE
/END
```

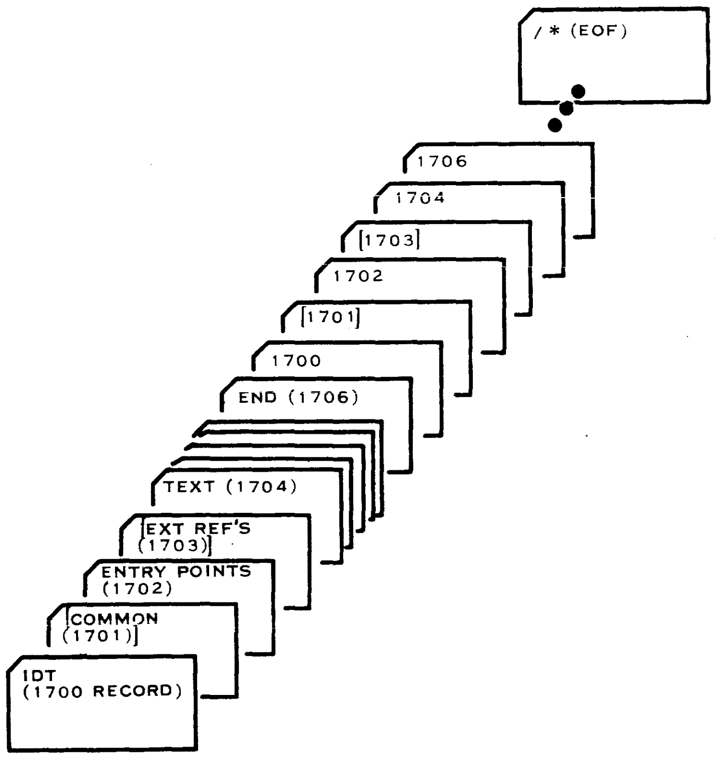
Table 8-19. SAPG Error Messages

Message Number	Message	Meaning (and Corrective Action)
1	FIELD SZ	Address beyond reach (use @ for extended format)
2	UNDF OP	Undefined operation code (check list of valid op codes)
3	LONG SYM	Symbol > 6 characters
4	MDF O/F	OPD or FRM multiply defined (rename label)
5	FRM > 16	FRM fields contain more than 16 bits
6	CAD > 10	Address expression > 10 elements
7	UNDF SYM	Symbol not defined (label probably omitted)
8	MDF SYM	Symbol multiply defined (rename labels)
9	RELOC	A relocation error (use only relocatable label in arithmetic expression, or ORG statement can use only one relocatable label)
10	SYM OVF	Too many symbols have been defined (cut out symbols or divide program)
11	BAD NUM	Numeric element not valid (properly define item in label or address field)
12	IMP R/D	A REF or DEF symbol has been used improperly (REF symbol defined inside and outside the program; DEF symbol not defined in the program)
13	X RF USE	A REF symbol has appeared invalidly in an unrelocatable expression
14	IXB ERR	Address mode error (improper use of IXB field)
15	OPD ERR	No such OPD format number
16	ADR MODE	Illegal addressing mode (improperly written address)



(A)130396

Figure 8-8. Assembly Language Source Input to SAPG



(A)130397

Figure 8-9. Object Module Output from SAPG

8.10.2 MEMORY PARTITION REQUIREMENTS

The job area required to run Phase 1 and Phase 2 of the FORTRAN compiler consists of space for the load modules as well as dynamic workspace. For Phase 1, this job area is defined by the equation:

$$\langle jarea\rangle_1 = LM_1 + WS_1$$

where

LM_1 = 6650 (Phase 1 compiler load module size)

WS_1 = workspace required (program size dependent)

A job area size of 8000 words for the Phase 1 compilation job step allows for compiling a small FORTRAN program. For Phase 2, the job area size is defined by the equation:

$$\langle jarea\rangle_2 = LM_2 + WS_2$$

where

$\langle jarea\rangle_2$ = job area size for Phase 2

LM_2 = 2800 (Phase 2 compiler load module size)

WS_2 = workspace required (program size dependent)

A job area size of 8000 words for the Phase 2 compilation job step allows for compiling a small FORTRAN program.

The job extension area can be calculated according to the following formula:

$$\langle jearea\rangle = \sum_{i=1}^{NF} \sum_{j=1}^{NB} (BS_i + 1) + 17(NT) + 7(NL) + SS + 11$$

where

NF = Number of files assigned to the job step ($0 \le NF \le 3$)

NB = Number of buffers per file ($0 \le NB \le x$, depending upon amount of blocking)

BS = Physical record length in words of the individual files assigned to the job step

NT = 1 co-resident task

NL = 4 LUNs assigned to the compiler

SS = Dynamic TCB stack requirement (300 for compiler)

Therefore,

$$\langle jearea\rangle = 356 + BB$$

where BB is the total blocking buffer requirements for all files assigned to the job step. For compiling the previously listed standard JCL procedure using input and output files having physical record lengths of 256 words or less, a job extension area of 1000 words is adequate. For Phase 2, the required jearea is also 1000.

8.10.3 LUN ASSIGNMENTS

Tables 8-20 and 8-21 list the logical unit assignments for the Phase 1 and Phase 2 FORTRAN compiler job steps, respectively.

Table 8-20. FORTRAN Compiler Phase 1 LUN Assignments

LUN	Description	Comments
0	System error messages	Any input/output printing device
5	Source input	Any input device/file
6	Output listing and error messages	Any printing device/file
7	Output to Phase 2	Output device/file

Table 8-21. FORTRAN Compiler Phase 2 LUN Assignments

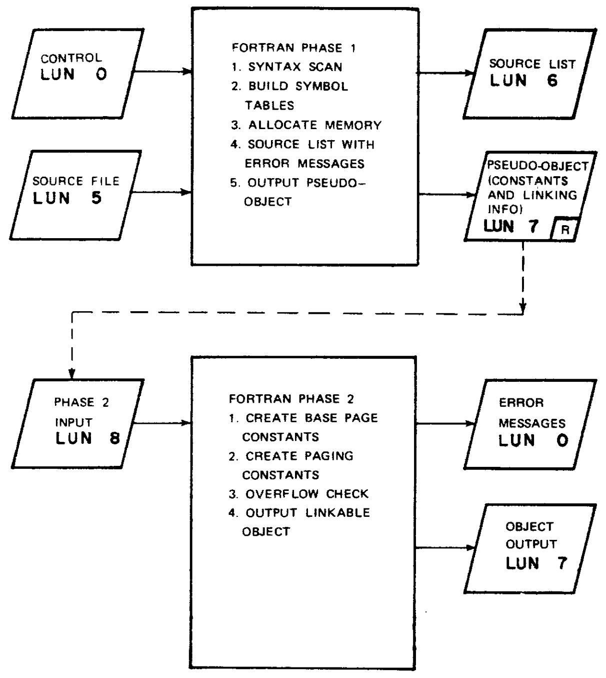
LUN	Description	Comments
0	System error messages	Any input/output printing device
6	Output listing and error messages	Any printing device/file
7	Relocatable object output	Output device/file
8	Input from Phase 1	Any input device/file

8.10.4 OPERATION

The FORTRAN compiler is a single-task program that runs under DX980 in the protected mode. The compilation of a FORTRAN program consists of a 2-step job: Phase 1 and Phase 2 compilations. Figure 8-10 provides a functional diagram of the FORTRAN compiler, including LUN assignments.

The FORTRAN Phase 1 compiler first determines if the source input (LUN 5) is rewindable. If it is not rewindable, the compiler prints the message:

```
READY INPUT, HIT C/R
```



NOTE:
SYSTEM LOGICAL UNITS CORRESPONDING TO FORTRAN UNITS THAT HAVE BEEN USED IN A FORTRAN PROGRAM MUST BE ASSIGNED TO APPLICABLE PHYSICAL DEVICES BEFORE THE FORTRAN PROGRAM IS RUN. THESE SYSTEM RUN-TIME UNITS ARE LISTED IN THE FOLLOWING TABLE:

FORTRAN RUN-TIME LOGICAL UNITS	
FORTRAN UNIT	SYSTEM UNIT
0 TO 9	B0 TO B9
10 TO 15	BA TO BF
16 TO 25	C0 TO C9
26 TO 31	CA TO CF

(A)130106

Figure 8-10. FORTRAN Execution Functional Diagram

on LUN 0. When this message appears, the operator should ready the source device and then select a carriage return on LUN 0. Phase 1 then reads the source input and outputs an intermediate pseudo-object of tables and linkage information to LUN 7. If after the END statement of a subprogram Phase 1 encounters a record containing two asterisks (**) as the first two characters, it produces delimiting characters for Phase 2 and restarts itself to process an additional subprogram. This procedure compiles several subprograms with a single application of the compiler. Once Phase 1 encounters an End of File (EOF) record (minimally a /*), Phase 1 terminates and Phase 2 begins.

The FORTRAN compiler Phase 2 determines if the intermediate object input is rewindable. If the input is not rewindable, the compiler prints the message:

READY INPUT, HIT C/R

on LUN 0. When this message appears, the operator should ready the input device (LUN 8) if necessary, and then select a carriage return on LUN 0. Phase 2 then reads the intermediate object and completes the compilation process. If no errors occur in this process, Phase 2 prints the message:

COMPILATION COMPLETE

on LUN 6.

The object code produced by 980 FORTRAN is in relocatable format suitable for input to the link editor, DXOLE. The program ID for the object module is taken from the name of the function or subroutine subprogram. The name cannot exceed six characters in length. If it is less than six characters, the name field is right filled with blanks. Those programs that are not identified as functions or subroutines are automatically named ƀMAINƀ. DXOLE does not recognize an IDT name beginning with ƀ (blank space) within an Include control record. Therefore, DXOLE cannot access a main program produced by the FORTRAN compiler from within an indexed file by using the keyname.

The object output from Phase 2 is ready for linkage editing with the FORTRAN library to acquire any library modules referenced in the program. The FORTRAN library contains a collection of commonly used subprograms typically referenced by programs generated by the FORTRAN compiler. The library is provided to DX980 users in the form of a key indexed library file identified as (SYSTEM, USRFTN). Assign this library file to LUN 9 of the DXOLE job step for most efficient linking. The output load module from DXOLE may then be executed. The FORTRAN runtime package adds a value of $B0_{16}$ to the LUN of all user I/O requests. For example, if a user program specifies an output for LUN 6, the JCL needed to execute the load module must assign LUN $B6_{16}$ to the program. FORTRAN runtime error messages (explained in paragraph 8.10.5) are printed on LUN 0. Therefore, the JCL procedure for executing any FORTRAN program must also assign LUN 0. The standard JCL procedures provided in paragraph 8.10.1, illustrate the assignment of LUN 0 for these error messages.

8.10.5 ERROR CODES

Error messages may originate from several sources in the FORTRAN compile, link and go sequence. Table 8-22 lists the error codes generated by the FORTRAN compiler Phase 1. Table 8-23 lists the error codes generated by the FORTRAN compiler Phase 2. Table 8-24 lists the error codes generated by the FORTRAN runtime library. These codes can be produced during the actual execution of the load module.

Table 8-22. FORTRAN Compiler Phase 1 Error Messages

Comment	Meaning
Line-by-Line Messages	
SYNTAX	Erroneous punctuation or illegally constructed arithmetic expression.
NUMBER	A constant or label is too large or is incorrectly constructed.

Table 8-22. FORTRAN Compiler Phase 1 Error Messages (Continued)

Comment	Meaning
ID CONFLICT	The identifier marked is being used in a context which contradicts a previous explicit or implicit declaration.
TYPE CONFLICT	The identifier or expression marked is in conflict with another identifier or expression.
MODE	The identifier or expression marked has a type in conflict with the context.
SUBSCRIPTS	The number of subscript expressions used in an array does not equal the number declared for the array.
ALLOCATION	A non-dummy variable has been given as an adjustable dimension, or a variable has been placed in COMMON more than once, or a dummy variable appears in a COMMON or EQUIVALENCE statement.
ORDER	The statement appears in the program at a point in violation to the stated rules governing the order of appearance of statements in the program.
MISSING LABEL	The statement must have a label in order to be reached or referenced.
DATA COUNT	The number of items in the data list of a DATA statement is not equal to the number of items in the variable list.
BLOCK DATA	An executable statement appears in a BLOCK DATA subprogram.
OVERFLOW	The statement caused the compiler capacity to be exceeded. Compilation does not continue.
End-of-Compilation Messages	
LABEL ERRORS	Labeling a Do loop structure errors. The message is followed by a list of statement numbers.
ALLOCATION ERRORS	Memory allocation errors. The message is followed by a list of identifiers to which memory cannot be allocated, due to programming errors.

Table 8-23. FORTRAN Compiler Phase 2 Error/Termination Messages

Comment	Meaning
INCORRECT FORMAT	The Phase 1 output contains a format error
PROGRM OVER	The program exceeds Phase 2 capacity
BSPAGE OVER	The base page exceeds the capacity allotted
INVALID CODE	The Phase 1 output contains a code not recognized by Phase 2
FIELD SIZE ERROR	The IAL statement on the indicated line referenced a location that is not directly addressable.
COMPILATION ABORTED - PASS 1 ERROR	Error encountered in intermediate object input from Phase 1
ERRORS - COMPILATION ABORTED	Errors encountered in Phase 2.
PROGRAM END	Phase 1 output successfully input to Phase 2.
COMPILATION COMPLETE	No errors encountered in either phase

Table 8-24. Runtime Error Messages

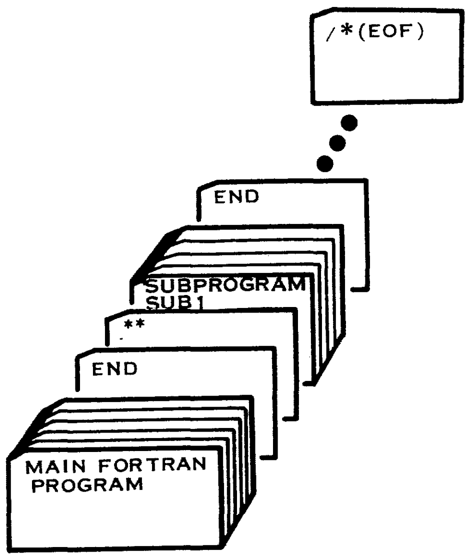
Message	Meaning
NOTE The user program is terminated in all cases except the **WARNING** condition.	
ILLEGAL FORMAT CHARACTER	Illegal character encountered in runtime format statement.
ILLEGAL INPUT CHARACTER	Illegal character encountered in input stream during READ execution.
FORMAT PARENTHESIS ERROR	Runtime format statement contains unbalanced set of parenthesis.
UNDER/OVER FLOW	Real number in input stream or result of a floating point operation is outside range of numbers allowed.

Table 8-24. Runtime Error Messages (Continued)

Message	Meaning
WARNING RECORD SIZE ERROR	User program attempting to input or output record containing greater than 132 characters.
INPUT (OUTPUT, REWIND, BACKSP, ENDFILE) UNIT LIMIT ERROR	I/O was attempted on a FORTRAN unit greater than 7.
OUT OF DATA	End-of-file encountered during a READ operation.
ERR, OR WRTBIN (WRTBCD, REDBCD, REDBIN, ENDFIL, REWIND, BACKSP) COMM IGND ON UNIT B0 (1, 2, . . . , 7) JOB ABORTED	Hardware error, or attempted operation is illegal on device specified, i.e., rewind card punch is illegal.
DIVIDE CHECK	An attempt was made to perform a floating-point division by zero.

8.10.6 SAMPLE INPUT

Figure 8-11 illustrates input format for the FORTRAN compiler.

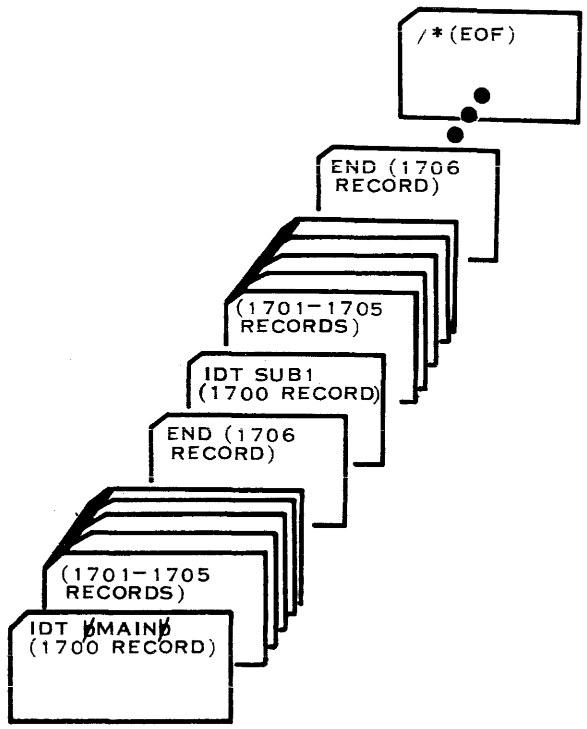


(A)130399

Figure 8-11. FORTRAN Source Input to FORTRAN Compiler

8.10.7 SAMPLE OUTPUT

Figure 8-12 illustrates output format from the FORTRAN Compiler.



(A)130400

Figure 8-12. Object Module Output from FORTRAN Compiler

8.11 LOAD MODULE UPDATE (LMUPDT)

This utility updates an unplanned overlay on a load module file. The new overlay must have been link edited as a subsystem using DXOLE. The relative record file of the new overlay must contain a directory entry for the overlay, a dummy root phase, and the new overlay. Use the following control cards for DXOLE:

ƀ SUBSYSTEM OVLY	Subsystem Mode, Overlay
ƀ ROOT MAIN.	Dummy Root Phase
ƀ SEGMENT 1	Level 1 Overlay
Object Deck	Overlay 1
ƀ SEGMENT 1	Level 1 Overlay
Object Deck	Overlay 2

•
•
•

(listing continued on next text page)

(listing continued from previous text page)

ƀ SEGMENT 1	Level 1 Overlay
Object Deck	Overlay n (n Load Modules to be Updated)
/*	End of File Terminates DXOLE Input

8.11.1 STANDARD JCL PROCEDURE

The following listing is a standard procedure for LMUPDT:

```
.# CREATE LMUPDT,COMMENT,"LOAD MODULE UPDATE          "
/REPLACE LMUPDT      . LOAD MODULE UPDATE .
/EXEC OBJ=(1,SYSTEM,LMUPDT) MEM=(300,3000,500) PRTY=(1,1);
/            TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  4 SC     DEVICE:=DMSG                              ; SYSTEM MESSAGE
/ASSIGN  5 SC     DEVICE:=DCON FILE:=FCON BUFFERS=1         ; CONTROL INPUT
/ASSIGN  6 DISC1 DEVICE:=DLM FILE:=FLM BUFFERS=1            ; LOAD MODULE INPUT
/ASSIGN  7 DISC1 DEVICE:=DUPD FILE:=FUPD BUFFERS=1          ; UPDATE FILE
/END

.# CREATE LINKUP,COMMENT,"LINK MOD AND UPDATE L M FILE"
/REPLACE LINKUP      . LINK MODULE AND UPDATE LOAD MODULE FILE .
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(300,12000,3000) PRTY=(1,1);
/            TIME=-1 MEM:=MEML
/ASSIGN  1 DUMMY DEVICE:=DOBJ FILE:=FOBJ BUFFERS=2          ; SECONDARY OBJ IN
/ASSIGN  5 DISC1 DEVICE:=DIN FILE:=FIN BUFFERS=1            ; PRIMARY INPUT/CON
/ASSIGN  6 SC     DEVICE:=DLST FILE:=FLST BUFFERS=1         ; LOADMAP LIST/ERR
/ASSIGN  8 DISC1 FILE=(TEMP,LM) NEW BUFFERS=1 RELREC LRECL=64;
/            ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,32,10)  ; LOAD MOD OUTPUT
/ASSIGN  9 DISC1 DEVICE:=DPLX FILE=(SYSTEM,USRPLX) FILE:=FPLX;
/            BUFFERS=2                                      ; PLEXUS LIBRARY
/ASSIGN 10 DISC1 FILE=(TEMP,SCRL) NEW DELETE BUFFERS=1;
/            LINKSEQ ACCESS=(ANY,ANY,ANY,ANY);              ; LINKSEQ SCRATCH
/            ALLOCATE=(10,300,256,30)
/ASSIGN 11 DISC1 DEVICE:=DEXT FILE=(SYSTEM,DXEXTD) FILE:=FEXT;
/            BUFFERS=1                                      ; SYSTEM EXT DEFS
/ASSIGN 13 DISC1 FILE=(TEMP,SCRR) NEW DELETE BUFFERS=1 RELREC;
/            ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10);
/            LRECL=100                                      ; RELREC SCRATCH
/EXEC OBJ=(1,SYSTEM,LMUPDT) MEM=(300,3000,500) PRTY=(1,1);
/            TIME=-1 MEM:=MEMU
/ASSIGN  4 SC     DEVICE:=DMSG                              ; SYSTEM MESSAGE
/ASSIGN  5 SC     DEVICE:=DCON SHARE:=SCON                  ; CONTROL INPUT
/ASSIGN  6 DISC1 FILE=(TEMP,LM) BUFFERS=1                   ; LOAD MODULE INPUT
/ASSIGN  7 DISC1 DEVICE:=DUPD FILE:=FUPD BUFFERS=1          ; UPDATE FILE
/END
```

8.11.2 MEMORY PARTITION REQUIREMENTS

The memory requirements for the update utility are:

MEM=(500, 12000, 4000).

8.11.3 LUN ASSIGNMENTS

LUN assignments for LMUPDT are given in table 8-25.

Table 8-25. LMUPDT Logical Unit Assignments

LUN	Description
4	Error messages
5	Control record input
6	Load module input file
7	Load module file to be updated

8.11.4 OPERATION

One control record must be input for each load module to be updated. The control record contains only a base 10 value of the Memory Image Phase (MIP) to be updated. The value is obtained from the DX980 system link load map listing. To find this value, scan the load map listing for the IDT name of the module to be updated. The base 10 value of the MIP number for this overlay is entered, left justified, on the control record as the base ID. After the updates have been made, perform the Initial Program Loading (IPL) procedure to bring the modified load module dictionary into memory. This procedure is described in the DX980 System Operation Guide, Part Number 943004-9701.

8.11.5 SAMPLE INPUT

A typical format for input on LUN 5 is as follows:

```
Control Record for Overlay 1

Control Record for Overlay 2
          •
          •
          •
Control Record for Overlay n

/*                              Terminates Control Record Input
```

8.12 SOURCE MAINTENANCE ROUTINE (SMR)

The DX980 Source Maintenance Routine (SMR) maintains source libraries for large software projects. The SMR keeps a history of changes made to a source library by recording the change level for each record within a program in the library, for each program in the library, and for the library itself. The SMR can access any version (through all change levels) of any program in the library. It can be used for either batch or interactive applications, and has commands that create a new program on the library, modify an existing program in the library, and that delete, list or extract programs in the library. The SMR can also produce an index of all the programs in the library.

8.12.1 STANDARD JCL PROCEDURE

The following listing is a standard procedure for SMR:

```
.# CREATE SMR   ,COMMENT,"SOURCE MAINTENANCE ROUTINE  "
/REPLACE SMR          . SOURCE MAINTENANCE ROUTINE .
/EXEC OBJ=(1,SYSTEM,SMR) MEM=(300,11500,5000) PRTY=(1,15);
/              TIME=-1 MEM:=MEM
/ASSIGN   0 SC    DEVICE:=DMSG                              . ERROR/USER MSG
/ASSIGN   4 SC    DEVICE:=DCON FILE:=FCON BUFFERS=1         . CONTROL
/ASSIGN   6 DUMMY DEVICE:=DLST FILE:=FLST BUFFERS=1         . LISTING
/ASSIGN >15 MT1   DEVICE:=DOLD FILE:=FOLD BUFFERS=1 LINKSEQ . OLD LIBRARY FILE
/ASSIGN >22 DUMMY DEVICE:=DCOM FILE:=FCOM REPLACE:=RCOM;
/           BUFFERS=1 LINKSEQ ACCESS=(ANY,ANY,ANY,ANY);
/           ALLOCATE:=LCOM                                  . COMPILE OUT FILE
/ASSIGN >25 DUMMY DEVICE:=DNEW FILE:=FNEW REPLACE:=RNEW;
/           BUFFERS=1 LINKSEQ ACCESS=(ANY,ANY,ANY,ANY);
/           ALLOCATE:=LNEW                                  . NEW LIBRARY FILE
/ASSIGN >26 DUMMY                                           . JCL UPDAT CON OUT
/ASSIGN >35 DUMMY DEVICE:=DEV35 FILE:=FIL35 BUFFERS=2       . INCLUDE LUN OPT 1
/ASSIGN >45 DUMMY DEVICE:=DEV45 FILE:=FIL45 BUFFERS=2       . INCLUDE LUN OPT 2
/END
```

8.12.2 MEMORY PARTITION REQUIREMENTS

SMR memory requirements are defaulted in the standard procedure to the following:

MEM=(300, 11500, 2000)

8.12.3 LUN ASSIGNMENTS

The logical unit numbers that must be assigned for SMR are provided in table 8-26. In addition to these standard required LUNs, any other LUNs can be assigned to SMR and accessed by the include option.

NOTE

All LUN assignments for SMR, unlike most utilities, are hexadecimal numbers.

8.12.4 OPERATION

SMR executes in either an interactive or a batch manner depending on the type of device used for command input. No special command is required since SMR determines the manner of processing from the attributes of the command LUN. Error processing is the main difference between the two operational types. When executing interactively almost no errors are fatal. The error message is output to the interactive device (in addition to the listing device), and the user can re-enter the command. When batch processing, however, some errors are fatal and some are logical. Logical errors allow recovery. See section 8.12.5 for a description of SMR error codes.

Table 8-26. SMR Logical Unit Assignments

LUN	Description	Comments
0	Operator Console Messages	Used only as explained in discussion of ". #PAUSE" command.
4	Command Input (Logical/fatal error messages if interactive device)	If assigned to an input/output device capable of printing, SMR executes in an interactive manner; if assigned to an input-only device/file, SMR executes in a batch manner. If assigned to DUMMY, the SMR assumes that the first command is ".# UPDATE *COPYLIB". (See discussion of UPDATE mode command.)
6	Listing	Any sequential device/file. SMR uses this LUN to list all submitted commands, for messages, and for listings of programs and indexes of the library as described in discussion of ".#LIST", ".#LSTALL", and ".#INDEX" commands. If LUN 4 is assigned to an interactive device and LUN 6 is assigned to DUMMY, then all messages and the output caused by ".#LIST", ".#LSTALL, and ".# INDEX" commands are written to LUN 4. Therefore, LUN 6 may be assigned to DUMMY unless a hard copy of the processing performed is desired.
15_{16}	Old Library File	Any rewindable sequential device/file. Must be assigned to DUMMY if OLDLIB is not being used.
22_{16}	Compile Output File	Any sequential device/file. This file is the output LUN for all ".#COMPILE" commands and for ".#MODIFY" commands if in EXTRACT mode (as discussed later).
25_{16}	New Library File	Any rewindable sequential device/file. If running SMR in EXTRACT mode (as discussed later), this LUN is not used and may be assigned to DUMMY. Otherwise this file is the output LUN when any changes are made to a program on OLDLIB; all unmodified programs are copied unchanged.

Table 8-26. SMR Logical Unit Assignments (Continued)

LUN	Description	Comments
26_{16}	Scratch File	Used in JCL installation procedure only. LUN 26 must be assigned to DUMMY for general use of SMR.

All modules that comprise a project reside in a source library (OLDLIB). OLDLIB is a sequential data set with rewind capability, either a linked sequential file or a magnetic tape. The directory entry for each program in the library contains such information as module name, revision level, a descriptive title, a general comment field for any other information the user may desire to include, and the language in which the module is written (e.g. PLEXUS, SAP, FORTRAN, etc). All SMR commands that reference existing library programs read from OLDLIB, the input source library. Whenever a library program is to be modified or a library program added, a new source library data set (NEWLIB) is created. OLDLIB is not modified. NEWLIB is a copy of OLDLIB except for the modifications made via SMR commands. NEWLIB has a revision level one greater than OLDLIB and all programs modified have as a latest change level the revision level of NEWLIB.

Source programs can also be copied from OLDLIB to a compile file; that is, a sequential file suitable for input to a compiler or assembler. The compile file is useful when a copy of a program is desired but no permanent changes are being made.

Functionally, SMR can be operated in three modes: Extract, Update and Verify. The mode to be used is determined when the first command is read and cannot be changed after that point. The following paragraphs describe each of the modes.

8.12.4.1 UPDATE MODE. In Update mode, SMR assumes that no output is to be made to the compile file. SMR reads from OLDLIB and writes to NEWLIB. If OLDLIB does not exist (a new library is being built), then the logical unit for OLDLIB (LUN 15_{16}) must be assigned to DUMMY. When in Update mode, SMR requires that the programs in the library be in alphabetical order. Whenever it adds a new program to a library, it first copies any existing programs from OLDLIB that alphabetically precede the new program (sorted by the name assigned to the program when it was created in the library). Furthermore, all SMR commands that reference programs in OLDLIB must reference those programs in alphabetical order. Any programs in OLDLIB that are not specifically referenced in the command are copied to NEWLIB unaltered. If a program is referenced out of order, SMR will search to the end of OLDLIB without finding the program and will not rewind OLDLIB to find the program (rewinding OLDLIB would create multiple copies of programs from OLDLIB in NEWLIB).

8.12.4.2 VERIFY MODE. When operating in Verify mode, SMR assumes that no output is to be made to the compile file. NEWLIB may not be assigned to DUMMY. This mode verifies that a new library (NEWLIB) was generated without I/O errors. SMR commands operate the same in this mode as in Update mode, except that SMR does not write to the output library, NEWLIB. Instead, SMR reads NEWLIB and compares it to verify the contents of the record.

8.12.4.3 EXTRACT MODE. In Extract mode, SMR assumes that no new library is being created. All commands, except .# CREATE and .# DELETE, have meaning as discussed later in this description. References to programs in OLDLIB do not need to be in alphabetical order since SMR rewinds OLDLIB to search for the programs when in Extract mode. However, SMR is more efficient if references are alphabetically ordered.

8.12.4.4 SPECIFYING UPDATE MODE. To use the Update mode, the first command to SMR must be one of the three following commands:

```
.#UPDATE

.#UPDATE *COPYLIB

.#UPDATE *DH*'<rl>'.
```

The first command (.#UPDATE) activates the Update mode as previously described with no modifications.

The second command (.#UPDATE *COPYLIB) instructs SMR to copy OLDLIB without changes to NEWLIB. Specifying this option prevents SMR from incrementing the library revision level, and produces an exact duplicate of OLDLIB in NEWLIB. This option replaces the DXCOPY utility for copying SMR libraries since DXCOPY cannot handle the large size logical records (up to 3600 characters) that SMR writes. If the command input device (LUN 4) is assigned to DUMMY, then SMR assumes that the first input command is .#UPDATE *COPYLIB. If the *COPYLIB option is specified, SMR does not request further input.

The third command (.#UPDATE *DH*'<rl>'.) instructs SMR to Destroy History. The symbol <rl> represents a 2-character code supplied by the user for a revision level indication. The command must appear exactly as it is shown to correctly specify the Destroy History option. When SMR receives this command, it does not write any record onto NEWLIB that was deleted previous to the specified revision level (<rl>). Refer to the discussion of .#MODIFY and .#DELETE commands for an explanation of the deleted records.

8.12.4.5 SPECIFYING VERIFY MODE. To use the Verify mode, the first command to SMR must be one of the three following commands:

```
.#VERIFY

.#VERIFY *COPYLIB

.#VERIFY *DH*'<rl>'.
```

The options in the above commands are the same as those previously described for the .#UPDATE command. To run SMR in the Verify mode, use one of the .#VERIFY commands as the first command. Any options used in this command must be the same options as specified in the .#UPDATE command used to create the NEWLIB that the command is verifying. Following the initial command should be the same sequence of commands that followed the .#UPDATE command (excluding the .#UPDATE command). Similarly, the LUN assignments for OLDLIB and NEWLIB should be exactly the same as those used to create the NEWLIB.

8.12.4.6 SPECIFYING EXTRACT MODE. To specify Extract mode, the first command may be any valid .#xxxx command except those commands used to specify the Update and Verify modes. The following two commands specifically designate the Extract mode:

```
.#EXTRACT

.#EXTRACT *SEQ
```

The term, *SEQ, specifies a sequencing option used with the .#LIST, .#LSTALL, and .#MODIFY commands. Refer to the discussion of the .#LIST command for a description of this option.

8.12.4.7 CREATING A NEW MODULE ON NEWLIB. The command to create a new module on the new library tape is:

```
.#CREATE<name>,<language>, <title>, <comment>.
```

The user specified fields are defined as follows:

- <name> - name to be assigned to the module. This is the name used to reference the module in any other command. Limited to 9 characters.
- <language> - source language of the module. Languages recognized include:
  a) SAL - 960 Assembly Language Source
  b) SAP - 980 Assembly Language Source
  c) OBJECT - Object Decks
  d) FORTRAN

e) FORT - Alternate Identifier for FORTRAN
f) COBOL
g) XPL
h) PLEXUS - Specifies PLEXUS language
i) PL1 - Specifies PL/I language
j) ASM - S/360 Assembly Language Source
k) ALGOL
l) COMMENT - module containing USASCII source records for comment only.

- <title> - a quoted string, limited to 36 characters, that provides a more descriptive title than the 9-character name.
- <comment> - a quoted string, limited to 20 characters, that can hold any additional information such as programmer, author, project name, etc.

When the CREATE command is invoked, all programs in OLDLIB (if any) that alphabetically precede the current program are first copied to NEWLIB. If a program with the specified name already exists in OLDLIB an error is declared. The commands that follow this command normally define the desired contents of the new module. Two different commands can follow a .# CREATE. They are:

1) ./INCLUDE <lun>,<lun>, . . .

   Includes 80 character records (64 if the specified language name is OBJECT) from the specified <lun> until an end-of-file is read. The <lun> is not rewound before input. A .#REWIND command should be used to rewind the <lun> if desired. More than one ./INCLUDE command may be used. The <lun> may be a decimal number, or a hexadecimal number prefixed by a ">" (greater than) or a "#" (pound sign). Any number of <luns> may be specified. If the language name is OBJECT, the contents of the module must be inserted using ./INCLUDE commands since no object records may be encountered in the command input stream.

2) Any record from the command input stream that does not have a '.#' or './' in column 1 and 2 is interpreted as part of the module definition. The definition is completed with either another .# command or an end of file in the command input.

8.12.4.8 MODIFYING A PROGRAM ON OLDLIB. The command to begin modification of a program on OLDLIB is:

```
.#MODIFY   <name>.
```

or

```
.#MODIFY   <name>,<language>,<title>,<comment>.
```

The <name> field represents the name of the program to be modified. If the program is not defined on OLDLIB, an error condition exists and the command is terminated. The <language>, <title>, and <comment> fields are optional. They need only be supplied to change those fields on the program header record that SMR maintains for the program and that are displayed for each program when a .#INDEX command is processed. If any one of these three fields is specified, then the field (or fields) preceding it must also be specified. For example, to change the title, the .#MODIFY command must specify the name, the language and then the title.

If in Update mode, then the resulting program image is written on NEWLIB. If in Extract mode, the resulting program image is written to the compile file. When the .#MODIFY command is invoked in Update mode, all programs in OLDLIB that alphabetically precede the specified program are first copied to NEWLIB. More commands are then input to define the new program contents. Four different commands can follow a .#MODIFY command. These are:

1) @NNNNN.

   This command is the Insert After command. This command copies from OLDLIB all records in the program up to and including record number NNNNN. The number, NNNNN, should be a decimal number (leading zeroes not required). (The .# LIST command description discusses sequenced listings of a program in OLDLIB. The sequence numbers given by such a listing are the appropriate numbers to use for NNNNN.) When the records have been copied (to NEWLIB if in Update mode or to the Compile File if in Extract mode) then a new command is read to continue defining the new program contents.

2) @NNNNN, MMMMM.

   This command is the Delete Records command. This command performs the following functions:

   - Copies all records of the program from OLDLIB up to and including record number NNNNN-1.
   - If in Extract mode, it reads and discards all records up to and including record number MMMMM from OLDLIB.
   - If in Update mode, it reads and marks as deleted (at the revision level of NEWLIB) all records up to and including record number MMMMM and then writes the records to NEWLIB. SMR does not delete these records; it only marks them 'deleted'. In this way SMR can reproduce the image of a program in the library exactly as it appeared at any previous revision.

3) ./INCLUDE <lun>,<lun>, . . .

   This command operates the same as explained in the discussion of the . #CREATE command.

4) Module Definition.

   Any record from the command input stream that does not have an "@" in column 1, or a ".#" or "./" in columns 1 and 2 is interpreted as part of the module definition. It is inserted into the compile file or NEWLIB immediately. The module definition is completed when either a ".#" command or an End-of-file is encountered in the command input stream.

If an "@NNNNN." command is encountered after record NNNNN+1 has been read from OLDLIB, then an error message is output and the command is ignored. Similiarly, if a "@NNNNN, MMMMM." command is encountered after record NNNNN has been read from OLDLIB, then an error message is given and the command is ignored.

If in Extract mode and the "*SEQ" option was specified in the ".#EXTRACT" command, then columns 73 through 80 of the record are modified before being placed in the Compile File. The modification is as follows:

1) If the record was not previously in OLDLIB (inserted into the command stream or via a "./INCLUDE" command), then columns 73 through 80 are blanked.

2) If the record was in OLDLIB previously, then columns 73 and 76 are blanked, columns 74 and 75 contain the 2-character revision level that represents the revision level of the library when the record was added, and columns 76 through 80 contain a 4-digit (decimal) sequence number.

8.12.4.9 COMPILE. The format of this command is:

$$.\#\text{COMPILE} \quad \left\{ \begin{array}{c} <\text{name}> \\ * \\ /* \end{array} \right\} \quad .$$

This command moves records to the compile file when no editing is required. Only one of the three options may be specified. If not in Extract mode, then use of the "/*" option causes the command to be ignored and use of either <name> or * options produces a logical error message output and the command is then ignored.

The user supplied parameters are interpreted as follows:

- <name> - Name of a module to be copied from OLDLIB to the compile file.

- * - Records are to be copied directly from the command input into the compile file until a ".#" command or an end of file is encountered.
- /* - Write an end of file in the compile file. An end of file is automatically written on the compile file when SMR terminates. This command permits the user to build compile files with more than one end of file.

To obtain a copy of a module in OLDLIB as it was at a revision previous to the current revision of the module, a second option may be added to the <name> option as follows:

```
.# COMPILE <name>, *'<rl>'.
```

The notation <name> is the name of the module and <rl> represents a 2-character revision code for the desired revision level. The allowed revision character codes with interpretation are:

**	= 0
*A	= 1
⋮	⋮
*Z	= 26
AA	= 27
⋮	⋮
AZ	= 52
⋮	⋮
ZZ	= 702

The * and quote marks are required to correctly specify the revision level option.

8.12.4.10 DELETE. The command for designating modules as deleted from the library is:

```
.#DELETE <name>,<name>, ...
```

If in Extract mode, this command is ignored. Otherwise, each module named is located in OLDLIB and then the header record is marked "deleted". After marking the header record, SMR marks all previously non-deleted records as being "deleted" at the revision level of NEWLIB. All records (including the header record) are copied to NEWLIB. This "deletion" prevents the module name from appearing when an Index (as discussed later) is done. All subsequent references to the module name result in an error unless the command that references the module name also contains the necessary options to specify that the revision level desired is previous to the "deletion" of this module. However, if a .# CREATE command is later used and the name given matches the name of a previously deleted module, then the header record and module name are reactivated, all the previously deleted records are copied, unaltered, to NEWLIB, and then the next command is read to start defining the module's contents. The source language of the new module must be OBJECT if and only if the source language of the old module was OBJECT.

8.12.4.11 LIST. The command for listing a library module is:

```
.#LIST  < name>,< name>, . . .
```

This command lists the source records in the modules specified. The modules referenced must always be on OLDLIB. If in Extract mode and the "*SEQ" option was given on the ".#EXTRACT" command, then columns 73 and 76 are blanked, columns 74 and 75 have the 2-character code for the revision level of the module when the record was created, and columns 77 through 80 contain a 4-digit (decimal) sequence number.

To get listings of certain modules as they appeared at a revision previous to their latest revision, the revision level may be specified as in the following example:

```
.#LIST   NAME1, *'*A', NAME2, NAME3, *'BZ', NAME4, *'**'.
```

This example generates a listing of:

1) Module NAME1 as it appeared at revision *A
2) Module NAME2 as it appears at the current revision
3) Module NAME3 at revision BZ
4) Module NAME4 at revision ** (when the library was first created.

8.12.4.12 LIST ALL. The command for listing all modules in OLDLIB (except those with source language OBJECT) is:

```
.#LSTALL.
```

This command is invalid unless in Extract mode. Every non-OBJECT module in OLDLIB will be listed on LUN 6. However, if LUN 4 is assigned to an interactive device and LUN 6 is assigned to DUMMY, the listing appears on LUN 4. The *SEQ option on the .# EXTRACT command has the same effect when the .# LSTALL command is used as for the . #LIST command.

8.12.4.13 REWIND LUN. The command to rewind one or more logical units is:

```
.#REWIND<lun>,<lun>, ---.
```

The notation may be a decimal number, or a hexadecimal number prefixed by a '>' or a '#'. This command can be used to rewind a< lun> referenced in a ./INCLUDE command.

8.12.4.14 INDEX. The command for listing a library index is:

```
.#INDEX
```

The .#INDEX command lists on LUN 6 a catalog of all the modules on the library tape. However, if LUN 4 is assigned to an interactive device and LUN 6 is assigned to DUMMY, the listing appears on LUN 4. If the .#INDEX command is input after records are written to NEWLIB, then the new library will be indexed when the .#ENDALL command is entered to terminate SMR. Two possible options may be specified with the .#INDEX command (but only one at a time). The first:

```
.#INDEX *'<rl>'
```

causes SMR to produce an index of the library as it appeared at the revision specified by the 2-character revision code represented in the example by <rl>.

The second:

```
.#INDEX *TH      (Note the lack of quotes)
```

causes SMR to produce a listing of the library header records for all previous revisions of the library (beginning with the most recent and going backwards). This option is useful if the library header information has been changed in previous revisions via the .#TAPE command.

8.12.4.15 TAPE. The command for creating a library header record is:

```
.#TAPE '<title>', '<part number>', '<date>', *'<revision level>'.
```

The user specified fields are defined as follows:

- <title> - a quoted string, limited to 26 characters, that is the name of the source library.
- <part number> - a quoted string, limited to 12 characters, to be used to document the part number the source library was released under.
- <date> - a quoted string, limited to eight characters, used to document the date the source library was generated.
- <revision level> - a quoted 2-character string indicating the revision level of NEWLIB.. If not specified, NEWLIB will have a revision level which will be the next sequential alphabet character; i.e. *B if OLDLIB was *A.

The .#TAPE command is optional and need only be specified the first time that a new library is generated or to change the title, part number, or date on the library header record for the next revision of the library. This command, if used, may be preceded only by .#PAUSE, .#INDEX, .#UPDATE, and .#VERIFY commands.

8.12.4.16 PAUSE. The command to write a message to the operator's console is:

.#PAUSE'<message>'.

This command writes the quoted string on LUN 0. Processing is continued when a response (carriage return or end of file) is entered on LUN 0.

8.12.4.17 ENDALL. The command for terminating SMR is:

.#ENDALL

An end of file on LUN 4 also terminates SMR. If a .#INDEX of NEWLIB has been specified, it is processed when the copy from OLDLIB to NEWLIB is completed.

8.12.5 ERROR CODES

Error codes for SMR are output to the listing device (LUN 6) and in the interactive mode, to the interactive command device (LUN 4) as well. In the interactive mode, only two errors are considered fatal. Unlimited user retry is normally allowed. In the batch mode, errors are classified as fatal, (i.e. abortive to the job) or logical (i.e. program recoverable). Logical errors for batch mode include invalid LUN, missing arguments, unmatched quotation mark, etc. In these cases the field is either skipped, assumed to be a blank, or assumed to be terminated by quotation mark in column 81.

The two fatal errors for both SMR batch and interactive modes are:

- Specifying Update or Verify mode with NEWLIB assigned to DUMMY.
- Specifying (or defaulting to) Extract mode with OLDLIB assigned to DUMMY.

The two fatal errors for SMR batch mode are:

- Referencing a file name that does not exist in the old source library
- Exceeding 100 logical errors.

8.12.6 SAMPLE OUTPUT

The following paragraphs contain examples of the use of SMR.

8.12.6.1 CREATING A NEW LIBRARY TAPE IN BATCH MODE. The JOB and RUN commands for this execution are:

```
/JOB S SYSTEM

/RUN SMR DOLD=DUMMY    DNEW=MT1    DCON=CR1;

/DMSG=SC    DLST=LP1    DCOM=DUMMY.
```

The control cards and statements defining the modules are shown in figure 8-13. Figure 8-14 shows the index of NEWLIB at the termination of SMR.

```
SOURCE MAINTENANCE ROUTINE


.#UPDATE
.#TAPE    'SMR EXAMPLES','123456-7800','04/10/75','*'**'
.#CREATE EFF330,FORTRAN,'COMPUTE DS330 DISC EFFICIENCY'.
      WRITE(6,300)
300   FORMAT(1H1,3X,13HRECORDS/TRACK,4X,14HSECTORS/RECORD,
     1  2X,12HWORDS/RECORD,2X,18HUSEFUL WORDS/TRACK,
     2  2X,10HEFFICIENCY)
      DO 100 ISECR = 1,88
      WC = ((ISECR*1221.0-677.0)*32.0)/(34.0*16.0)
      IWRDR = WC/32
      IWRDR = IWRDR*32
      IRECT = 88/ISECR
      IUSWT  = IRECT*IWRDR
      EFF= (IUSWT*100.0)/6720.0
100   WRITE(6,200)IRECT,ISECR,IWRDR,IUSWT,EFF
200   FORMAT(1H ,/,9X,I5,10X,I5,10X,I5,13X,I5,9X,F8.2)
       STOP
       END
.#CREATE SMREXAMPLE,COMMENT,'SMR EXAMPLE # 2 ','NO COMMENT'
         THIS IS ORIGINAL RECORD # 1
         THIS IS ORIGINAL RECORD # 2
         THIS IS ORIGINAL RECORD # 3
         THIS IS ORIGINAL RECORD # 4
         THIS IS ORIGINAL RECORD # 5
         THIS IS ORIGINAL RECORD # 6
         THIS IS ORIGINAL RECORD # 7
         THIS IS ORIGINAL RECORD # 8
         THIS IS ORIGINAL RECORD # 9
         THIS IS ORIGINAL RECORD # 10
.#INDEX.
FLAG SET TO INDEX NEW LIBRARY
.#ENDALL.
```

Figure 8-13. SMR Batch Input to Create a New Library

```
PART NO.=123456-7890  REV=** DATE=04/10/75 TITLE=SMR EXAMPLES


NAME       REV LANG     TITLE                                    COMMENT

EFF330     ** FORTRAN  COMPUTE DS330 DISC EFFICIENCY
SMREXAMPL ** COMMENT  SMR EXAMPIE # 2                           NO COMMENT
2 FILES ON NEW LIBRARY
```

Figure 8-14. Index of NEWLIB

8.12.6.2 GETTING A SEQUENCED LISTING OF A MODULE. Figure 8-15 shows the commands entered to get a sequenced listing of a module in OLDLIB and the resulting list. The JCL to run SMR for this example is:

```
/JOB S SYSTEM
/RUN SMR DNEW=DUMMY    DOLD=MT1
/DCON=SC    DLST=LP1    DCOM=DUMMY
```

```
PART NO.=123456-7890  REV=** DATE=04/10/75 TITLE=SMR EXAMPLES

.#EXTRACT *SEQ
.#LIST SMREXAMPLE .




NAME       REV LANG      TITLE                                         COMMENT

SMREXAMPL ** CUMMENT  SMR EXAMPLE # 2                                  NO COMMENT
           THIS IS ORIGINAL RECORD # 1                                                ** 0001
           THIS IS ORIGINAL RECORD # 2                                                ** 0002
           THIS IS ORIGINAL RECORD # 3                                                ** 0003
           THIS IS ORIGINAL RECORD # 4                                                ** 0004
           THIS IS ORIGINAL RECORD # 5                                                ** 0005
           THIS IS ORIGINAL RECORD # 6                                                ** 0006
           THIS IS ORIGINAL RECORD # 7                                                ** 0007
           THIS IS ORIGINAL RECORD # 8                                                ** 0008
           THIS IS ORIGINAL RECORD # 9                                                ** 0009
           THIS IS ORIGINAL RECORD # 10                                               ** 0010
.#ENDALL
```

Figure 8-15. Sequenced Listing Commands and Listing

8.12.6.3 ADDING A MODULE TO GET A NEW LIBRARY. Figure 8-16 shows the command input stream used to add two modules to the library. Figure 8-17 shows an index of the new library. Note that the modules have been inserted in alphabetical order.

8.12.6.4 MODIFYING A MODULE. Figure 8-18 shows the command input stream to modify a module in the library. Figure 8-19 shows an Index of the new library generated.

8.12.6.5 THE SEQUENCED LISTING OF THE MODIFIED MODULE. Figure 8-20 shows the command input stream used to get a sequenced listing of the module modified in example 8.12.6.4, as well as the sequenced listing of that module.

```
PART NO.=123456-7890  REV=** DATE=04/10/75 TITLE=SMR EXAMPLES

.*UPDATE .
.*CREATE EDITOR,PLEXUS,'CHARACTER EDITOR PROGRAM'.
         /*  SC SS SB   SA      */
 DECLARE EDITOR PROCEDURE CHARACTER,
         (INSTRN,OTSTRN,NWSTRN) CHARACTER(80);
         INSTRN = '  ';
         DO WHILE SUBSTR(INSTRN,1,2) A= '$$';
           INSTRN = INPUT;
           NWSTRN = EDITOR(INSTRN,'JLD','JLX');
           OUTPUT = NWSTRN;
           END;
 EDITOR: PROCEDURE (INSTRING,VICTIM,VICTOR);
         DECLARE (INSTRING,RESULT) CHARACTER(80),
                 (VICTIM,VICTOR) CHARACTER,
                 I FIXED(16);
         RESULT = INSTRING;
         I = 0;
         DO WHILE I<=(LENGTH(INSTRING) - LENGTH(VICTOR)) &
                  I<=(LENGTH(INSTRING) - LENGTH(VICTIM)) &
                  (I < 100);
               IF VICTIM = SUBSTR(INSTRING,I,I+LENGTH(VICTIM)) THEN
                 RESULT = SUBSTR(INSTRING,0,I)||VICTOR||
                          SUBSTR(INSTRING,I+LENGTH(VICTIM));
               I = I + 1;
               END;
         RETURN RESULT;
 END EDITOR;
EOF
.*CREATE SEEK,SAP,'DS330 INDEPENDENT SEEK TEST',
       IDT  ISTEST
       HED  DS330 INDEPENTENT SEEK TEST
SVC    OPD  >C380,3
START  EQU  $
CK1    EQU  $
       TMBZ BUSY,PRB1
       BRU  CK2
       BRU  RBUF1
CK2    EQU  $
       TMBZ BUSY,PRB2
       BRU  CK1
       BRU  RBUF2
RBUF1  SMBZ WRITE,PRB1+1
       DLD  PRB1+4
       DAD  INC
       DST  PRB1+4
       *LDM *PRB1
       BRU  GO
RBUF2  SMBZ WRITE,PRB2+1
       DLD  PRB2+4
       DAD  INC
       DST  PRB2+4
       *LDM *PRB2
GO     SVC  0
       BRU  CK1
PRB1   DATA 5,>8017,32,BUF1,0,0
WRITE  EQU  15
BUSY   EQU  0
INC    DATA 0,88
PRB2   DATA 6,>8017,32,BUF2,0,88
BUF1   BSS  32
BUF2   BSS  32
       END  START
.*INDEX .
FLAG SET TO INDEX NEW LIBRARY
.*ENDALL
```

Figure 8-16. Input Stream to Add Two Modules

```
PART NO.=123456-789Ø  REV=*A DATE=Ø4/1Ø/75 TITLE=SMR EXAMPLES


NAME        REV LANG      TITLE                                                 COMMENT

EDITOR      *A  PLEXUS    CHARACTER EDITOR PROGRAM
EFF33Ø      **  FORTRAN   COMPUTE DS33Ø DISC EFFICIENCY
SEEK        *A  SAP       DS33Ø INDEPENDENT SEEK TEST
SMREXAMPL   **  COMMENT   SMR EXAMPLE # 2                                       NO COMMENT
4 FILES ON NEW LIBRARY
```

Figure 8-17. Index of NEWLIB with Added Modules

```
PART NO.=123456-789Ø  REV=*A DATE=Ø4/1Ø/75 TITLE=SMR EXAMPLES
.#UPDATE
.#MODIFY SMREXAMPLE
         THIS IS NEW RECORD # 1
#5
         THIS IS NEW RECORD # 5-A
         THIS IS NEW RECORD # 5-B
#7,9
         THIS IS NEW RECORD 7-A
#>7FFF
         THIS IS THE NEW LAST RECORD
.#INDEX
FLAG SET TO INDEX NEW LIBRARY
.#ENDALL
```

Figure 8-18. Input Stream to Modify a Module

```
PART NO.=123456-789Ø  REV=*B DATE=Ø4/1Ø/75 TITLE=SMR EXAMPLES


NAME        REV LANG      TITLE                                                 COMMENT

EDITOR      *A  PLEXUS    CHARACTER EDITOR PROGRAM
EFF33Ø      **  FORTRAN   COMPUTE DS33Ø DISC EFFICIENCY
SEEK        *A  SAP       DS33Ø INDEPENDENT SEEK TEST
SMREXAMPL   *B  COMMENT   SMR EXAMPLE # 2                                       NO COMMENT
4 FILES ON NEW LIBRARY
```

Figure 8-19. Index of Modified Module

8.12.6.6 DELETING A MODULE. Figure 8-21 shows the command input stream to delete a module. Figure 8-22 shows an index of the new library generated. This index was generated in a separate run of SMR.

```
PART NO.=123456-7890  REV=*B DATE=04/10/75 TITLE=SMR EXAMPLES

.*EXTRACT *SEQ
.*LIST SMREXAMPLE
```

```
NAME       REV LANG      TITLE                                      COMMENT

SMREXAMPL *B COMMENT  SMR EXAMPLE # 2                               NO COMMENT
           THIS IS NEW RECORD # 1                                                *B 0001
           THIS IS ORIGINAL RECORD # 1                                           ** 0002
           THIS IS ORIGINAL RECORD # 2                                           ** 0003
           THIS IS ORIGINAL RECORD # 3                                           ** 0004
           THIS IS ORIGINAL RECORD # 4                                           ** 0005
           THIS IS ORIGINAL RECORD # 5                                           ** 0006
           THIS IS NEW RECORD # 5-A                                              *B 0007
           THIS IS NEW RECORD # 5-B                                              *B 0008
           THIS IS ORIGINAL RECORD # 6                                           ** 0009
           THIS IS NEW RECORD 7-A                                                *B 0010
           THIS IS ORIGINAL RECORD # 10                                          ** 0011
           THIS IS THE NEW LAST RECORD                                           *B 0012
.*ENDALL
```

Figure 8-20. Sequenced Listing of Modified Module - Input Commands and Listing

```
PART NO.=123456-7890  REV=*B DATE=04/10/75 TITLE=SMR EXAMPLES

.*UPDATE
.*DELETE EDITOR
.*ENDALL
```

Figure 8-21. Input Stream to Delete a Module

```
PART NO.=123456-7890  REV=*C DATE=04/10/75 TITLE=SMR EXAMPLES


NAME        REV LANG      TITLE                                   COMMENT

EFF330      ** FORTRAN  COMPUTE DS330 DISC EFFICIENCY
SEEK        *A SAP      DS330 INDEPENDENT SEEK TEST
SMREXAMPL  *B COMMENT  SMR EXAMPLE # 2                            NO COMMENT
3 FILES ON NEW LIBRARY
```

Figure 8-22. Index of NEWLIB without Deleted Module

8.13 LINKABLE PARTS FILE BUILD UTILITY (LPFBLD)

The LPFBLD utility maintains a key indexed file of 980 object records. Each object deck in the file has a key equal to the IDT name with all of the object records for that program sequentially linked to the key. LPFBLD can be used to modify, extract, or delete members of a linkable parts file. LPFBLD creates a key of the program if added to the file and replaces the old object if the IDT name already exists as a key.

LPFBLD is useful for building an object file for input to the DX980 link editor (DXOLE). To retrieve an object deck from the key indexed file, use the DXOLE include with key option (i.e. INCLUDE 20(IDTNAM)). Using DXOLE and LPFBLD together in this manner allows easy manipulation of object files and easy retrieval for link editor input.

8.13.1 STANDARD JCL PROCEDURE

The following listings are standard procedures for LPFBLD. The first is for a single step job executing LPFBLD. The second is for a two step job performing an assembly of the module and including this assembled module in the library file.

```
.# CREATE LPFBLD,COMMENT,"UPDATE LINKABLE PARTS FILE  "
/REPLACE LPFBLD       . UPDATE LINKABLE PARTS FILE .
/EXEC OBJ=(1,SYSTEM,LPFBLD) MEM=(300,3300,1000) PRTY=(1,2);
/           TIME=-1 OBJ:=OBJ MEM:=MEM
/ASSIGN    0 DUMMY DEVICE:=DCON                                . CONTROL
/ASSIGN    5 DISC1 DEVICE:=DOBJ FILE=(USER01,ASMOUT) FILE:=FOBJ;
/           BUFFERS=1                                          . OBJECT INPUT
/ASSIGN    6 SC     DEVICE:=DLST                               . LISTING
/ASSIGN    7 DUMMY DEVICE:=DEXT FILE:=FEXT REPLACE:=REXT;
/           BUFFERS=1 LINKSEQ ACCESS=(ANY,CREAT,CREAT,CREAT);
/           ACCESS:=CEXT ALLOCATE=(1,0,128,20) ALLOCATE:=LEXT . EXTRACT FILE
/ASSIGN    9 DISC1 DEVICE:=DUPD FILE:=FUPD REPLACE:=RUPD;
/           BUFFERS=2 INDEXED ACCESS=(CREAT,CREAT,CREAT,CREAT);
/           ACCESS:=CUPD ALLOCATE=(1,0,256,10) ALLOCATE:=LUPD;
/           KEYLEN=6                                           . UPDATE FILE
/END
```

```
.# CREATE ASMUP ,COMMENT,"ASSEMBLE MOD AND UPDATE LPF,"
/REPLACE ASMUP       . ASSEMBLE MODULE AND UPDATE LPF ,
/EXEC OBJ=(1,SYSTEM,ASMBLR) MEM=(300,6000,2000) PRTY=(1,2);
/          TIME=-1 MEM:=MEMA
/ASSIGN  0 DUMMY DEVICE:=DMSG                              ; SYSTEM MESSAGE
/ASSIGN  4 DUMMY DEVICE:=DCON                              , CONTROL/MESSAGE
/ASSIGN  5 DISC1 DEVICE:=DSRC FILE:=FSRC BUFFERS=1         , SOURCE INPUT
/ASSIGN  6 SC    DEVICE:=DLSTA FILE:=FLST BUFFERS=1        . SOURCE LIST/ERROR
/ASSIGN  7 DISC1 FILE=(TEMP,OBJECT) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,64,20) . OBJECT OUTPUT
/ASSIGN 16 DISC1 FILE=(TEMP,SCRL) NEW DELETE BUFFERS=1;
/          LINKSEQ ACCESS=(ANY,ANY,ANY,ANY);
/          ALLOCATE=(10,300,256,30)                        . SOURCE SCRATCH
/EXEC OBJ=(1,SYSTEM,LPFBLD) MEM=(300,3300,1000) PRTY=(1,2);
/          TIME=-1 MEM:=MEMU
/ASSIGN  0 DUMMY DEVICE:=DCON                              ; CONTROL
/ASSIGN  5 DISC1 FILE=(TEMP,OBJECT) BUFFERS=1              , OBJECT INPUT
/ASSIGN  6 SC    DEVICE:=DLSTU FILE:=FLST BUFFERS=1        , LISTING
/ASSIGN  9 DISC1 DEVICE:=DUPD FILE:=FUPD REPLACE:=RUPD;    .
/          BUFFERS=2 INDEXED ACCESS=(CREAT,CREAT,CREAT,CREAT);
/          ACCESS:=CUPD ALLOCATE:=LUPD KEYLEN=6            . UPDATE FILE
/END
```

8.13.2 MEMORY PARTITION REQUIREMENTS

The memory allocation parameter for LPFBLD should be:

MEM=(300,3300,1000).

The job extension parameter is dependent upon the blocking factors and the number of buffers assigned for the LUNs, but 1000 words is adequate for most files.

8.13.3 LUN ASSIGNMENTS

The logical unit assignments for LPFBLD are given in table 8-27. Figure 8-23 illustrates the linkable parts file (LPF) built on LUN 9.

8.13.4 OPERATION

LPFBLD first reads a control record from LUN 0. The control record determines the mode of execution. Mode 2 is the default LPFBLD mode of execution.

MODE 1: The control record of 'CRb̸b̸' implies all of the old keys and data are deleted from the linkable parts file. Then processing continues the same as in Mode 2.

MODE 2: The control record of 'MODb̸' or an end of file (from DUMMY) modifies only those modules input from LUN 5 either by replacing the old object, or by creating a new key and inserting the new object. The new object is reformatted and output to LUN 9.

MODE 3: The control record of 'EXb̸b̸' implies that the object for members of the linkable parts file assigned to LUN 9 is to be output to LUN 7 as a sequential file. The names of the modules to extract should start in column 5 of the control command and be separated by commas. The first blank column terminates the scan for member names. A control card cannot be continued but multiple control commands may be entered. An end of file or a control record other than 'EXb̸b̸' terminates LPFBLD in the EXTRACT mode. For example,

```
EX  A,B,C
EX  D
/*
```

would extract modules A, B, C, and D from LUN 9 and output the object to LUN 7. An end of file is written at the end of LUN 7.

MODE 4: The control record of 'DELb̸' implies that the object for the members of the linkable parts file that are named on the control card is to be deleted from LUN 9. The syntax of the DELETE (DEL) command is the same as for the EXTRACT command in mode 3. Multiple DEL commands may be entered if necessary.

Table 8-27. LPFBLD Logical Unit Assignments

LUN	Description	Comments
0	Control Input	Control records are read to select the LPFBLD mode of execution.
5	Object Input	One or more object modules, input terminated by an EOF. This LUN is referenced for the 'CR' and 'MOD' modes. The LUN is rewound before any input is read.
6	Listing	Listing of IDT names of all object modules entered through LUN 5 and error messages.
7	Extract Mode Output	Output LUN for the 'EX' mode. This LUN is rewound before any object is written and is terminated with an end of file. This LUN is processed as a link sequential file or a device that supports binary output.
9	Update File	Key indexed linkable parts file built with a key length of 6 characters and a logical record length of 80 characters.

8.13.5 ERROR CODES

- "INDEX FILE FULL"

  This message indicates that the linkable parts file is full and no records can be added. The user must define a new file with a larger maximum allocation and then recreate the file or use the DXCOPY utility to copy the old file into the new file.

- "INVALID MODE"

  The control record was invalid. Columns 1 through 4 of the control record must be 'CRb̸b̸', 'MODb̸', 'EXb̸b̸', or 'DELb̸' unless an end of file is input.

  "<name> IS NOT IN FILE"

  The number name specified on the DEL or EXT control record does not exist on LUN 9. The processing does not terminate but just skips over the invalid name.

Record Key	Data Associated with Key (OBJECT Record)	
PGM01b̸	1700	PGM01
	1702	
	⋮	
	1706	
TABLEb̸	1700	TABLE
	⋮	
	1706	

Figure 8-23. Format of Linkable Parts File Built by LPFBLD

8.14 BUILD EDIT FILE UTILITY (BLDEDT)

The build edit file utility, BLDEDT, constructs keyed indexed files for editing under the Interactive File Editor (IFE) of the Interactive Terminal Subsystem. Reference the Interactive File Editor description in Section VII of this manual for a discussion of the uses for BLDEDT.

8.14.1 STANDARD JCL PROCEDURE

The following listing is a standard procedure for BLDEDT:

```
.# CREATE BLDEDT,COMMENT,"BUILD EDIT FILE          "
/REPLACE BLDEDT     . BUILD EDIT FILE .
/EXEC OBJ=(1,SYSTEM,BLDEDT) MEM=(3ØØ,55Ø,2ØØØ) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN 1Ø DISC1 DEVICE:=DIN FILE:=FIN BUFFERS=1              . SOURCE INPUT
/ASSIGN 2Ø DISC1 DEVICE:=DOUT FILE:=FOUT REPLACE BUFFERS=2;
/          BUFFERS:=BOUT INDEXED ACCESS=(ANY,ANY,ANY,ANY);
/          ACCESS:=COUT ALLOCATE=(1,Ø,256,1ØØ) ALLOCATE:=LOUT;
/          KEYLEN=2                                           . SOURCE OUT FILE
/END
```

8.14.2 MEMORY PARTITION REQUIREMENTS

The following are the memory requirements for BLDEDT utility:

<stksiz> = 300 words

<jarea> = 550 words

<jearea> = 2000 words

8.14.3 LUN ASSIGNMENTS

The LUN assignments for BLDEDT are given in table 8-28.

8.14.4 OPERATION

The Build Edit File (BLDEDT) utility is a 2-task program operating under DX980. BLDEDT accepts input from a sequential access source (either device or file) and builds the key indexed edit file such that the 1-word keys are record numbers and the data is the source text. The first record has a key of "1", the second "2", etc.

Table 8-28. BLDEDT Logical Unit Assignments

LUN	Description	Comment
10	Source Input	Sequential device/file
20	Source Output	Key indexed file with key length of 2 characters.

8.14.5 ERROR CODES

BLDEDT does not have any unique error codes. The system detects any abnormal condition. Refer to Appendix A for the error codes.

8.15 LIST EDIT FILES UTILITY (LSTEDT)

The List Edit Files Utility (LSTEDT) lists an entire IFE File on a formattable device. Although the List Record edit command of the Interactive File Editor lists selected records in an edit file, this utility program lists the entire edit file. This listing can be valuable for subsequent editing sessions since the record number for each record is displayed along with the record. The format of the listing is a four digit record number followed by a separator blank and the data record.

8.15.1 STANDARD JCL PROCEDURE

The following listing is a standard procedure for LSTEDT:

```
.# CREATE LSTEDT,COMMENT,"LIST EDIT FILE                "
/REPLACE LSTEDT      . LIST EDIT FILE .
/EXEC OBJ=(1,SYSTEM,LSTEDT) MEM=(300,1700,650) PRTY=(1,15);
/           TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN 10 SC     DEVICE:=DLST                                 . SOURCE OUTPUT
/ASSIGN 20 DISC1 DEVICE:=DIN FILE:=FIN BUFFERS=1               . SOURCE INPUT FILE
/END
```

8.15.2 MEMORY PARTITION REQUIREMENTS

The following are the memory requirements for the LSTEDT utility:

<stksiz> = 300 words

<jarea> = 1700

<jearea> = 650 words

8.15.3 LUN ASSIGNMENTS

The required LUN assignments for the LSTEDT utility are given in table 8-29.

Table 8-29. LSTEDT Logical Unit Assignments

LUN	Description	Comment
10	Source Output	Formattable device
20	Source Input	Key indexed file with key length of 2 characters.

8.15.4 OPERATION

The LSTEDT utility is a single-task program that operates under DX980. No command input is required. The key indexed file assigned to LUN 20 is listed to the formattable device assigned to LUN 10.

8.15.5 ERROR CODES

LSTEDT does not have any unique error codes. The system detects abnormal conditions. Refer to Appendix A for error codes.

8.16 CREATE, DELETE, OR REPLACE FILE (FILMGR)

The File Manager Utility (FILMGR) creates, deletes or replaces a file. If the user specifies replacement and a file with the same name presently exists, FILMGR deletes the old file and creates a new file.

8.16.1 STANDARD JCL PROCEDURE

The following listing is a standard procedure for FILMGR:

```
.# CREATE FILMGR,COMMENT,"FILE CREATE/DELETE CAPABILIT"
/REPLACE FILMGR     . FILE CREATE/DELETE CAPABILITY .
/EXEC OBJ=(1,SYSTEM,DXCOPY) MEM=(300,2650,850) PRTY=(1,15);
/           TIME=-1 PRTY:=PRI TIME:=TIM
/ASSIGN  1 DISC1 DEVICE:=DISC FILE:=FILE NEW:=NEW REPLACE:=REP;
/           DELETE:=DEL BUFFERS=1 LINKSEQ:=LIN RELREC:=REL;   . FILE TO BE
/           INDEXED:=IND ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=ACC;. CREATED/DELETED
/           ALLOCATE:=ALL KEYLEN=6 KEYLEN:=KEY LRECL=64;
/           LRECL:=LRE
/ASSIGN  5 DUMMY
/ASSIGN  6 DUMMY
/ASSIGN  7 DUMMY
/ASSIGN  8 DUMMY
/END
```

8.16.2 MEMORY PARTITION REQUIREMENTS

The following memory parameters are required for the FILMGR utility:

<stksiz> = 300

<jarea> = 2650

<jearea>= 850

8.16.3 LUN ASSIGNMENTS

The required LUN assignments for the FILMGR utility are given in table 8-30.

Table 8-30. FILMGR Logical Unit Assignments

LUN	Description	Comment
1	File to be created, deleted or replaced	Any file type
5	Not used	Assign to DUMMY
6	Not used	Assign to DUMMY
7	Not used	Assign to DUMMY
8	Not used	Assign to DUMMY

8.16.4 OPERATION

The FILMGR utility executes the load module, DXCOPY. Assignment of LUN 1 to the file being managed accomplishes the action specified in the job control when processed by DX980 job management. Parameters in the Run command indicate if the file is to be created, deleted or replaced. Deleting a file need only specify the file name and password (if the file is password protected against deleting). Both the NEW and REPLACE specifications require further statements to allocate new disc space, define the access code and state the type of file.

8.16.4.1 DELETING FILES. The Run command required to delete a file is of the following form:

```
//RUN FILMGR   DISC=<devnam> FILE=(<fileid>,<filnam>,<pswd>) DEL
```

Section II of this manual (Job Control Language) describes the variable parameters. The password (<pswd>) may be omitted if the file is not password protected against deleting.

8.16.4.2 CREATING A LINKED SEQUENTIAL FILE. The Run command necessary to create a new linked sequential file is of the form:

```
//RUN FILMGR  DISC=<devnam> FILE=(<fileid>,<filnam>,<pswd>) NEW LIN;
  ACC=(<integ>,<integ>,<integ>,<integ>) ALL=(<itrks>,<trknum>,<prwrds>,<mtrks>)
```

Section II of this manual describes the parameters in this statement. A sample Run command to create a 50-track, linked sequential file that anyone can access appears as follows:

```
//RUN  FILMGR  DISC=DISC1  FILE=(USER01,FILE1)  NEW  LIN;
  ACC=(ANY,ANY,ANY,ANY)  ALL=(50,0,256,60).
```

The file is created on DISC1, has no password, has a physical record length of 256 words and a maximum allocation of 60 tracks. The search for available space starts with track zero.

8.16.4.3 CREATING A RELATIVE RECORD FILE. The RUN command used to create a relative record file appears as:

```
//RUN  FILMGR  DISC=<devnam> FILE =(<fileid>,<filnam>,<pswd>) NEW  REL;
  ACC=(<integ>,<integ>,<integ>,<integ>);
  ALL=(<itrks>,<trknum>,<prwrds>,<mtrks>)  LRECL=<lrchar>
```

The allocation of a relative record file is similiar to a link sequential except that the logical record length, <lrchar>, must also be specified. A sample allocation of a relative record file with a physical record length of 256 words and a logical record length of 64 characters appears as follows:

```
//RUN  FILMGR  DISC=DISC1  FILE=(USER01,FILE2,ABC);
  NEW  REL  ACC=(ANY,PSWD,CREAT,CREAT);
  ALL=(50,0,256,50)  LRECL=64
```

This file is created with a password of ABC. Note the initial and maximum track allocation must be the same since the relative record file is not expandable.

8.16.4.4 CREATING A KEY INDEXED FILE. A key indexed file may be created with a Run command of the form:

```
//RUN FILMGR DISC=<devnam> FILE=(<fileid>,<filnam>,<pswd>) NEW IND;
  ACC=(<integ>,<integ>,<integ>,<integ>);
  ALL=(<intrks>,<trknum>,<prwrds>,<mtrks>) KEYLEN=<klchar>
```

The parameter, <klchar>, specifies the key length in characters. The maximum key length is 30 characters. To create a key indexed file with a key length of 6 characters, the following command may be used:

```
//RUN FILMGR DISC=DISC2 FILE=(SYSTEM, FILE3);
  NEW IND ACC=(ANY, PSWD, CREAT, NONE);
  ALL=(10, 0, 128, 20) KEYLEN=6
```

This file is created on DISC2 and has a physical record length of 128 words.

8.16.4.5 REPLACING FILES. If the replace (REP) parameter is specified instead of new (NEW), any file with the specified name is deleted before the new file is created. All information in the old file is destroyed.

8.16.5 ERROR CODES

Error messages for FILMGR are the same as those for DXCOPY.

APPENDIX A
ERROR MESSAGES

```
                         DX980 ERROR CODES
AS OF 04/09/75

  1      -OUT OF PARTITION REFERENCE IMPLIED BY PARAMETERS OF AN SVC CALL
  2      -JOB EXTENSION AREA TOO SMALL
  4      -NO SPACE IN DSCA
  5      -ILLEGAL NUMBER OF PARAMETERS IN SVC LIST
  6      -I/O ATTEMPTED ON NON-ASSIGNED LUNO
  7      -I/O ATTEMPTED WITHOUT OPEN
  8      -DUPLICATE OPEN ON SAME LUNO
  9      -WAIT CONTROL LIST ERROR FOUND ON USER SUSPEND
 10      -PRIORITY ERROR
 11      -CPU TIME EXCEEDED
 12      -ILLEGAL USER POST
 13      -ILLEGAL INSTRUCTION
 14      -A NON-EXISTENT SVC WAS ISSUED
 15      -USER HAS REQUESTED ACCESS TO A PRIVILEGED SVC
 16      -ILLEGAL SVC ARGUMENT -OUTSIDE USER PARTITION
 17      -PTR TO SVC ARG LIST OUTSIDE USER PARTITION
 18      -INVALID DEVICE I D
 19      -NO SPACE IN PCB
 20      -NO SYSTEM LUNO = 141
 21      -USER FILE DIRECTORY OVERFLOW
 22      -MASTER FILE DIRECTORY OVERFLOW
 23      -PREVIOUSLY DEFINED USER ID
 24      -ILLEGAL USER ID
 25      -ATTEMPT TO DELETE SYSTEM DISC MFD
 26      -INVALID ABNORMAL JOB TERMINATION CODE
 27      -UNDEFINED FILE
 28      -UNDEFINED USER ID
 29      -ATTEMPT TO REPLACE PREV ASSIGNED FILE
 30      -PREVIOUSLY DEFINED FILE
 31      -INVALID FILE TYPE
 32      -INSUFFICIENT TRK SPACE ON DEFINE
 33      -INSUFF. CONTIG. TRK SPACE ON DEFINE
 34      -EXCEEDED DISC SIZE ON DEFINE
 35      -ZERO KEY LENGTH FOR DEFINE
 36      -READY JSB FILE BAD
 37      -ATTEMPT TO DELETE A SHARED FILE
 38      -INVALID FILE DISPOSITION CODE
 39      -DEVICE OFFLINE
 40      -ATTEMPT TO SHARE UNSHARABLE DEVICE
 41      -ATTEMPT TO SHARE BLOCKED DEVICE
 42      -ATTEMPT TO ASSIGN EXCL A SHARED PASSED RESOURCE
 43      -OPERATOR CANCELLATION
 44      -TOO MANY JOB STEPS
 45      -INVALID JCB
 46      -INVALID INPUT LDT
 47      -JOB NO/STEP NO. NOT IN SYSTEM
 48      -JOB NAME NOT IN SYSTEM
 49      -ATTEMPT TO ILLEGALLY ACCESS FILE
 50      -POTENTIAL RESOURCE DEADLOCK DUE TO INCOMPLETE PASSING
 51      -INVALID JCB SIZE SPECIFIED
 52      -ATTEMPT TO DEASSIGN UNASSIGNED LUNO
 53      -TOO MANY JOB STEPS (>15) IN ONE JOB STRING
 54      -PARENT JOB ENDED BEFORE JOB STRING STARTED
 55      -LOAD MODULE TOO BIG FOR SPECIFIED USER SPACE.
 56      -NO JOB INITIATION SYSTEM TASK FOR JOB
 57      -LOAD MODULE LOAD NO GOOD
 58      -ATTEMPT TO ASSIGN TO DISC DIRECTLY
 59      -DEASSIGNMENT OF OPEN DEVICE/FILE
 60  SOQM-SYSTEM OUTPUT QUEUE OVERFLOW
 61  SOQM-TOO MANY OUTPUT FILES
 62      -MEMORY PARITY ERROR
 63      -MEMORY PROTECT ERROR - ADDRESSING ERROR
 64      -PRIVILEDGE INSTRUCTION VIOLATION
 65      -RESOURCE STACK OVERFLOW
 66      -BYTE RELOCATION ADDR BAD IN LD MOD
 67      -MIP NO BAD FOR LOAD OR LOADR
 69      -LOAD OR LOADR EXTENDS BEYOND USER MEM
 70   BPS-CAN NOT ALLOCATE INPUT DEVICE (REASON-MEMORY OR DEVICE NOT AVAILABLE)
 71   BPS-READ ERROR ON INPUT DEVICE
 72BPSOC-ILLEGAL OR MISSING "JOB" COMMAND
 73BPSOC-ILLEGAL RUN COMMAND
 74   BPS-ILLEGAL DATA COMMAND
 75   BPS-TOO MANY INPUT DATA FILE
 76   BPS-NO OF INPUT DATA FILES UNMATCHED WITH NO OF ASSIGNED INPUT DATA FILES
 77   BPS-DATA COMMAND UNMATCHED WITH INPUT ASSIGNMENTS
 78   BPS-OUTPUT QUEUE ERROR - REINITIALIZE QUEUE TO USE BOS
 80      -JOB/STEP NOT IN ROLL FILE DIRECTORY
 81      -NO SPACE AVAILABLE IN ROLL FILE
 82      -INSUFFICIENT ROLLABLE MEMORY
 83      -ROLL PERFORMED NORMALLY
 84      -ROLL FILE CLOBBERED
 85   BPS-DATA ERROR ON LINE PRINTER
 86   BPS-END OF FILE ENCOUNTERED WHILE SKIPPING RECORDS
 87      -REQUEST FOR MORE MEMORY THAN IN FREE MEMORY
 88      -DEVICE FILE REQUESTED AT RUNTIME NOT AVAILABLE
 89      -INVALID JOB STEP NUMBER IN JSB
 90      -ILLEGAL NUMBER INPUT
 91      -ILLEGAL COMMAND
 92      -TOO MANY JOBS IN THE SYSTEM
 93      -LUNO LDT NOT FOUND
 94      -NOT A RR FILE
 95   JCL-USER-ID SPECIFIED FOR PROCEDURE LIBRARY DOES NOT EXIT
 96   JCL-PROCEDURE LIBRARY DOES NOT EXIST UNDER SPECIFIED USER-ID
 97   JCL-USER CAN'T GAIN ACCESS TO PROCEDURE LIBRARY BECAUSE OF INTEGRITY CODE
 98   JCL-SPECIFIED PROCEDURE DOES NOT EXIST IN PROCEDURE LIBRARY
 99   JCL-HARDWARE FAILURE WHILE ATTEMPTING READ FROM SPECIFIED PROCEDURE LIB
100    JM-ATTEMPTED TO USE FILE OF RESTRICTED USER ID
101    OC-HARDWARE I/O ERROR IN OP. COMMUNICATIONS
102    OC-INVALID MESSAGE ID
103    OC-INVALID OPERAND IN OP.COMMUNICATIONS
104    OC-INVALID ARGUMENT LIST IN OP.COMMUNICATIONS
105    OC-INVALID JOB NUMBER PASSED IN OP.COMMUNICATIONS
106    OC-ATTEMPT TO OFFLINE SYSTEM DISC OR SYSTEM CONSOLE
107    OC-NO SPACE IN DSCA OR JEA
108    OC-INVALID OP.COMMUNICATIONS COMMAND
109    OC-JOB NUMBER NOT FOUND BY OP.COMMUNICATIONS
110    OC-IN OJCBPR, INVALID SIZE REQUIRED FOR '//JOB' JSB
111    OC-INVALID NUMBER USED FOR SKIP COMMAND TO BATCH OUTPUT SPOOLER
112    OC-UNDEFINED COMMAND GIVEN TO BATCH OUTPUT SPOOLER
201    IO-DEVICE NOT READY
202    IO-CONTROLLER ERROR
203    IO-DATA ERROR
204    IO-CONTROLLER BUSY ERROR
205    IO-WRITE PROTECT ERROR
206    IO-EOR ERROR
207    IO-READ-AFTER-WRITE ERROR
208    IO-DEVICE OFFLINE
209    IO-ILLEGAL OP-CODE
```

210	IO-DEVICE TIMEOUT (DEVICE DID NOT RESPOND)
233	FM-NO SPACE AVAILABLE ON DISC VOLUME
234	FM-FILE FULL I/O ERROR
235	FM-ATTEMPTED WRITE, LOGICAL RECORD >= PHYSICAL RECORD + OVERHEAD
236	FM-HARDWARE FAILURE ON DISC VOLUME
237	FM-INDEX, REPLACE ATTEMPTED ON NON-EXISTING KEY
238	FM-EXISTING KEY FOUND ON 'WRITE' OP-CODE --OPERATION NOT PERFORMED
239	FM-INDEX, WRITE/REPLACE ATTEMPTED ON NON-KEYED RECORD
240	FM-INDEX, REPLACE ATTEMPTED ON KEYED RECORD
241	FM-INDEX, REPLACE ATTEMPTED ON NULL DATA (NON-EXISTENT)
243	FM-INDEX, REL-REC, NO KEY AFTER SEARCH
249	FM-INVALID FILE TYPE (NON-EXISTENT)
250	FM-INSUFFICIENT TRACKS FOR ALLOCATION
251	FM-INSUFFICIENT CONTIGUOUS TRACKS LEFT ON DISC VOLUME
252	FM-ALLOCATION EXCEEDS DISC VOLUME CAPACITY
254	FM-UNABLE TO ALLOCATE BUFFERS BECAUSE OF JOB EXTENSION SIZE
256	FM-INSUFFICIENT NUMBER OF BUFFERS FOR ATTEMPTED OPERATION
257	FM-OPCODE IS EITHER NON-EXISTENT OR ILLEGAL
258	FM-ACCESS VIOLATION, INTEGRITY ERROR
401	CSCN-OVERFLOW OF KEYWORD AREA
402	CSCN-OVERFLOW OF PACKED STRING STORAGE
403	CSCN-R.H.S. OF EXPRESSION OR TERM MISSING
404	CSCN-ILLEGAL EXPRESSION SUBSCRIPT
405	CSCN-MISSING DELIMETER AFTER COMMAND ID
406	CSCN-NUMBER IS LARGER THAN 16 BITS
407	CSCN-OPERAND STARTS WITH ILLEGAL CHARACTER
408	CSCN-ILLEGAL DIGIT IN DECIMAL NUMBER
409	CSCN-MISSING DELIMTER BETWEEN OPERANDS
410	CSCN-MISSING DELIMETER BETWEEN SUBSCRIPTS
411	CSCN-ILLEGAL CHARACTER PRECEEDS COMMAND
412	ITS-RUN COMMAND DOES NOT CONTAIN A LABEL OR AN EXPRESSION
413	CSCN-MISSING EQUAL SIGN IN ASSIGNMENT
414	CSCN-RIGHT HAND SIDE OF ASSIGNMENT MISSING
415	CSCN-MORE THAN ONE = SIGN IN EXPRESSION
416	CSCN-SIZE OF PACKED STRING < 0 CHARACTERS
417	CSCN-UPPER BOUND ON KEYWORD AREA < 1
418	CSCN-NUMBER OF RESERVED LABELS < 0
419	CSCN-STARTING COLUMN FOR SCAN NOT IN RANGE (0:79)
420	JCL-FOR A NEW FILE, USER DIRECTORY NAME DIFFERS FROM CURRENT USER
421	JCL-JSB MUST CONTAIN DEVICE INDEX, NOT THE PDT ADDRESS
422	JCL-DEVICE INDEX MUST BE <= 255
423	JCL-PHYSICAL R.L. < KEY-LENGTH + 14
424	JCL-FILE HAS BAD ACCESS CODE VALUE
425	JCL-BOTH 'DELETE' & 'PASS' SPECIFIED
426	JCL-LOGICAL R.L. > PHYSICAL R.L.
427	JCL-LOGICAL R.L. ^ MULTIPLE OF 32
428	JCL-DEVICE NOT SPECIFIED OR INCORRECTLY SPECIFIED
429	JCL-USER ID NOT SPECIFIED
430	JCL-FILE NAME NOT SPECIFIED
433	JCL-# PRIORITY LEVELS > 31 OR < 1
434	JCL-JOB STEP PRIORITY > 31 OR < 1
435	JCL-OBJ. VOLUMN ID IS > 20 OR < 1
436	JCL-TCB STACK SIZE < 1 WORD
437	JCL-ILLEGAL COMMAND AFTER 'DELETE'
439	JCL-TIME LIMIT < 1 SECOND
443	JCL-# PRIORITY LEVELS + JOB STEP PRIORITY IS > 31 OR < 1
444	JCL-JOB EXTENSION SIZE <= TCB STACK SIZE + 15 WORDS
446	JCL-VOL USER ID NOT INITIALIZED
447	JCL-VOLUMN FILE NOT INITIALIZED
448	JCL-VOL USER ID OR FILE NAME > 6 CHARS
449	JCL-VOL PASSWORD > 4 CHARS
450	JCL-WRONG # OF OPERANDS ON THE RIGHT SIDE OF AN EXPRESSION

451	JCL-OPERAND IS NOT A LABEL OR A SUBSCRIPTED EXPRESSION
452	JCL-OPERAND ON THE RIGHT SIDE IS NOT A LABEL OR A NUMBER
453	JCL-BAD DEVICE NAME
454	JCL-BLOCK SIZE < 1
456	JCL-REDEFINITION OF LUNO IN JOBSTEP
457	JCL-PASSWORD > 4 CHARACTERS
458	JCL-USER ID OR FILE NAME > 6 CHARACTERS
459	JCL-NUMBER BUFFERS < 1
46Ø	JCL-RE-INITIALIZATION OF JSB ITEM
461	JCL-BAD LABEL FOR ACCESS CODE
462	JCL-INITIAL TRACKS < 1
463	JCL-FIRST TRACK ADDRESS < Ø
464	JCL-PHYSICAL RECORD LENGTH < 32
465	JCL-PHYSICAL R.L. NOT MULTIPLE OF 32
466	JCL-MAX TRACKS < (INITIAL OR 1)
467	JCL-LOGICAL RECORD LENGTH < 1
468	JCL-LUNO NUMBER NOT IN RANGE Ø TO 254
469	JCL-KEY LENGTH NOT IN RANGE 1 TO 3Ø
47Ø	JCL-OPERAND DOESN'T START WITH LABEL
471	JCL-KEYNAME ON THE LEFT SIDE OF AN EXPRESSION IN NOT DEFINED
475	JCL-OVERFLOW OF KEY-ENTRY TABLE
476	JCL-OVERFLOW OF KEY-REFERENCE TABLE
477	JCL-OVERFLOW OF KEY-CHARS TABLE
478	JCL-DISC ERROR ON LUNO 4
479	JCL-INDEX KEY NAME FOLLOWING 'CREATE' OR 'REPLACE' > 6 CHARACTERS
48Ø	JCL-GREATER 31 ASSIGN CARDS IN THIS JOB STEP
481	JCL-'CREATE' OR 'REPLACE' NOT FOLLOWED BY INDEX KEY NAME
49Ø	JCL-DEV INDEX < 21 OR FILE VOL > 2Ø
491	JCL-PHYSICAL R.L. (CHAR) < (KEYLEN + 2 + 14)
492	JCL-FILE TYPE NOT SPECIFIED WHEN NEEDED (DEFINE, ETC.)
5Ø1	JCL-KEYNAME IS NOT IN RESERVED WORD LIST
5Ø5	JCL-# OF ENTRIES IN TABLE PASSED TO 'CRLOOK' IS NEGATIVE
51Ø	JCL-SUBSCRIPTS APPEAR ON LHS OF EQUAL SIGN
511	JCL-TRIED TO FETCH NON-EXISTANT RHS SUBSCRIPT
512	JCL-TRIED TO FETCH NON-EXISTANT OPERAND
52Ø	JCL-I/O ERROR ON LUNO = 2. (ECHO PRINT)
521	JCL-EOF, EOM, OR I/O ERROR ON LUNO = 1. (JCL IN)
522	JCL-MISSING SLASH IN FIRST COLUMN OF JCL
523	JCL-OVERRIDING KEY WORD ON RUN CARD DOES NOT EXIST FOR THIS PROCEDURE
526	JCL-NUMBER OF KEY-ENTRIES IS NEGATIVE
527	JCL-NUMBER OF KEY-ENTRIES > 19
528	JCL-NUMB5R OF KEY-REFERENCES < Ø OR > 19
529	JCL-NUMBER OF KEY-CHARACTERS NOT IN RANGE Ø TO 12Ø
53Ø	JCL-KEY-ENTRY POINTS TO KEY REFERENCE WHICH IS NOT INITIALIZED
531	JCL-KEY-ENTRY FLAG INDICATES THAT NEITHER JCB NOR LDT IS BEING INITIALIZED
532	JCL-LDT NUMBER REFERENCE BY KEY-ENTRY IS NOT IN RANGE Ø TO 3Ø
533	JCL-PRODUCTION NUMBER NOT IN RANGE 3 TO 31
534	JCL-JOB STEP NUMBER NOT IN RANGE 1 TO 15
534	JCL-JOB STEP NUMBER NOT IN RANGE 1 TO 15
535	JCL-KEY-CHARS HAS LESS THAN Ø CHARACTERS
536	JCL-CHARACTERS OVERFLOW KEY-CHAR STORAGE

APPENDIX B
SAMPLE JCL SEQUENCES

APPENDIX B

SAMPLE JCL SEQUENCES

Job Control Sequence (JCS) for particular system installations are supplied on file JCLSRC contained on the disc image tape. List this file after completing installation.

This appendix contains sample listings of some sequences. These samples are not necessarily the exact sequences supplied on the previously mentioned tape.

```
.#UPDATE
.#TAPE    'DX980 JCL SOURCE FILE','           ','05/15/75',*'*A'.
.# CREATE ASMBLR,COMMENT,"ASSEMBLE                          "
/REPLACE ASMBLR     . ASSEMBLE .
/EXEC OBJ=(1,SYSTEM,ASMBLR) MEM=(300,5000,1000) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  0 DUMMY DEVICE:=DMSG SHARE                          . SYSTEM MESSAGE
/ASSIGN  4 DUMMY DEVICE:=DCON SHARE:=SCON                    . CONTROL/MESSAGE
/ASSIGN  5 DISC1 DEVICE:=DSRC FILE:=FSRC BUFFERS=1           . SOURCE INPUT
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.SOURCE LIST/ERROR
/ASSIGN  7 DISC1 DEVICE:=DOBJ FILE:=FOBJ NEW:=NOBJ;
/          REPLACE:=ROBJ BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=COBJ;
/          ALLOCATE=(1,0,64,10) ALLOCATE:=LOBJ              . OBJECT OUTPUT
/ASSIGN 16 DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,256,30) . SOURCE SCRATCH
/END
.# CREATE ASMGO ,COMMENT,"EXECUTE ASM LANG GENED L M  "
/REPLACE ASMGO      . EXECUTE ASSEMBLY LANGUAGE GENERATED LOAD MODULE .
/EXEC OBJ=(1,USER01,GO) MEM=(300,4000,1000) PRTY=(1,15);
/           TIME=100 OBJ:=OBJ MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  4 DUMMY LUNO:=LUN4 DEVICE:=DEV4 FILE:=FIL4 BUFFERS=2. USER PROG LUN 4
/ASSIGN  5 SC    LUNO:=LUN5 DEVICE:=DEV5 FILE:=FIL5 BUFFERS=2. USER PROG LUN 5
/ASSIGN  6 SC    LUNO:=LUN6 DEVICE:=DEV6 FILE:=FIL6 BUFFERS=2. USER PROG LUN 6
/ASSIGN  7 DUMMY LUNO:=LUN7 DEVICE:=DEV7 FILE:=FIL7 BUFFERS=2. USER PROG LUN 7
/ASSIGN  8 DUMMY LUNO:=LUN8 DEVICE:=DEV8 FILE:=FIL8 BUFFERS=2. USER PROG LUN 8
/END
.# CREATE ASMLGO,COMMENT,"ASSEMBLE, LINK, AND GO      "
/REPLACE ASMLGO     . ASSEMBLE, LINK, AND GO .
/EXEC OBJ=(1,SYSTEM,ASMBLR) MEM=(300,5000,1000) PRTY=(1,15);
/          TIME=-1 MEM:=MEMA
/ASSIGN  0 DUMMY DEVICE:=DMSG SHARE                          . SYSTEM MESSAGE
/ASSIGN  4 DUMMY DEVICE:=DCON SHARE:=SCON                    . CONTROL/MESSAGE
/ASSIGN  5 DISC1 DEVICE:=DSRC FILE:=FSRC BUFFERS=1           . SOURCE INPUT
/ASSIGN  6 SC    DEVICE:=DLSTA FILE:=FLSTA BUFFERS=1         . SOURCE LIST/ERROR
/ASSIGN  7 DISC1 FILE=(TEMP,OBJECT) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,64,10)  . OBJECT OUTPUT
/ASSIGN 16 DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,256,30) . SOURCE SCRATCH
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(300,12000,3000) PRTY=(1,15);
/          TIME=-1 MEM:=MEML
/ASSIGN  5 DISC1 FILE=(TEMP,OBJECT) DELETE BUFFERS=1         . PRIMARY INPUT/CON
/ASSIGN  6 SC    DEVICE:=DLSTL FILE:=FLSTL BUFFERS=1         . LOADMAP LIST/ERR
/ASSIGN  8 DISC1 FILE=(TEMP,LM) NEW BUFFERS=1 RELREC LRECL=64;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,32,10)  . LOAD MOD OUTPUT
/ASSIGN  9 DUMMY                                             . LIBRARY FILE
/ASSIGN 10 DISC1 FILE=(TEMP,SCRL) DELETE BUFFERS=1           . LINKSEQ SCRATCH
/ASSIGN 13 DISC1 FILE=(TEMP,SCRR) NEW DELETE BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10);
/          LRECL=100                                         . RELREC SCRATCH
/EXEC OBJ=(1,TEMP,LM) MEM=(300,4000,1000) PRTY=(1,15);
/          TIME=100 MEM:=MEMG TIME:=TIMG
/ASSIGN  4 DUMMY DEVICE:=DEV4 FILE:=FIL4 BUFFERS=2           . USER PROG LUN 4
/ASSIGN  5 SC    DEVICE:=DEV5 FILE:=FIL5 BUFFERS=2           . USER PROG LUN 5
/ASSIGN  6 SC    DEVICE:=DEV6 FILE:=FIL6 BUFFERS=2           . USER PROG LUN 6
/ASSIGN  7 DUMMY DEVICE:=DEV7                                . USER PROG LUN 7
/END
```

```
.# CREATE ASMLNK,COMMENT,"ASSEMBLE AND LINK          "
/REPLACE ASMLNK     . ASSEMBLE AND LINK .
/EXEC OBJ=(1,SYSTEM,ASMBLR) MEM=(300,5000,1000) PRTY=(1,15);
/          TIME=-1 MEM:=MEMA
/ASSIGN  0 DUMMY DEVICE:=DMSG SHARE                         , SYSTEM MESSAGE
/ASSIGN  4 DUMMY DEVICE:=DCON SHARE:=SCON                   , CONTROL/MESSAGE
/ASSIGN  5 DISC1 DEVICE:=DSRC FILE:=FSRC BUFFERS=1          . SOURCE INPUT
/ASSIGN  6 SC    DEVICE:=DLSTA FILE:=FLSTA BUFFERS=1        . SOURCE LIST/ERROR
/ASSIGN  7 DISC1 FILE=(TEMP,OBJECT) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,64,10)  . OBJECT OUTPUT
/ASSIGN 16 DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,256,30) . SOURCE SCRATCH
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(300,12000,3000) PRTY=(1,15);
/          TIME=-1 MEM:=MEML
/ASSIGN  5 DISC1 FILE=(TEMP,OBJECT) DELETE BUFFERS=1        , PRIMARY INPUT/CON
/ASSIGN  6 SC    DEVICE:=DLSTL FILE:=FLSTL BUFFERS=1        . LOADMAP LIST/ERR
/ASSIGN  8 DISC1 DEVICE:=DLM FILE=(USER01,GO) FILE:=FLM;
/          REPLACE:=RLM BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=CLM;
/          ALLOCATE=(10,0,32,10) ALLOCATE:=LLM LRECL=64     , LOAD MOD OUTPUT
/ASSIGN  9 DUMMY DEVICE:=DLIB FILE:=FLIB BUFFERS=1          , LIBRARY FILE
/ASSIGN 10 DISC1 FILE=(TEMP,SCRL) DELETE BUFFERS=1          . LINKSEQ SCRATCH
/ASSIGN 13 DISC1 FILE=(TEMP,SCRR) NEW DELETE BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10);
/          LRECL=100                                         . RELREC SCRATCH
/END
.# CREATE ASMUP ,COMMENT,"ASSEMBLE MOD AND UPDATE LPF "
/REPLACE ASMUP      . ASSEMBLE MODULE AND UPDATE LPF .
/EXEC OBJ=(1,SYSTEM,ASMBLR) MEM=(300,6000,2000) PRTY=(1,2);
/          TIME=-1 MEM:=MEMA
/ASSIGN  0 DUMMY DEVICE:=DMSG                               , SYSTEM MESSAGE
/ASSIGN  4 DUMMY DEVICE:=DCON                               , CONTROL/MESSAGE
/ASSIGN  5 DISC1 DEVICE:=DSRC FILE:=FSRC BUFFERS=1          , SOURCE INPUT
/ASSIGN  6 SC    DEVICE:=DLSTA FILE:=FLST BUFFERS=1         . SOURCE LIST/ERROR
/ASSIGN  7 DISC1 FILE=(TEMP,OBJECT) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,64,20)  . OBJECT OUTPUT
/ASSIGN 16 DISC1 FILE=(TEMP,SCRL) NEW DELETE BUFFERS=1;
/          LINKSEQ ACCESS=(ANY,ANY,ANY,ANY);
/          ALLOCATE=(10,300,256,30)                          . SOURCE SCRATCH
/EXEC OBJ=(1,SYSTEM,LPFBLD) MEM=(300,600,1000) PRTY=(1,2);
/          TIME=-1 MEM:=MEMU
/ASSIGN  0 DUMMY DEVICE:=DCON                               , CONTROL
/ASSIGN  5 DISC1 FILE=(TEMP,OBJECT) BUFFERS=1               , OBJECT INPUT
/ASSIGN  6 SC    DEVICE:=DLSTU FILE:=FLST BUFFERS=1         , LISTING
/ASSIGN  9 DISC1 DEVICE:=DUPD FILE:=FUPD REPLACE:=RUPD;     .
/          BUFFERS=2 INDEXED ACCESS=(CREAT,CREAT,CREAT,CREAT);
/          ACCESS:=CUPD ALLOCATE:=LUPD KEYLEN=6             . UPDATE FILE
/END
.# CREATE BIR   ,COMMENT,"BATCH INPUT READER            "
/REPLACE BIR        . BATCH INPUT READER .
/EXEC OBJ=(1,SYSTEM,BIR) MEM=(300,900,600) PRTY=(1,5);
/          PRIV TIME=-1 PRTY:=PRI TIME:=TIM
/ASSIGN 10 CR1   DEVICE:=DIN                                . INPUT STREAM
/END
.# CREATE BIS   ,COMMENT,"BATCH INPUT SPOOLER           "
/REPLACE BIS        . BATCH INPUT SPOOLER .
/EXEC OBJ=(1,SYSTEM,BIS) MEM=(300,1200,800) PRTY=(1,5);
/          PRIV TIME=-1 PRTY:=PRI TIME:=TIM   MEM:=MEM
/ASSIGN 5 CR1   DEVICE:=DIN                                 . INPUT STREAM
/END
```

```
.# CREATE BLDEDT,COMMENT,"BUILD EDIT FILE                "
/REPLACE BLDEDT      . BUILD EDIT FILE .
/EXEC OBJ=(1,SYSTEM,BLDEDT) MEM=(300,550,2000) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN 10 DISC1 DEVICE:=DIN FILE:=FIN BUFFERS=1              . SOURCE INPUT
/ASSIGN 20 DISC1 DEVICE:=DOUT FILE:=FOUT REPLACE BUFFERS=2;
/          BUFFERS:=BOUT INDEXED ACCESS=(ANY,ANY,ANY,ANY);
/          ACCESS:=COUT ALLOCATE=(1,0,256,100) ALLOCATE:=LOUT;
/          KEYLEN=2                                           . SOURCE OUT FILE
/END
.# CREATE BOS   ,COMMENT,"BATCH OUTPUT SPOOLER           "
/REPLACE BOS          . BATCH OUTPUT SPOOLER .
/EXEC OBJ=(1,SYSTEM,BOS) MEM=(300,1100,800) PRTY=(1,5);
/          PRIV TIME=-1 PRTY:=PRI TIME:=TIM
/ASSIGN  5 SC    DEVICE:=DOUT                                 . OUTPUT LISTING
/ASSIGN  6 DUMMY DEVICE:=DNEW FILE=(SYSTEM,NEWS) FILE:=FNEW;
/          BUFFERS=1
/END
.# CREATE CATFIL,COMMENT,"LIST FILES UNDER A USER        "
/REPLACE CATFIL      . LIST FILES UNDER A USER .
/EXEC OBJ=(1,SYSTEM,CATFIL) MEM=(300,1850,670) PRTY=(1,5);
/          PRIV TIME=-1 PRTY:=PRI
/ASSIGN  0 SC    DEVICE:=DCON SHARE:=SCON                     . CONTROL
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.FILE LISTING
/END
.# CREATE CATLOG,COMMENT,"LIST, CREATE,OR DELETE USERS"
/REPLACE CATLOG      . LIST, CREATE, OR DELETE USERS .
/EXEC OBJ=(1,SYSTEM,CATLOG) MEM=(300,4000,670) PRTY=(1,5);
/          PRIV TIME=-1 PRTY:=PRI
/ASSIGN  0 SC    DEVICE:=DCON SHARE:=SCON                     . CONTROL
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.USER LISTING
/END
.# CREATE DEB980,COMMENT,"PROGRAM DEBUG AID              "
/REPLACE DEB980      . PROGRAM DEBUG AID .
/EXEC OBJ=(1,SYSTEM,DEB980) MEM=(300,6000,600) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN >F0 SC    DEVICE:=DCIN                                , CONTROL INPUT
/ASSIGN >F1 SC    DEVICE:=DMSG                                . SYSTEM MESSAGE
/ASSIGN >F2 DUMMY DEVICE:=DCLST                               , CONTROL LISTING
/ASSIGN >F3 SC    DEVICE:=DUMP FILE:=FUMP BUFFERS=1           , MEMORY DUMP
/ASSIGN >F4 DISC1 DEVICE:=DOBJ FILE:=FOBJ BUFFERS=1           , RELOC OBJECT IN
/ASSIGN   0 DUMMY DEVICE:=DEV0                                , USER PROG LUN 0
/ASSIGN   4 DUMMY DEVICE:=DEV4 FILE:=FIL4 BUFFERS=1           , USER PROG LUN 4
/ASSIGN   5 DUMMY DEVICE:=DEV5 FILE:=FIL5 BUFFERS=1           , USER PROG LUN 5
/ASSIGN   6 DUMMY DEVICE:=DEV6 FILE:=FIL6 BUFFERS=1           , USER PROG LUN 6
/ASSIGN   7 DUMMY DEVICE:=DEV7                                , USER PROG LUN 7
/ASSIGN   8 DUMMY LUNO:=LUN8 DEVICE:=DEV8                     . USER PROG LUN 8
/END
.# CREATE DUMPLP,COMMENT,"DUMP LPF FROM DISC TO MT       "
/REPLACE DUMPLP      . DUMP LPF FROM DISC TO MAG TAPE .
/EXEC OBJ=(1,SYSTEM,DXCOPY) MEM=(300,4000,3000) PRTY=(1,2);
/          TIME=-1 MEM:=MEM
/ASSIGN  5 DUMMY DEVICE:=DCON                                 , CONTROL/MESSAGE
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST BUFFERS=1            , LISTING
/ASSIGN  7 MT1   DEVICE:=DOUT FILE:=FOUT BUFFERS=2            , OUTPUT
/ASSIGN  8 DISC1 DEVICE:=DIN  FILE=(SYSTEM,LPF) FILE:=FIN;    . INPUT
/                BUFFERS=2
/END
```

```
.# CREATE DXCOPY,COMMENT,"GENERAL PURPOSE COPY         "
/REPLACE DXCOPY     . GENERAL PURPOSE COPY .
/EXEC OBJ=(1,SYSTEM,DXCOPY) MEM=(300,3700,2000) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  5 DUMMY DEVICE:=DCON                                  . CONTROL/MESSAGE
/ASSIGN  6 DUMMY DEVICE:=DLST FILE:=FLST BUFFERS=1             . LISTING
/ASSIGN  7 DISC1 DEVICE:=DOUT FILE:=FOUT REPLACE:=ROUT;
/          BUFFERS=2 BUFFERS:=BOUT LINKSEQ:=LIN RELREC:=REL;
/          INDEXED:=IND ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=COUT;
/          ALLOCATE=(1,0,128,10) ALLOCATE:=LOUT KEYLEN=6;
/          KEYLEN:=KOUT LRECL=64  LRECL:=GOUT                  . OUTPUT
/ASSIGN  8 DISC1 DEVICE:=DIN FILE:=FIN DELETE:=TIN BUFFERS=1   . INPUT
/END
.# CREATE DXLINK,COMMENT,"LINK DX980 OPERATING SYSTEM "
/REPLACE DXLINK     . LINK DX980 OPERATING SYSTEM .
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(300,31500,8000) PRTY=(1,2);
/          TIME=-1
/ASSIGN  1 DUMMY DEVICE:=DLP1 FILE:=FLP1 BUFFERS=2             . ALT 1 OBJECT IN
/ASSIGN  2 DISC1 DEVICE:=DLP2 FILE=(SYSTEM,DXLPF) FILE:=FLP2;
/          BUFFERS=2                                           . ALT 2 OBJECT IN
/ASSIGN  3 DUMMY DEVICE:=DLP3 FILE:=FLP3 BUFFERS=2             . ALT 3 OBJECT IN
/ASSIGN  5 DISC1 DEVICE:=DIN  FILE=(USER01,LINKDX) FILE:=FIN;
/          BUFFERS=1                                           . PRIMARY INPUT/CON
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST BUFFERS=1             . LOADMAP LIST/ERR
/ASSIGN  8 DISC1 DEVICE:=DLM  FILE=(USER01,DXMIP)  FILE:=FLM;
/          REPLACE:=RLM BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY));
/          ALLOCATE=(32,247,32,32) ALLOCATE:=LLM LRECL=64      . LOAD MOD OUTPUT
/ASSIGN  9 DISC1 DEVICE:=DLIB FILE=(SYSTEM,USRPLX) FILE:=FLIB;
/          BUFFERS=2                                           . LIBRARY FILE
/ASSIGN 10 DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,256,80)   . LINKSEQ SCRATCH
/ASSIGN 12 DISC1 DEVICE:=DEXT FILE=(USER01,DXEXTD) FILE:=FEXT;
/          REPLACE:=REXT BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=CEXT;
/          ALLOCATE=(1,0,128,1)                                . SYS EXT DEFS FILE
/ASSIGN 13 DISC1 FILE=(TEMP,SCRR) NEW BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(2,300,128,2);
/          LRECL=100                                           . RELREC SCRATCH
/END

/REPLACE DXOLE      . LINK EDITOR .
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(300,12000,2000) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  1 DUMMY DEVICE:=DOB1 FILE:=FOB1 BUFFERS=2               . ALT 1 OBJECT IN
/ASSIGN  2 DUMMY DEVICE:=DOB2 FILE:=FOB2 BUFFERS=2               . ALT 2 OBJECT IN
/ASSIGN  5 DISC1 DEVICE:=DIN  FILE:=FIN  BUFFERS=1               . PRIMARY INPUT/CON
/ASSIGN  6 SC    DEVICE:=DLST                                    . LOADMAP LIST/ERR
/ASSIGN  7 DUMMY DEVICE:=DOBJ FILE:=FOBJ BUFFERS=1               . COMPACT OBJ OUT
/ASSIGN  8 DISC1 DEVICE:=DLM  FILE:=FLM  REPLACE:=RLM;
/          BUFFERS=1 RELREC ACCESS=(ANY,CREAT,CREAT,ANY));
/          ACCESS:=CLM ALLOCATE=(1,0,32,1) ALLOCATE:=LLM;
/          LRECL=64                                              . LOAD MOD OUTPUT
/ASSIGN  9 DISC1 FILE=(SYSTEM,USRFTN) DEVICE:=DLIB;
/          FILE:=FLIB BUFFERS=2                                  . LIBRARY FILE
/ASSIGN 10 DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,256,30)     . LINKSEQ SCRATCH
/ASSIGN 11 DISC1 FILE=(SYSTEM,DXEXTD) FILE:=FEXT BUFFERS=1       . SYS EXT DEFS OPT
/ASSIGN 13 DISC1 FILE=(TEMP,SCRR) NEW BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10));
/          LRECL=100                                             . RELREC SCRATCH
/END
```

```
.# CREATE DXOLEP,COMMENT,"LINK PLEXUS PROGRAMS         "
/REPLACE DXOLEP      . LINK PLEXUS PROGRAMS .
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(3ØØ,32ØØØ,3ØØØ) PRTY=(1,2);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  1 DUMMY DEVICE:=DOB1 FILE=(USERØ1,ASMOUT) FILE:=FOB1;
/          BUFFERS=2                                     . ALT 1 OBJECT IN
/ASSIGN  2 DUMMY DEVICE:=DOB2 FILE:=FOB2 BUFFERS=2       . ALT 2 OBJECT IN
/ASSIGN  5 DISC1 DEVICE:=DIN FILE:=FIN BUFFERS=1         . PRIMARY INPUT/CON
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST BUFFERS=1       . LOADMAP LIST/ERR
/ASSIGN  7 DUMMY DEVICE:=DOBJ FILE:=FOBJ BUFFERS=1       . COMPACT OBJ OUT
/ASSIGN  8 DISC1 DEVICE:=DLM  FILE:=FLM  REPLACE:=RLM;
/          BUFFERS=1 RELREC ACCESS=(ANY,CREAT,CREAT,ANY);
/          ACCESS:=CLM ALLOCATE=(1,Ø,32,1) ALLOCATE:=LLM;
/          LRECL=64                                      . LOAD MOD OUTPUT
/ASSIGN  9 DISC1 FILE=(SYSTEM,USRPLX) FILE:=FLIB BUFFERS=2   . LIBRARY FILE
/ASSIGN 1Ø DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(1Ø,3ØØ,256,3Ø) . LINKSEQ SCRATCH
/ASSIGN 11 DISC1 FILE=(SYSTEM,DXEXTD) FILE:=FEXT BUFFERS=1   . SYS EXT DEFS OPT
/ASSIGN 13 DISC1 FILE=(TEMP,SCRR) NEW BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(1Ø,3ØØ,128,1Ø);
/          LRECL=1ØØ                                     . RELREC SCRATCH
/END
.# CREATE FILMGR,COMMENT,"FILE CREATE/DELETE CAPABILITY"
/REPLACE FILMGR      . FILE CREATE/DELETE CAPABILITY .
/EXEC OBJ=(1,SYSTEM,DXCOPY) MEM=(3ØØ,265Ø,85Ø) PRTY=(1,15);
/          TIME=-1 PRTY:=PRI TIME:=TIM
/ASSIGN  1 DISC1 DEVICE:=DISC FILE:=FILE NEW:=NEW REPLACE:=REP;
/          DELETE:=DEL BUFFERS=1 LINKSEQ:=LIN RELREC:=REL;   . FILE TO BE
/          INDEXED:=IND ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=ACC;. CREATED/DELETED
/          ALLOCATE:=ALL KEYLEN=6 KEYLEN:=KEY LRECL=64;
/          LRECL:=LRE
/ASSIGN  5 DUMMY
/ASSIGN  6 DUMMY
/ASSIGN  7 DUMMY
/ASSIGN  8 DUMMY
/END
.# CREATE FTNGO ,COMMENT,"EXECUTE FORTRAN GENERATED LM"
/REPLACE FTNGO       . EXECUTE FORTRAN GENERATED LOAD MODULE .
/EXEC OBJ=(1,USERØ1,GO) MEM=(3ØØ,12ØØØ,1ØØØ) PRTY=(1,15);
/            TIME=1ØØ OBJ:=OBJ MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN   Ø SC     DEVICE:=DMSG                          . SYSTEM MESSAGE
/ASSIGN >BØ SC     LUNO:=LUNØ DEVICE:=DEVØ               . USER PROG LUN Ø
/ASSIGN >B1 DUMMY  LUNO:=LUN1 DEVICE:=DEV1               . USER PROG LUN 1
/ASSIGN >B4 DUMMY  LUNO:=LUN4 DEVICE:=DEV4 FILE:=FIL4 BUFFERS=2.USER PROG LUN 4
/ASSIGN >B5 SC     LUNO:=LUN5 DEVICE:=DEV5 FILE:=FIL5 BUFFERS=2.USER PROG LUN 5
/ASSIGN >B6 SC     LUNO:=LUN6 DEVICE:=DEV6 FILE:=FIL6 BUFFERS=2.USER PROG LUN 6
/ASSIGN >B8 DISC1  LUNO:=LUN8 FILE=(TEMP,SCRL) NEW BUFFERS=1;
/            LINKSEQ ACCESS=(ANY,ANY,ANY,ANY);
/            ALLOCATE=(1,3ØØ,32,1Ø)                      . USER SCRATCH FILE
/END
```

B-7

```
.# CREATE FTNLGO,COMMENT,"FORTRAN COMPILE, LINK,AND GO"
/REPLACE FTNLGO     . FORTRAN COMPILE, LINK, AND GO .
/EXEC OBJ=(1,SYSTEM,FTN) MEM=(300,10000,1000) PRTY=(1,15);
/          TIME=-1, MEM:=MEMC
/ASSIGN  0 DUMMY DEVICE:=DMSG SHARE                          . SYSTEM MESSAGE
/ASSIGN  5 DISC1 DEVICE:=DSRC FILE:=FSRC BUFFERS=1           . SOURCE INPUT
/ASSIGN  6 SC    DEVICE:=DLST1 FILE:=FLST BUFFERS=1          . SOURCE LIST/ERROR
/ASSIGN  7 DISC1 FILE=(TEMP,PHASE1) NEW BUFFERS=1 LINKSEQ;   . INTERMED OBJECT
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,64,30)  . SOURCE SCRATCH
/EXEC OBJ=(1,SYSTEM,FTNPS2) MEM=(300,8000,1000) PRTY=(1,15);
/          TIME=-1 MEM:=MEMC
/ASSIGN  0 DUMMY                                             . SYSTEM MESSAGE
/ASSIGN  6 SC    DEVICE:=DLST2                               . ERROR MESSAGE
/ASSIGN  7 DISC1 FILE=(TEMP,OBJECT) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,64,10)  . OBJECT OUTPUT
/ASSIGN  8 DISC1 FILE=(TEMP,PHASE1) BUFFERS=1                . INTERMED OBJECT
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(300,12000,3000) PRTY=(1,15);
/          TIME=-1 MEM:=MEML
/ASSIGN  5 DISC1 FILE=(TEMP,OBJECT) DELETE BUFFERS=1         . PRIMARY INPUT/CON
/ASSIGN  6 SC    DEVICE:=DLSTL FILE:=FLST BUFFERS=1          . LOADMAP LIST/ERR
/ASSIGN  8 DISC1 FILE=(TEMP,LM) NEW BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,32,10); . LOAD MOD OUTPUT
/          LRECL=64
/ASSIGN  9 DISC1 FILE=(SYSTEM,USRFTN) BUFFERS=2              . LIBRARY
/ASSIGN 10 DISC1 FILE=(TEMP,PHASE1) DELETE BUFFERS=1         . LINKSEQ SCRATCH
/ASSIGN 13 DISC1 FILE=(TEMP,SCRR) NEW DELETE BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10);. RELREC SCRATCH
/          LRECL=100
/EXEC OBJ=(1,TEMP,LM) MEM=(300,8000,1000) PRTY=(1,15);
/          TIME=100 MEM:=MEMG TIME:=TIMG
/ASSIGN   0 SC    DEVICE:=DMSG                               . SYSTEM MESSAGE
/ASSIGN >B0 SC    DEVICE:=DEV0                               . USER LUN 0
/ASSIGN >B1 DUMMY DEVICE:=DEV1                               . USER LUN 1
/ASSIGN >B5 SC    DEVICE:=DEV5 FILE:=FIL5 BUFFERS=2          . USER LUN 5=INPUT
/ASSIGN >B6 SC    DEVICE:=DEV6 FILE:=FIL6 BUFFERS=2          . USER LUN 6=OUTPUT
/ASSIGN >B8 DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,32,10) . USER SCRATCH FILE
/END
.# CREATE FTNLNK,COMMENT,"FORTRAN COMPILE AND LINK    "
/REPLACE FTNLNK     . FORTRAN COMPILE AND LINK .
/EXEC OBJ=(1,SYSTEM,FTN) MEM=(300,10000,1000) PRTY=(1,15);
/          TIME=-1, MEM:=MEMC
/ASSIGN  0 DUMMY DEVICE:=DMSG SHARE                          . SYSTEM MESSAGE
/ASSIGN  5 DISC1 DEVICE:=DSRC FILE:=FSRC BUFFERS=1           . SOURCE INPUT
/ASSIGN  6 SC    DEVICE:=DLST1 FILE:=FLST BUFFERS=1          . SOURCE LIST/ERROR
/ASSIGN  7 DISC1 FILE=(TEMP,PHASE1) NEW BUFFERS=1 LINKSEQ;   . INTERMED OBJECT
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,64,30)  . SOURCE SCRATCH
/EXEC OBJ=(1,SYSTEM,FTNPS2) MEM=(300,8000,1000) PRTY=(1,15);
/          TIME=-1 MEM:=MEMC
/ASSIGN  0 DUMMY                                             . SYSTEM MESSAGE
/ASSIGN  6 SC    DEVICE:=DLST2                               . ERROR MESSAGE
/ASSIGN  7 DISC1 FILE=(TEMP,OBJECT) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,64,10)  . OBJECT OUTPUT
/ASSIGN  8 DISC1 FILE=(TEMP,PHASE1) BUFFERS=1                . INTERMED OBJECT
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(300,12000,3000) PRTY=(1,15);
/          TIME=-1 MEM:=MEML
/ASSIGN  5 DISC1 FILE=(TEMP,OBJECT) DELETE BUFFERS=1         . PRIMARY INPUT/CON
/ASSIGN  6 SC    DEVICE:=DLSTL FILE:=FLST BUFFERS=1          . LOADMAP LIST/ERR
/ASSIGN  8 DISC1 DEVICE:=DLM FILE=(USER01,GO) FILE:=FLM;
/          REPLACE:=RLM BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=CLM;
/          ALLOCATE=(10,0,32,10) ALLOCATE:=LLM LRECL=64      . LOAD MOD OUTPUT
/ASSIGN  9 DISC1 FILE=(SYSTEM,USRFTN) BUFFERS=2              . LIBRARY
/ASSIGN 10 DISC1 FILE=(TEMP,PHASE1) DELETE BUFFERS=1         . LINKSEQ SCRATCH
/ASSIGN 13 DISC1 FILE=(TEMP,SCRR) NEW DELETE BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10);. RELREC SCRATCH
/          LRECL=100
/END
```

```
.# CREATE FTNPS1,COMMENT,"FORTRAN PHASE 1 COMPILE     "
/REPLACE FTNPS1      . FORTRAN PHASE 1 COMPILE .
/EXEC OBJ=(1,SYSTEM,FTN) MEM=(300,8000,1000) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  0 DUMMY DEVICE:=DMSG SHARE                          . SYSTEM MESSAGE
/ASSIGN  5 DISC1 DEVICE:=DSRC FILE:=FSRC BUFFERS=1           . SOURCE INPUT
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.SOURCE LIST/ERROR
/ASSIGN  7 DISC1 DEVICE:=DINT FILE:=FINT BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(1,0,64,30);    . SOURCE SCRATCH
/          NEW:=NINT REPLACE:=RINT ACCESS:=CINT ALLOCATE:=LINT
/END
.# CREATE FTNPS2,COMMENT,"FORTRAN PHASE 2 COMPILE     "
/REPLACE FTNPS2      . FORTRAN PHASE 2 COMPILE .
/EXEC OBJ=(1,SYSTEM,FTNPS2) MEM=(300,8000,1000) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  0 DUMMY DEVICE:=DMSG SHARE                          . SYSTEM MESSAGE
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.ERROR MMSSG&5
/ASSIGN  7 DISC1 DEVICE:=DOBJ FILE:=FOBJ BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(1,0,64,10);    . OBJECT OUTPUT
/          NEW:=NOBJ REPLACE:=ROBJ ACCESS:=COBJ ALLOCATE:=LOBJ
/ASSIGN  8 DISC1 DEVICE:=DINT FILE:=FINT BUFFERS=1           . INTERMED OBJECT
/END
.# CREATE HELP  ,COMMENT,"H-E-L-P OPERATOR            "
/REPLACE HELP        . H-E-L-P OPERATOR .
/EXEC OBJ=(1,SYSTEM,DXCOPY) MEM=(300,3700,1500) PRTY=(1,15);
/           TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  5 DUMMY                                             . CONTROL/MESSAGE
/ASSIGN  6 SC    DEVICE:=DLST                                . LISTING
/ASSIGN  7 DUMMY                                             . OUTPUT
/ASSIGN  8 DISC1 FILE=(SYSTEM,HELP) BUFFERS=1                . INPUT
/END
.# CREATE INITSP,COMMENT,"INITIAL BATCH OUT SPOOL FILE"
/REPLACE INITSP      . INITIALIZE BATCH OUTPUT SPOOLER FILE .
/EXEC OBJ=(1,SYSTEM,INITSP) MEM=(300,300,700) PRTY=(1,5);
/          TIME=-1 PRTY:=PRI
/ASSIGN  6 DISC1 DEVICE:=DSOQ FILE=(SYSTEM,SOQ) REPLACE;
/          BUFFERS=1 RELREC ACCESS=(CREAT,CREAT,CREAT,CREAT);
/          ALLOCATE=(1,0,32,1) LRECL=64                      . SOQ FILE
/END
.# CREATE IPLINK,COMMENT,"LINK IPL PROGRAM            "
/REPLACE IPLINK      . LINK IPL PROGRAM .
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(300,15000,6000) PRTY=(1,2);
/          TIME=-1 MEM:=MEM
/ASSIGN  1 DISC1 DEVICE:=DIPL FILE=(SYSTEM,DXLPF) FILE:=FIPL;
/          BUFFERS=3 BUFFERS:=BIPL                           . IPL OBJECT IN
/ASSIGN  5 DISC1 DEVICE:=DIN  FILE:=FIN  BUFFERS=1           . PRIMARY INPUT/CON
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST BUFFERS=1           . LOADMAP LIST/ERR
/ASSIGN  8 DISC1 DEVICE:=DLM  FILE=(USER01,IPL) FILE:=FLM;
/          REPLACE:=RLM BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=CLM;
/          ALLOCATE=(6,0,32,6) ALLOCATE:=LLM LRECL=64        . IPL LOAD MOD OUT
/ASSIGN  9 DISC1 DEVICE:=DLIB FILE=(SYSTEM,USRPLX) FILE:=FLIB;
/          BUFFERS=2                                         . LIBRARY FILE
/ASSIGN 10 DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,256,10) . LINKSEQ SCRATCH
/ASSIGN 11 DUMMY
/ASSIGN 12 DUMMY                                             . SYS EXT DEFS OPT
/ASSIGN 13 DISC1 FILE=(TEMP,SCRR) NEW BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(2,300,128,2);
/          LRECL=100                                         . RELREC SCRATCH
/END
```

```
.# CREATE ITS   ,COMMENT,"INTERACTIVE TERMINAL SUBSYS "
/REPLACE ITS          . INTERACTIVE TERMINAL SUBSYSTEM .
/EXEC OBJ=(1,SYSTEM,ITS) MEM=(300,8000,2000) PRTY=(2,1);
/          PRIV TIME=-1 OBJ:=OBJ MEM:=MEM PRTY:=PRI PROT:=PRO
/ASSIGN  1 DUMMY DEVICE:=T1  SHARE:=S1                       , TERMINAL 1
/ASSIGN  3 DUMMY DEVICE:=T2  SHARE:=S2                       , TERMINAL 2
/ASSIGN  5 DUMMY DEVICE:=T3  SHARE:=S3                       , TERMINAL 3
/ASSIGN  7 DUMMY DEVICE:=T4  SHARE:=S4                       , TERMINAL 4
/ASSIGN  9 DUMMY DEVICE:=T5  SHARE:=S5                       , TERMINAL 5
/ASSIGN 11 DUMMY DEVICE:=T6  SHARE:=S6                       , TERMINAL 6
/ASSIGN 13 DUMMY DEVICE:=T7                                  , TERMINAL 7
/ASSIGN 15 DUMMY DEVICE:=T8                                  , TERMINAL 8
/ASSIGN 17 DUMMY DEVICE:=T9                                  , TERMINAL 9
/ASSIGN 19 DUMMY DEVICE:=T10                                 . TERMINAL 10
/END
.# CREATE JCL   ,COMMENT,"CREATE JCL PROCEDURE         "
/REPLACE JCL          . CREATE JCL PROCEDURE .
/EXEC OBJ=(1,SYSTEM,JCLTRN) MEM=(300,7550,1000) PRTY=(1,15);
/          TIME=-1 MEM:=MEM PRTY:=PRI
/ASSIGN  1 SC    DEVICE:=DSRC FILE:=FSRC BUFFERS=1           , SOURCE INPUT
/ASSIGN  2 SC    DEVICE:=DERR SHARE:=SERR                    . ERROR MESSAGE
/ASSIGN  3 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.SOURCE LISTING
/ASSIGN  4 DISC1 DEVICE:=DOBJ FILE=(SYSTEM,SJCBFL,AB);
/          FILE:=FOBJ REPLACE:=ROBJ BUFFERS=2 INDEXED;
/          ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=COBJ;
/          ALLOCATE=(1,0,96,20) ALLOCATE:=LOBJ KEYLEN=6      . OBJECT OUT FILE
/END
.# CREATE JCLUP ,COMMENT,"UPDATE JCL SOURCE AND BINARY"
/REPLACE JCLUP        . UPDATE JCL SOURCE AND BINARY FILES .
/EXEC OBJ=(1,SYSTEM,JCLTRN) MEM=(300,7550,1000) PRTY=(1,15);
/           TIME=-1 MEM:=MEM
/ASSIGN   1 DISC1 FILE=(SYSTEM,JCWORK) BUFFERS=1             . SOURCE INPUT
/ASSIGN   2 SC    DEVICE:=DERR                               , ERROR MESSAGE
/ASSIGN   3 SC    DEVICE:=DLST                               . SOURCE LISTING
/ASSIGN  4 DISC1 DEVICE:=DOBJ FILE=(SYSTEM,SJCBFL,AB);
/           FILE:=FOBJ REPLACE:=ROBJ BUFFERS=2 INDEXED;
/           ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=COBJ;
/           ALLOCATE=(1,0,96,32) ALLOCATE:=LOBJ KEYLEN=6     . OBJECT OUT FILE
/EXEC OBJ=(1,SYSTEM,SMR) MEM=(300,11500,2000) PRTY=(1.15);
/           TIME=-1 MEM:=MEM
/ASSIGN   0 DUMMY                                            , ERROR/USER MSG
/ASSIGN   4 DISC1 FILE=(SYSTEM,JCLCUP) DELETE BUFFERS=1      , CONTROL
/ASSIGN   6 DUMMY                                            , LISTING
/ASSIGN >15 DISC1 FILE=(SYSTEM,JCLSRC) BUFFERS=1             , OLD LIBRARY FILE
/ASSIGN >22 DUMMY                                            . COMPILE OUT FILE
/ASSIGN >25 DISC1 FILE=(TEMP,JCLFIL) NEW LINKSEQ BUFFERS=1;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(7,300,256,30) , NEW LIBRARY FILE
/ASSIGN >26 DUMMY                                            , JCL UPDAT CON OUT
/ASSIGN >35 DISC1 FILE=(SYSTEM,JCWORK) BUFFERS=1             . INCLUDE
/EXEC OBJ=(1,SYSTEM,SMR) MEM=(300,11500,2000) PRTY=(1.15);
/           TIME=-1 MEM:=MEM
/ASSIGN   0 DUMMY                                            , ERROR/USER MSG
/ASSIGN   4 DUMMY                                            , CONTROL
/ASSIGN   6 DUMMY                                            , LISTING
/ASSIGN >15 DISC1 FILE=(TEMP,JCLFIL) BUFFERS=1               , OLD LIBRARY FILE
/ASSIGN >22 DUMMY                                            , COMPILE OUT FILE
/ASSIGN >25 DISC1 FILE=(SYSTEM,JCLSRC) BUFFERS=1             , NEW LIBRARY FILE
/ASSIGN >26 DUMMY                                            . JCL UPDAT CON OUT
/END
```

```
.# CREATE LIBBLD,COMMENT,"BUILD LIBRARY FILE         "
/REPLACE LIBBLD      . BUILD LIBRARY FILE .
/EXEC OBJ=(1,SYSTEM,LIBBLD) MEM=(300,2000,1000) PRTY=(1,15);
/            TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  5 MT1   DEVICE:=DOBJ FILE:=FOBJ BUFFERS=1          . OBJECT INPUT
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST SHARE:=SLST BUFFERS=1.IDT/DEF LISTING
/ASSIGN  9 DISC1 DEVICE:=DLIB FILE:=FLIB REPLACE:=RLIB;
/          BUFFERS=2 INDEXED ACCESS=(ANY,ANY,ANY,ANY);
/          ACCESS:=CLIB ALLOCATE=(1,0,128,20) ALLOCATE:=LLIB;
/          KEYLEN=8                                         . OUTPUT LIB FILE
/END
.# CREATE LINKUP,COMMENT,"LINK MOD AND UPDATE L M FILE"
/REPLACE LINKUP      . LINK MODULE AND UPDATE LOAD MODULE FILE .
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(300,12000,3000) PRTY=(1,1);
/          TIME=-1 MEM:=MEML
/ASSIGN  1 DUMMY DEVICE:=DOBJ FILE:=FOBJ BUFFERS=2          . SECONDARY OBJ IN
/ASSIGN  5 DISC1 DEVICE:=DIN FILE:=FIN BUFFERS=1            . PRIMARY INPUT/CON
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST BUFFERS=1          . LOADMAP LIST/ERR
/ASSIGN  8 DISC1 FILE=(TEMP,LM) NEW BUFFERS=1 RELREC LRECL=64;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,32,10) . LOAD MOD OUTPUT
/ASSIGN  9 DISC1 DEVICE:=DPLX FILE=(SYSTEM,USRPLX) FILE:=FPLX;
/          BUFFERS=2                                        . PLEXUS LIBRARY
/ASSIGN 10 DISC1 FILE=(TEMP,SCRL) NEW DELETE BUFFERS=1;
/          LINKSEQ ACCESS=(ANY,ANY,ANY,ANY);                . LINKSEQ SCRATCH
/          ALLOCATE=(10,300,256,30)
/ASSIGN 11 DISC1 DEVICE:=DEXT FILE=(SYSTEM,DXEXTD) FILE:=FEXT;
/          BUFFERS=1                                        . SYSTEM EXT DEFS
/ASSIGN 13 DISC1 FILE=(TEMP,SCRR) NEW DELETE BUFFERS=1 RELREC;
/          ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10);
/          LRECL=100                                        . RELREC SCRATCH
/EXEC OBJ=(1,SYSTEM,LMUPDT) MEM=(300,3000,500) PRTY=(1,1);
/          TIME=-1 MEM:=MEMU
/ASSIGN  4 SC    DEVICE:=DMSG                               . SYSTEM MESSAGE
/ASSIGN  5 SC    DEVICE:=DCON SHARE:=SCON                   . CONTROL INPUT
/ASSIGN  6 DISC1 FILE=(TEMP,LM) BUFFERS=1                   . LOAD MODULE INPUT
/ASSIGN  7 DISC1 DEVICE:=DUPD FILE:=FUPD BUFFERS=1          . UPDATE FILE
/END
.# CREATE LMUPDT,COMMENT,"LOAD MODULE UPDATE          "
/REPLACE LMUPDT      . LOAD MODULE UPDATE .
/EXEC OBJ=(1,SYSTEM,LMUPDT) MEM=(300,3000,500) PRTY=(1,1);
/          TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  4 SC    DEVICE:=DMSG                               . SYSTEM MESSAGE
/ASSIGN  5 SC    DEVICE:=DCON FILE:=FCON BUFFERS=1          . CONTROL INPUT
/ASSIGN  6 DISC1 DEVICE:=DLM FILE:=FLM BUFFERS=1            . LOAD MODULE INPUT
/ASSIGN  7 DISC1 DEVICE:=DUPD FILE:=FUPD BUFFERS=1          . UPDATE FILE
/END

.# CREATE LPFBLD,COMMENT,"UPDATE LINKABLE PARTS FILE  "
/REPLACE LPFBLD      . UPDATE LINKABLE PARTS FILE .
/EXEC OBJ=(1,SYSTEM,LPFBLD) MEM=(300,3300,1000) PRTY=(1,2);
/            TIME=-1 OBJ:=OBJ MEM:=MEM
/ASSIGN   0 DUMMY DEVICE:=DCON                              . CONTROL
/ASSIGN   5 DISC1 DEVICE:=DOBJ FILE=(USER01,ASMOUT) FILE:=FOBJ;
/           BUFFERS=1                                       . OBJECT INPUT
/ASSIGN   6 SC    DEVICE:=DLST                              . LISTING
/ASSIGN   7 DUMMY DEVICE:=DEXT FILE:=FEXT REPLACE:=REXT;
/           BUFFERS=1 LINKSEQ ACCESS=(ANY,CREAT,CREAT,CREAT);
/           ACCESS:=CEXT ALLOCATE=(1,0,128,20) ALLOCATE:=LEXT . EXTRACT FILE
/ASSIGN   9 DISC1 DEVICE:=DUPD FILE:=FUPD REPLACE:=RUPD;
/           BUFFERS=2 INDEXED ACCESS=(CREAT,CREAT,CREAT,CREAT);
/           ACCESS:=CUPD ALLOCATE=(1,0,256,10) ALLOCATE:=LUPD;
/           KEYLEN=6                                        . UPDATE FILE
/END
```

```
.# CREATE LSTEDT,COMMENT,"LIST EDIT FILE            "
/REPLACE LSTEDT       . LIST EDIT FILE .
/EXEC OBJ=(1,SYSTEM,LSTEDT) MEM=(300,1700,650) PRTY=(1,15);
/          TIME=-1 MEM;=MEM PRTY;=PRI TIME;=TIM
/ASSIGN 10 SC     DEVICE;=DLST                              . SOURCE OUTPUT
/ASSIGN 20 DISC1 DEVICE;=DIN FILE;=FIN BUFFERS=1            . SOURCE INPUT FILE
/END
.# CREATE PIALGO,COMMENT,"PLEXUS COMP,ILT,ASM,LINK,GO "
/REPLACE PIALGO      . PLEXUS COMPILE, ILT, ASSEMBLE, LINK, AND GO .
/EXEC OBJ=(1,SYSTEM,PLEXUS) MEM=(300,>94C0,2700) PRTY=(1,2);
/           TIME=7200 MEM;=MEM
/ASSIGN   0 SC     DEVICE;=DERR                             . ERROR MESSAGE
/ASSIGN   5 DISC1 DEVICE;=DSRC FILE;=FSRC BUFFERS=1         . SOURCE INPUT
/ASSIGN   6 SC     DEVICE;=DLSTC                            . SOURCE LISTING
/ASSIGN >22 DISC1 FILE=(TEMP,DATA) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,160,10). DATA DIV INT CODE
/ASSIGN >23 DISC1 FILE=(TEMP,PROC) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,160,10). PROC DIV INT CODE
/ASSIGN >25 DUMMY
/EXEC OBJ=(1,SYSTEM,ILT980) MEM=(300,>94C0,2500) PRTY=(1,2);
/           TIME=7200 MEM;=MEM
/ASSIGN   0 SC     DEVICE;=DERR                             . ERROR MESSAGE
/ASSIGN   6 SC     DEVICE;=DLSTI                            . PRINTOUT
/ASSIGN >12 DISC1 FILE=(TEMP,DATA) DELETE BUFFERS=1         . DATA DIV INT CODE
/ASSIGN >13 DISC1 FILE=(TEMP,PROC) DELETE BUFFERS=1         . PROC DIV INT CODE
/ASSIGN >14 DISC1 FILE=(TEMP,SCRR) NEW BUFFERS=1 RELREC;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10);
/           LRECL=64                                        . REL REC SCRATCH
/ASSIGN >23 DISC1 FILE=(TEMP,ILTOUT) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,30). ILT OUTPUT
/ASSIGN >24 DISC1 FILE=(TEMP,SCRR) BUFFERS=1                . REL REC SCRATCH
/ASSIGN >30 DISC1 FILE=(SYSTEM,MDEF) SHARE BUFFERS=1        . MACHINE DESCRIPT
/EXEC OBJ=(1,SYSTEM,ASMBLR) MEM=(300,20000,2000) PRTY=(1,2);
/           TIME=1800
/ASSIGN   0 DUMMY                                           . SYSTEM MESSAGE
/ASSIGN   4 DUMMY                                           . CONTROL/MESSAGE
/ASSIGN   5 DISC1 FILE=(TEMP,ILTOUT) DELETE BUFFERS=1       . ASSEMBLY INPUT
/ASSIGN   6 SC     DEVICE;=DLSTA                            . ASSEMBLY LISTING
/ASSIGN   7 DISC1 FILE=(TEMP,OBJECT) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,20). OBJECT OUTPUT
/ASSIGN  16 DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,256,30). ASSEMBLY SCRATCH
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(300,20000,3000) PRTY=(1,2);
/           TIME=1800
/ASSIGN   5 DISC1 FILE=(TEMP,OBJECT) DELETE BUFFERS=1       . PRIMARY INPUT/CON
/ASSIGN   6 SC     DEVICE;=DLSTL                            . LOADMAP LIST/ERR
/ASSIGN   8 DISC1 FILE=(TEMP,LM) NEW BUFFERS=1 RELREC;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,32,10);
/           LRECL=64                                        . LOAD MOD OUTPUT
/ASSIGN   9 DISC1 FILE=(SYSTEM,USRPLX) BUFFERS=2            . LIBRARY
/ASSIGN  10 DISC1 FILE=(TEMP,SCRL) DELETE BUFFERS=1         . LINKSEQ SCRATCH
/ASSIGN  13 DISC1 FILE=(TEMP,SCRR) DELETE BUFFERS=1         . RELREC  SCRATCH
/EXEC OBJ=(1,TEMP,LM) MEM=(300,20000,3000) PRTY=(1,2);
/           TIME=1800
/ASSIGN   0 SC     DEVICE;=DERR                             . ERROR MESSAGE
/ASSIGN   5 DISC1 DEVICE;=DIN1   FILE;=FIN1   BUFFERS=1     . PRIMARY INPUT
/ASSIGN   6 SC     DEVICE;=DOUT1                            . PRIMARY OUTPUT
/ASSIGN >12 DUMMY DEVICE;=DIN2   FILE;=FIN2   BUFFERS=1     . SECONDARY INPUT
/ASSIGN >22 DUMMY DEVICE;=DOUT2 FILE;=FOUT2 BUFFERS=1       . SECONDARY INPUT
/ASSIGN >30 DUMMY DEVICE;=DF0    FILE;=FF0    BUFFERS=1     . FILE 0
/ASSIGN >31 DUMMY DEVICE;=DF1    FILE;=FF1    BUFFERS=1     . FILE 1
/END
```

```
.# CREATE PIALNK,COMMENT,"PLEXUS COMP,ILT,ASM,AND LINK"
/REPLACE PIALNK     . PLEXUS COMPILE, ILT, ASSEMBLE, AND LINK .
/EXEC OBJ=(1,SYSTEM,PLEXUS) MEM=(300,>94C0,2700) PRTY=(1,2);
/           TIME=7200 MEM:=MEM                                    ;
/ASSIGN   0 SC     DEVICE:=DERR                                   ; ERROR MESSAGE
/ASSIGN   5 DISC1 DEVICE:=DSRC FILE:=FSRC BUFFERS=1               ; SOURCE INPUT
/ASSIGN   6 SC     DEVICE:=DLSTC                                  . SOURCE LISTING
/ASSIGN >22 DISC1 FILE=(TEMP,DATA) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,160,10);. DATA DIV INT CODE
/ASSIGN >23 DISC1 FILE=(TEMP,PROC) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,160,10);. PROC DIV INT CODE
/ASSIGN >25 DUMMY
/EXEC OBJ=(1,SYSTEM,ILT980) MEM=(300,>94C0,2500) PRTY=(1,2);
/           TIME=7200 MEM:=MEM                                    ;
/ASSIGN   0 SC     DEVICE:=DERR                                   ; ERROR MESSAGE
/ASSIGN   6 SC     DEVICE:=DLSTI                                  ; PRINTOUT
/ASSIGN >12 DISC1 FILE=(TEMP,DATA) DELETE BUFFERS=1               ; DATA DIV INT CODE
/ASSIGN >13 DISC1 FILE=(TEMP,PROC) DELETE BUFFERS=1               . PROC DIV INT CODE
/ASSIGN >14 DISC1 FILE=(TEMP,SCRR) NEW BUFFERS=1 RELREC;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10);
/           LRECL=64                                              . REL REC SCRATCH
/ASSIGN >23 DISC1 FILE=(TEMP,ILTOUT) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,30);. ILT OUTPUT
/ASSIGN >24 DISC1 FILE=(TEMP,SCRR) BUFFERS=1                      . REL REC SCRATCH
/ASSIGN >30 DISC1 FILE=(SYSTEM,MDEF) SHARE BUFFERS=1              . MACHINE DESCRIPT
/EXEC OBJ=(1,SYSTEM,ASMBLR) MEM=(300,20000,2000) PRTY=(1,2);
/           TIME=1800                                             ;
/ASSIGN   0 DUMMY                                                 ; SYSTEM MESSAGE
/ASSIGN   4 DUMMY                                                 ; CONTROL/MESSAGE
/ASSIGN   5 DISC1 FILE=(TEMP,ILTOUT) DELETE BUFFERS=1             . ASSEMBLY INPUT
/ASSIGN   6 SC     DEVICE:=DLSTA                                  . ASSEMBLY LISTING
/ASSIGN   7 DISC1 FILE=(TEMP,OBJECT) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,20);. OBJECT OUTPUT
/ASSIGN  16 DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,256,30);. ASSEMBLY SCRATCH
/EXEC OBJ=(1,SYSTEM,DXOLE) MEM=(300,20000,3000) PRTY=(1,2);
/           TIME=1800                                             ;
/ASSIGN   5 DISC1 FILE=(TEMP,OBJECT) DELETE BUFFERS=1             ; PRIMARY INPUT/CON
/ASSIGN   6 SC     DEVICE:=DLSTL                                  . LOADMAP LIST/ERR
/ASSIGN   8 DISC1 DEVICE:=DLM FILE=(USER01,GO) FILE:=FLM;
/         REPLACE:=RLM BUFFERS=1 RELREC;
/         ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=CLM;
/         ALLOCATE=(10,0,32,10) ALLOCATE:=LLM LRECL=64            ; LOAD MOD OUTPUT
/ASSIGN   9 DISC1 FILE=(SYSTEM,USRPLX) BUFFERS=2                  ; LIBRARY
/ASSIGN  10 DISC1 FILE=(TEMP,SCRL) DELETE BUFFERS=1               ; LINUSEQ SCRATCH
/ASSIGN  13 DISC1 FILE=(TEMP,SCRR) DELETE BUFFERS=1               . RELREC  SCRATCH
/END
.# CREATE PLEXGO,COMMENT,"EXECUTE PLEXUS GENERATED L M"
/REPLACE PLEXGO     . EXECUTE PLEXUS GENERATED LOAD MODULE .
/EXEC OBJ=(1,USER01,GO) MEM=(300,12000,1000) PRTY=(4,15);
/           TIME=1800 OBJ:=OBJ MEM:=MEM PRTY:=PRI TIME:=TIM       ;
/ASSIGN   0 SC     DEVICE:=DERR                                   ; ERROR MESSAGE
/ASSIGN   5 DISC1 DEVICE:=DIN1   FILE:=FIN1   BUFFERS=1           ; PRIMARY INPUT
/ASSIGN   6 SC     DEVICE:=DOUT1                                  ; PRIMARY OUTPUT
/ASSIGN >12 DUMMY DEVICE:=DIN2   FILE:=FIN2 BUFFERS=1             ; SECONDARY INPUT
/ASSIGN >13 DUMMY DEVICE:=DIN3   FILE:=FIN3   BUFFERS=1           ; TERTIARY INPUT
/ASSIGN >22 DUMMY DEVICE:=DOUT2 FILE:=FOUT2 BUFFERS=1             ; SECONDARY OUTPUT
/ASSIGN >23 DUMMY DEVICE:=DOUT3 FILE:=FOUT3 BUFFERS=1             ; TERTIARY OUTPUT
/ASSIGN >30 DUMMY DEVICE:=DF0    FILE:=FF0    BUFFERS=1           ; FILE 0
/ASSIGN >31 DUMMY DEVICE:=DF1    FILE:=FF1    BUFFERS=1           . FILE 1
/END
```

```
.* CREATE PLEXIA,COMMENT,"PLEXUS COMPILE, ILT, AND ASM"
/REPLACE PLEXIA     . PLEXUS COMPILE, ILT, AND ASSEMBLE .
/EXEC OBJ=(1,SYSTEM,PLEXUS) MEM=(300,>94C0,2700) PRTY=(1,2);
/           TIME=7200 MEM:=MEM TIME:=TIM
/ASSIGN   0 SC     DEVICE:=DERR                                 . ERROR MESSAGE
/ASSIGN   5 DISC1  DEVICE:=DSRC FILE:=FSRC BUFFERS=1            . SOURCE INPUT
/ASSIGN   6 SC     DEVICE:=DLSTC FILE:=FLST BUFFERS=1           . SOURCE LISTING
/ASSIGN >22 DISC1 FILE=(TEMP,DATA) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,160,10).  DATA DIV INT CODE
/ASSIGN >23 DISC1 FILE=(TEMP,PROC) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,160,10).  PROC DIV INT CODE
/ASSIGN >25 DUMMY
/EXEC OBJ=(1,SYSTEM,ILT980) MEM=(300,>94C0,2500) PRTY=(1,2);
/          TIME=7200 MEM:=MEM TIME:=TIM
/ASSIGN   0 SC     DEVICE:=DERR                                 . ERROR MESSAGE
/ASSIGN   6 SC     DEVICE:=DLSTI FILE:=FLST BUFFERS=1           . PRINTOUT
/ASSIGN >12 DISC1 FILE=(TEMP,DATA) DELETE BUFFERS=1             . DATA DIV INT CODE
/ASSIGN >13 DISC1 FILE=(TEMP,PROC) DELETE BUFFERS=1             . PROC DIV INT CODE
/ASSIGN >14 DISC1 FILE=(TEMP,SCRR) NEW BUFFERS=1 RELREC;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10);
/           LRECL=64                                            . REL REC SCRATCH
/ASSIGN >23 DISC1 FILE=(TEMP,ILTOUT) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,30).  ILT OUTPUT
/ASSIGN >24 DISC1 FILE=(TEMP,SCRR) DELETE BUFFERS=1             . REL REC SCRATCH
/ASSIGN >30 DISC1 FILE=(SYSTEM,MDEF) SHARE BUFFERS=1            . MACHINE DESCRIPT
/EXEC OBJ=(1,SYSTEM,ASMBLR) MEM=(300,20000,2000) PRTY=(1,2);
/           TIME=1800
/ASSIGN   0 DUMMY                                               . SYSTEM MESSAGE
/ASSIGN   4 DUMMY                                               . CONTROL/MESSAGE
/ASSIGN   5 DISC1 FILE=(TEMP,ILTOUT) BUFFERS=1                  . ASSEMBLY INPUT
/ASSIGN   6 SC    DEVICE:=DLSTA FILE:=FLST BUFFERS=1            . ASSEMBLY LISTING
/ASSIGN   7 DISC1 DEVICE:=DOBJ FILE=(USER01,ASMOUT) FILE:=FOBJ;
/           REPLACE BUFFERS=1 LINKSEQ ACCESS=(ANY,ANY,ANY,ANY);
/           ACCESS:=COBJ ALLOCATE=(1,0,128,20) ALLOCATE:=LOBJ.  OBJECT OUTPUT
/ASSIGN  16 DISC1 FILE=(TEMP,SCRL) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,256,30).  ASSEMBLY SCRATCH
/END
```

```
.# CREATE PLEXUP,COMMENT,"PLEXUS GENERATED UPDATE LPF "
/REPLACE PLEXUP     . PLEXUS COMPILE, ILT, ASSEMBLE MODULE AND UPDATE LPF .
/EXEC OBJ=(1,SYSTEM,PLEXUS) MEM=(300,>94C0,2700) PRTY=(1,2);
/             TIME=7200 MEM:=MEM TIME:=TIM
/ASSIGN   0 SC     DEVICE:=DERR                               , ERROR MESSAGE
/ASSIGN   5 DISC1 DEVICE:=DSRC FILE:=FSRC BUFFERS=1           , SOURCE INPUT
/ASSIGN   6 SC     DEVICE:=DLSTC FILE:=FLST BUFFERS=1         . SOURCE LISTING
/ASSIGN >22 DISC1 FILE=(TEMP,DATA) NEW BUFFERS=1 LINKSEQ;
/              ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,160,10). DATA DIV INT CODE
/ASSIGN >23 DISC1 FILE=(TEMP,PROC) NEW BUFFERS=1 LINKSEQ;
/              ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,160,10). PROC DIV INT CODE
/ASSIGN >25 DUMMY
/EXEC OBJ=(1,SYSTEM,ILT980) MEM=(300,>94C0,2500) PRTY=(1,2);
/             TIME=7200 MEM:=MEM TIME:=TIM
/ASSIGN   0 SC     DEVICE:=DERR                               , ERROR MESSAGE
/ASSIGN   6 SC     DEVICE:=DLSTI FILE:=FLST BUFFERS=1         , PRINTOUT
/ASSIGN >12 DISC1 FILE=(TEMP,DATA) DELETE BUFFERS=1           , DATA DIV INT CODE
/ASSIGN >13 DISC1 FILE=(TEMP,PROC) DELETE BUFFERS=1           . PROC DIV INT CODE
/ASSIGN >14 DISC1 FILE=(TEMP,SCRR) NEW BUFFERS=1 RELREC;
/              ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10);
/              LRECL=64                                       . REL REC SCRATCH
/ASSIGN >23 DISC1 FILE=(TEMP,ILTOUT) NEW BUFFERS=1 LINKSEQ;
/              ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,30). ILT OUTPUT
/ASSIGN >24 DISC1 FILE=(TEMP,SCRR) DELETE BUFFERS=1           , REL REC SCRATCH
/ASSIGN >30 DISC1 FILE=(SYSTEM,MDEF) SHARE BUFFERS=1          . MACHINE DESCRIPT
/EXEC OBJ=(1,SYSTEM,ASMBLR) MEM=(300,20000,2000) PRTY=(1,2);
/              TIME=1800
/ASSIGN   0 DUMMY                                             , SYSTEM MESSAGE
/ASSIGN   4 DUMMY                                             . CONTROL/MESSAGE
/ASSIGN   5 DISC1 FILE=(TEMP,ILTOUT) DELETE BUFFERS=1         , ASSEMBLY INPUT
/ASSIGN   6 SC     DEVICE:=DLSTA FILE:=FLST BUFFERS=1         . ASSEMBLY LISTING
/ASSIGN   7 DISC1 FILE=(TEMP,ASMOUT) NEW BUFFERS=1 LINKSEQ;
/              ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,20). OBJECT OUTPUT
/ASSIGN  16 DISC1 FILE=(TEMP,SCRL) NEW DELETE BUFFERS=1;
/              LINKSEQ ACCESS=(ANY,ANY,ANY,ANY);
/              ALLOCATE=(10,300,256,30)                       . ASSEMBLY SCRATCH
/EXEC OBJ=(1,SYSTEM,LPFBLD) MEM=(300,8000,3000) PRTY=(1,2);
/              TIME=-1
/ASSIGN   0 DUMMY                                             , CONTROL
/ASSIGN   5 DISC1 DEVICE:=DOBJ FILE=(TEMP,ASMOUT) BUFFERS=1   , OBJECT INPUT
/ASSIGN   6 SC     DEVICE:=DLSTU FILE:=FLST BUFFERS=1         . UPDATE LISTING
/ASSIGN   9 DISC1 DEVICE:=DUPD FILE=(SYSTEM,DXLPF) FILE:=FUPD;
/              BUFFERS=2                                      . UPDATE FILE
/END
.# CREATE RESTLP,COMMENT,"RESTORE LPF FROM MT TO DISC "
/REPLACE RESTLP     . RESTORE LPF FROM MAG TAPE TO DISC .
/EXEC OBJ=(1,SYSTEM,DXCOPY) MEM=(300,4000,3000) PRTY=(1,2);
/             TIME=-1 MEM:=MEM
/ASSIGN  5 DUMMY DEVICE:=DCON                                 , CONTROL/MESSAGE
/ASSIGN  6 SC    DEVICE:=DLST FILE:=FLST BUFFERS=1            . LISTING
/ASSIGN  7 DISC1 DEVICE:=DOUT FILE=(SYSTEM,LPF) FILE:=FOUT;
/            REPLACE BUFFERS=2 BUFFERS:=BOUT INDEXED;
/            ACCESS=(ANY,ANY,ANY,ANY) ACCESS:=COUT;
/            ALLOCATE=(2,0,128,50) ALLOCATE:=LOUT KEYLEN=6    , OUTPUT
/ASSIGN  8 MT1   DEVICE:=DIN  FILE:=FIN BUFFERS=2             . INPUT
/END
```

```
.# CREATE SMR   ,COMMENT,"SOURCE MAINTENANCE ROUTINE  "
/REPLACE SMR        . SOURCE MAINTENANCE ROUTINE .
/EXEC OBJ=(1,SYSTEM,SMR) MEM=(3ØØ,115ØØ,5ØØØ) PRTY=(1,15);
/            TIME=-1 MEM:=MEM
/ASSIGN   Ø SC    DEVICE:=DMSG                               ; ERROR/USER MSG
/ASSIGN   4 SC    DEVICE:=DCON FILE:=FCON BUFFERS=1          ; CONTROL
/ASSIGN   6 DUMMY DEVICE:=DLST FILE:=FLST BUFFERS=1          ; LISTING
/ASSIGN >15 MT1   DEVICE:=DOLD FILE:=FOLD BUFFERS=1 LINKSEQ  : OLD LIBRARY FILE
/ASSIGN >22 DUMMY DEVICE:=DCOM FILE:=FCOM REPLACE:=RCOM;
/            BUFFERS=1 LINKSEQ ACCESS=(ANY,ANY,ANY,ANY);
/            ALLOCATE:=LCOM                                  : COMPILE OUT FILE
/ASSIGN >25 DUMMY DEVICE:=DNEW FILE:=FNEW REPLACE:=RNEW;
/            BUFFERS=1 LINKSEQ ACCESS=(ANY,ANY,ANY,ANY);
/            ALLOCATE:=LNEW                                  ; NEW LIBRARY FILE
/ASSIGN >26 DUMMY                                            ; JCL UPDAT CON OUT
/ASSIGN >35 DUMMY DEVICE:=DEV35 FILE:=FIL35 BUFFERS=2        ; INCLUDE LUN OPT 1
/ASSIGN >45 DUMMY DEVICE:=DEV45 FILE:=FIL45 BUFFERS=2        . INCLUDE LUN OPT 2
/END
.# CREATE SMRASM,COMMENT,"SMR, ASM, AND UPDATE LPF     "
/REPLACE SMRASM     . SMR, ASM, UPDATE LPF .
/EXEC OBJ=(1,SYSTEM,SMR) MEM=(3ØØ,115ØØ,5ØØØ) PRTY=(1,15);
/            TIME=-1
/ASSIGN   Ø DUMMY                                            ; ERROR/USER MSG
/ASSIGN   4 SC    DEVICE:=DCON FILE:=FCON BUFFERS=1          ; CONTROL
/ASSIGN   6 DUMMY DEVICE:=DLST                               ; LISTING
/ASSIGN >15 MT1   DEVICE:=DOLD FILE:=FOLD BUFFERS=1          . OLD LIBRARY FILE
/ASSIGN >22 DISC1 FILE=(TEMP,SMRWRK) NEW BUFFERS=1 LINKSEQ;
/            ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(2,3ØØ,128,2Ø) ; COMPILE OUT FILE
/ASSIGN >25 DUMMY                                            ; NEW LIBRARY FILE
/ASSIGN >26 DUMMY                                            ; JCL CON OUT FILE
/ASSIGN >35 DUMMY                                            ; INCLUDE LUN OPT
/ASSIGN >45 DUMMY                                            . INCLUDE LUN OPT
/EXEC OBJ=(1,SYSTEM,ASMBLR) MEM=(3ØØ,2ØØØØ,2ØØØ) PRTY=(1,2);
/            TIME=18ØØ
/ASSIGN   Ø DUMMY                                            ; SYSTEM MESSAGE
/ASSIGN   4 DUMMY                                            ; CONTROL/MESSAGE
/ASSIGN   5 DISC1 FILE=(TEMP,SMRWRK) DELETE BUFFERS=1        ; ASSEMBLY INPUT
/ASSIGN   6 SC    DEVICE:=DLST                               . ASSEMBLY LISTING
/ASSIGN   7 DISC1 FILE=(TEMP,ASMOUT) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(1Ø,3ØØ,128,3Ø). OBJECT OUTPUT
/ASSIGN  16 DISC1 FILE=(TEMP,SCRL) NEW DELETE BUFFERS=1;
/           LINKSEQ ACCESS=(ANY,ANY,ANY,ANY);
/           ALLOCATE=(1Ø,3ØØ,256,3Ø)                          : ASSEMBLY SCRATCH
/EXEC OBJ=(1,SYSTEM,LPFBLD) MEM=(3ØØ,8ØØØ,3ØØØ) PRTY=(1,2);
/           TIME=-1
/ASSIGN   Ø DUMMY                                            ; CONTROL
/ASSIGN   5 DISC1 FILE=(TEMP,ASMOUT) BUFFERS=1               ; OBJECT INPUT
/ASSIGN   6 SC    DEVICE:=DLST                               . UPDATE LISTING
/ASSIGN   9 DISC1 DEVICE:=DUPD FILE=(SYSTEM,DXLPF) FILE:=FUPD;
/           BUFFERS=2                                        . UPDATE FILE
/END
```

```
.* CREATE SMRPLX,COMMENT,"SMR,PLEXUS,ILT,ASM,UPDAT LPF"
/REPLACE SMRPLX     . SMR, PLEXUS, ILT, ASM, UPDATE LPF .
/EXEC OBJ=(1,SYSTEM,SMR) MEM=(300,11500,5000) PRTY=(1,15);
/             TIME=-1
/ASSIGN   0 DUMMY                                            . ERROR/USER MSG
/ASSIGN   4 SC     DEVICE:=DCON FILE:=FCON BUFFERS=1         . CONTROL
/ASSIGN   6 DUMMY DEVICE:=DLST                               . LISTING
/ASSIGN >15 MT1    DEVICE:=DOLD FILE:=FOLD BUFFERS=1         . OLD LIBRARY FILE
/ASSIGN >22 DISC1 FILE=(TEMP,SMRWRK) NEW BUFFERS=1 LINKSEQ;
/             ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(2,300,128,30) . COMPILE OUT FILE
/ASSIGN >25 DUMMY                                            . NEW LIBRARY FILE
/ASSIGN >26 DUMMY                                            . JCL CON OUT FILE
/ASSIGN >35 DUMMY                                            . INCLUDE LUN OPT
/ASSIGN >45 DUMMY                                            . INCLUDE LUN OPT
/EXEC OBJ=(1,SYSTEM,PLEXUS) MEM=(300,>94C0,2500) PRTY=(1,2);
/             TIME=7200 MEM:=MEM
/ASSIGN   0 DUMMY                                            . ERROR MESSAGE
/ASSIGN 5 DISC1 FILE=(TEMP,SMRWRK) BUFFERS=1                 . PRIMARY INPUT
/ASSIGN   6 SC     DEVICE:=DLST                              . SOURCE LISTING
/ASSIGN >22 DISC1 FILE=(TEMP,DATA) NEW BUFFERS=1 LINKSEQ;
/             ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,160,10). DATA DIV INT CODE
/ASSIGN >23 DISC1 FILE=(TEMP,PROC) NEW BUFFERS=1 LINKSEQ;
/             ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,160,10). PROC DIV INT CODE
/ASSIGN >25 DUMMY
/EXEC OBJ=(1,SYSTEM,ILT980) MEM=(300,>94C0,2500) PRTY=(1,2);
/             TIME=7200 MEM:=MEM
/ASSIGN   0 DUMMY                                            . ERROR MESSAGE
/ASSIGN   6 SC     DEVICE:=DLST                              . PRINTOUT
/ASSIGN >12 DISC1 FILE=(TEMP,DATA) DELETE BUFFERS=1          . DATA DIV INT CODE
/ASSIGN >13 DISC1 FILE=(TEMP,PROC) DELETE BUFFERS=1          . PROC DIV INT CODE
/ASSIGN >14 DISC1 FILE=(TEMP,SCRR) NEW BUFFERS=1 RELREC;
/             ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,10);
/             LRECL=64                                       . REL REC SCRATCH
/ASSIGN >23 DISC1 FILE=(TEMP,ILTOUT) NEW BUFFERS=1 LINKSEQ;
/             ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(10,300,128,30). ILT OUTPUT
/ASSIGN >24 DISC1 FILE=(TEMP,SCRR) DELETE BUFFERS=1          . REL REC SCRATCH
/ASSIGN >30 DISC1 FILE=(SYSTEM,MDEF) BUFFERS=1               . MACHINE DESCRIPT
/EXEC OBJ=(1,SYSTEM,ASMBLR) MEM=(300,20000,2000) PRTY=(1,2);
/             TIME=1800
/ASSIGN   0 DUMMY                                            . SYSTEM MESSAGE
/ASSIGN   4 DUMMY                                            . CONTROL/MESSAGE
/ASSIGN   5 DISC1 FILE=(TEMP,ILTOUT) DELETE BUFFERS=1        . ASSEMBLY INPUT
/ASSIGN   6 SC     DEVICE:=DLST                              . ASSEMBLY LISTING
/ASSIGN   7 DISC1 FILE=(TEMP,SMRWRK) BUFFERS=1               . OBJECT OUTPUT
/ASSIGN  16 DISC1 FILE=(TEMP,SCRL) NEW DELETE BUFFERS=1;
/             LINKSEQ ACCESS=(ANY,ANY,ANY,ANY);
/             ALLOCATE=(10,300,256,30)                       . ASSEMBLY SCRATCH
/EXEC OBJ=(1,SYSTEM,LPFBLD) MEM=(300,8000,3000) PRTY=(1,2);
/             TIME=-1
/ASSIGN   0 DUMMY                                            . CONTROL
/ASSIGN   5 DISC1 FILE=(TEMP,SMRWRK) BUFFERS=1               . OBJECT INPUT
/ASSIGN   6 SC     DEVICE:=DLST                              . UPDATE LISTING
/ASSIGN   9 DISC1 DEVICE:=DUPD FILE=(SYSTEM,DXLPF) FILE:=FUPD;
/             BUFFERS=2                                      . UPDATE FILE
/END
```

```
.# CREATE YANK  ,COMMENT,"FETCH JCL SEQUENCE           "
/REPLACE YANK       . FETCH JCL SEQUENCE .
/EXEC OBJ=(1,SYSTEM,SMR) MEM=(300,11500,5000) PRTY=(1,15);
/           TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN   0 DUMMY                                            ; ERROR/USER MSG
/ASSIGN   4 SC    DEVICE:=DCON                               ; CONTROL
/ ASSIGN 6 DUMMY DEVICE:=DLST                                ; LISTING
/ASSIGN >15 DISC1 FILE=(SYSTEM,JCLSRC) BUFFERS=1             . OLD LIBRARY FILE
/ASSIGN >22 DISC1 FILE=(TEMP,COMPIL) NEW BUFFERS=1 LINKSEQ;
/           ACCESS=(ANY,ANY,ANY,ANY) ALLOCATE=(1,0,256,3)    ; COMPILE OUT FILE
/ASSIGN >25 DUMMY                                            ; NEW LIBRARY FILE
/ASSIGN >26 DISC1 FILE=(SYSTEM,JCLCUP) REPLACE BUFFERS=1;
/           LINKSEQ ACCESS=(ANY,ANY,ANY,ANY);
/           ALLOCATE=(1,0,64,1)                              ; JCL UPDAT CON OUT
/EXEC OBJ=(1,SYSTEM,BLDEDT) MEM=(300,550,2000) PRTY=(1,15);
/           TIME=-1 MEM:=MEM PRTY:=PRI TIME:=TIM
/ASSIGN  10 DISC1 FILE=(TEMP,COMPIL) BUFFERS=1               ; SOURCE INPUT
/ASSIGN  20 DISC1 FILE=(SYSTEM,JCWORK) REPLACE BUFFERS=2;
/           INDEXED ACCESS=(ANY,ANY,ANY,ANY);
/           ALLOCATE=(1,0,256,3) KEYLEN=2                    ; SOURCE OUT FILE
/END
```

APPENDIX C
ADDING TO ITS

APPENDIX C
ADDING TO ITS

C.1 ITS INTERNAL STRUCTURE

NOTE

The ITS design includes the ability to support a polled terminal for future expansion. This feature does not apply to the hardware currently supported with DX980.

Before modifying ITS or adding application programs to run under ITS, the programmer must understand the structure of the subsystem. The subsystem consists of four main parts: Terminal Process Monitor, Terminal I/O subroutines, Supervisor, and individual application programs. Figure C-1 illustrates the interrelation of these components. The following paragraphs explain the operation of each component.

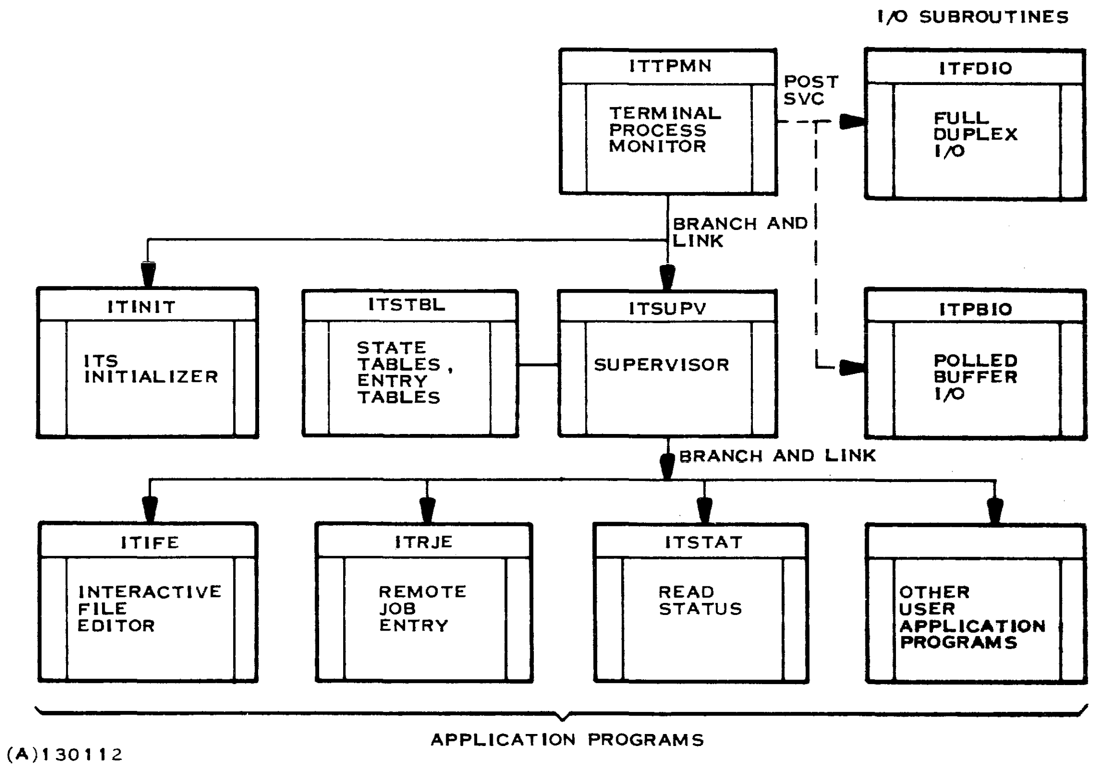


Figure C-1. ITS Components

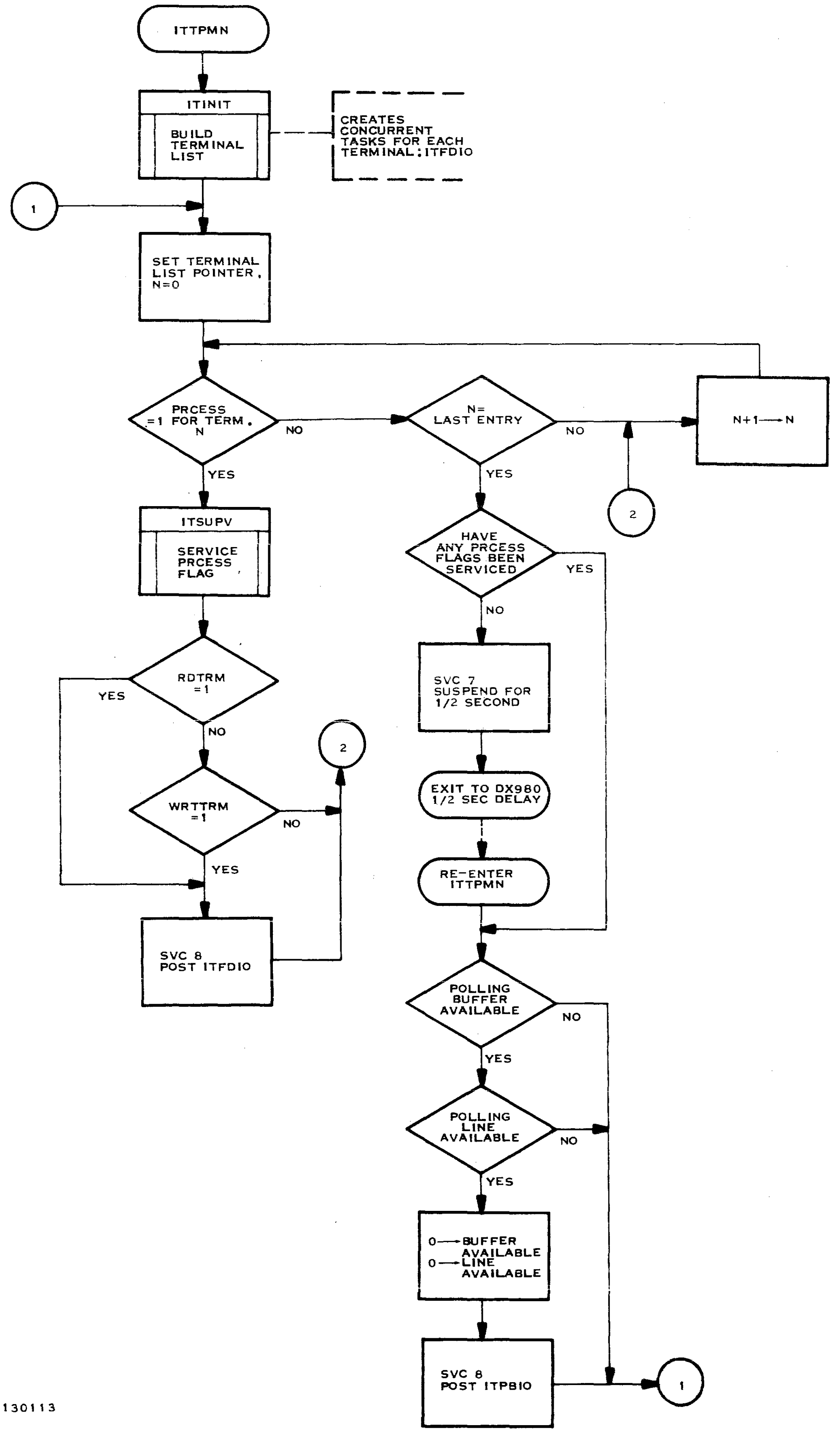
C.1.1 TERMINAL PROCESS MONITOR

The Terminal Process Monitor (ITTPMN) is the main task for ITS and coordinates the operation of the other tasks in the subsystem. Figure C-2 provides a conceptual flow chart for ITTPMN. The monitor calls two subroutines, scans the PRCESS flag for each of the terminals connected to it and activates the I/O subroutines.

C.1.1.1 ITINIT. When ITS is first activated, the monitor calls the ITINIT subroutine. This subroutine examines the Logical Device Tables (LDT) for the job, to determine what terminals are under control of ITS, and the DX980 Physical Device Table (PDT) for the assigned devices, to determine the characteristics of each terminal. (ITS uses a special SVC so that these functions can be performed while executing in the protected mode.) From this information the subroutine builds a terminal list in the ITS workspace area for each terminal assigned to ITS. The list contains the characteristics of the terminals. Table C-1 defines the information contained in a list entry for one of the terminals.

In addition to building a terminal list, ITINIT allocates one I/O buffer for each full duplex terminal and a specified number of I/O buffers to support polled terminals. Each full duplex I/O buffer is permanently assigned to the full duplex terminal. The polled terminal I/O buffers are dynamically assigned. ITINIT also allocates one Physical Record Block (PRB) and creates one I/O task for each I/O buffer. The I/O tasks for full duplex buffers use the reentrant procedure, ITFDIO. The I/O tasks for polled buffers use the reentrant procedure, ITPBIO. ITINIT sets the Read Terminal (RDTRM) flag for each of the terminals and starts them in their respective I/O task. When all I/O tasks are created, ITINIT returns control to ITTPMN. The I/O tasks are described later in this section.

C.1.1.2 TERMINAL LIST SCAN. When ITINIT returns to ITTPMN, the monitor starts scanning each entry in the Terminal List to determine if any PRCESS flags are set. The PRCESS flag indicates that the terminal operator has entered a complete record that is ready for processing by ITS. When ITTPMN encounters a terminal entry with a set PRCESS flag, it calls the Supervisor (ITSUPV) to service the request from the terminal. If ITTPMN scans the entire Terminal List without finding a set PRCESS flag, it suspends itself for one-half second. At the end of the delay period, ITTPMN reactivates and begins the table scan again. Each scan cycle after the delay period begins with the first entry in the list and continues sequentially through the list until it reaches the end or detects a PRCESS flag. When returning to the scan cycle after servicing a PRCESS flag, the scan starts at the terminal following the serviced terminal in the list. ITTPMN does not take the one-half second delay at the end of a cycle if it has serviced a terminal during that cycle.



(B)130113

Figure C-2. Terminal Process Monitor Conceptional Flow Chart

Table C-1. ITS Terminal List Entry

Word	Bit	Field Name	Definition
0	0	RDTRM	Read Terminal Flag: Setting this flag instructs the I/O task for the terminal to issue a Read Terminal I/O call.
	1	TYPTRM	Terminal Type: If this bit is a 0, the corresponding terminal is a full duplex terminal; if this bit is a 1, the terminal is a polled terminal.
	2	PRCESS	Process Flag: The terminal I/O task sets this flag to indicate to ITTPMN that data from the corresponding terminal is ready for processing.
	3	WRTTRM	Write Terminal Flag: Setting this flag instructs the I/O task for the terminal to issue a Write Terminal I/O call.
	4-7	BFNUM	Buffer Number: This number indicates which of the polling buffers has been dynamically assigned to this terminal. This field is not used for full duplex terminals.
	8-15	LUN	Logical Unit Number: Indicates which LUN that the terminal responds to.
1	0-15	TRMTSK	Terminal Task: Task identification of the last I/O task that serviced this terminal.. This field is static for a full duplex terminal and dynamic for a polled mode terminal.
2	0-15	UCBPTR	User Control Block Pointer: The location of the user control block for the terminal.
3	0-15	PRBPTR	Physical Record Block Pointer: The location of the PRB for the terminal. This field is static for a full duplex terminal and dynamic for a polled mode terminal.
4	0-15	TRBFLN	Terminal Buffer Length: The number of characters that can be put into the buffer for the terminal.

C.1.1.3 ITSUPV. ITTPMN calls the ITSUPV to service a terminal whose PRCESS flag is set. ITSUPV is a major part of ITS and is discussed in detail later in this section. During processing, ITSUPV resets the PRCESS flag so that it will not be recognized again during the next table scan. However, application programs called by ITSUPV may set the PRCESS flag again if they do not complete processing. If ITSUPV requires additional input from the terminal, it sets the RDTRM flag before returning to ITTPMN; if data output to the terminal is required, ITSUPV sets the WRTTRM flag before returning to ITTPMN.

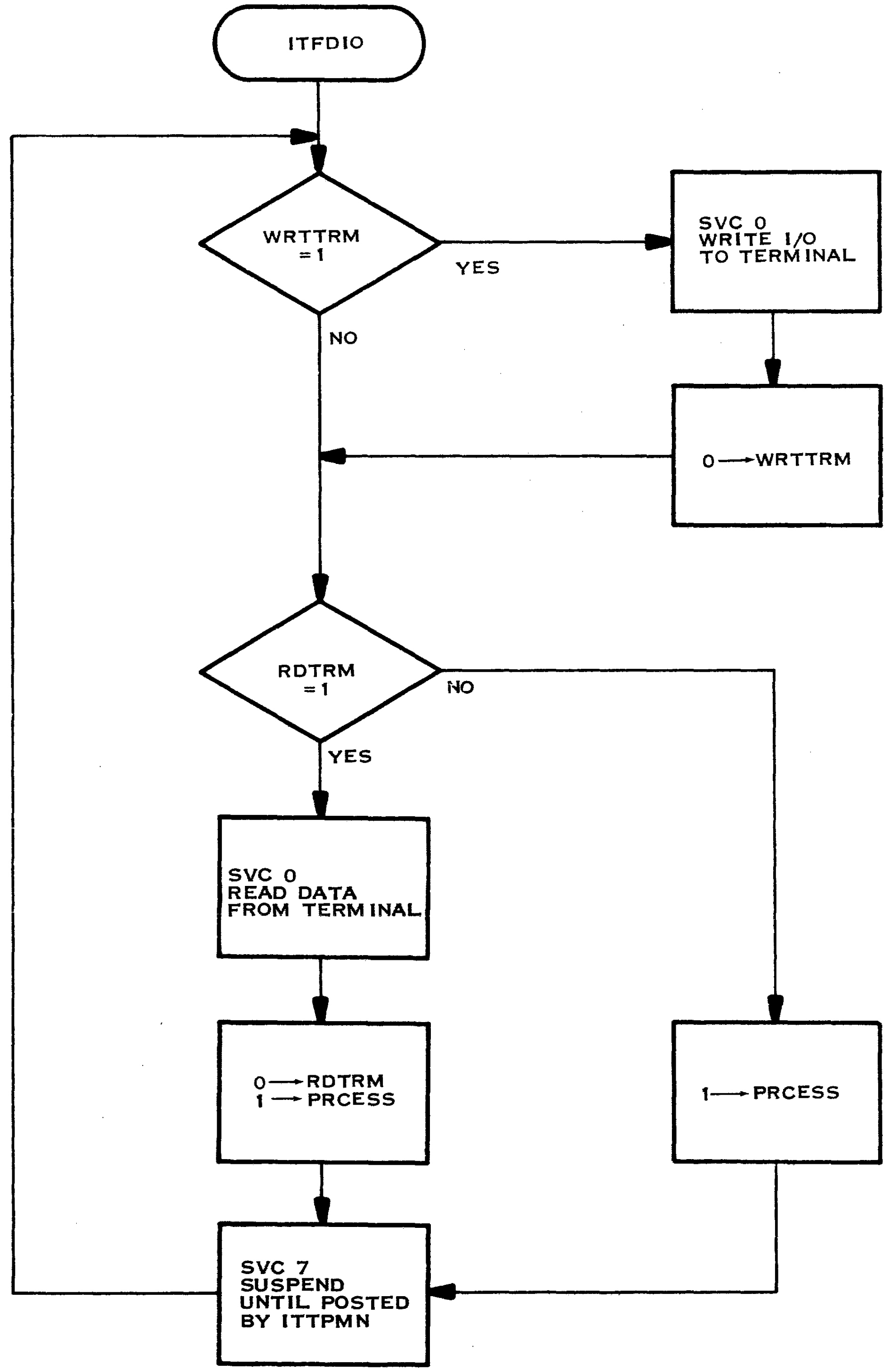
C.1.1.4 TERMINAL I/O. If either the RDTRM or the WRTTRM flag is set when ITSUPV returns to ITTPMN, the monitor issues a POST to activate an I/O task. If the terminal operates in full duplex mode, the monitor activates a task using the ITFDIO procedure; if the terminal is a polled terminal, the monitor activates a task using the ITPBIO procedure. Either of these tasks (initially created by ITINIT) transfers data to and from the terminal device by issuing I/O supervisor calls.

C.1.2 FULL DUPLEX I/O TASK (ITFDIO)

When building the Terminal List, ITINIT determines if a terminal is operating in full duplex mode. For full duplex terminals, ITINIT allocates memory to supply a terminal buffer (including one extra word for format control) and a Physical Record Block (PRB) for that terminal. It also creates a task with pointers to the PRB and the Terminal List as arguments and places the task identification in word 1 of the Terminal List entry for that terminal. The task, the terminal buffer and the PRB remain associated with the terminal. ITFDIO uses information in the terminal list to direct I/O operations between the ITS and the full duplex terminal. Figure C-3 illustrates the functions performed by the subroutine.

C.1.2.1 TASK INITIATION. Before ITINIT creates a terminal task for ITFDIO, it sets the RDTRM flag in the Terminal List entry for that terminal. Therefore, when ITFDIO begins execution, it issues an I/O SVC to read data from the terminal. ITINIT starts tasks for each terminal in the system in the same manner, so that when ITINIT returns control to ITTPMN, all terminal tasks are waiting for input from the corresponding terminals.

C.1.2.2 PROCESS REQUEST. When the operator enters a complete record of data at the terminal, the data terminal Device Service Routine (DSR) returns control to ITFDIO. ITFDIO then resets the RDTRM flag to indicate the completion of a read operation, and sets the PRCESS flag to indicate that the data is in the input buffer and requires attention from ITSUPV. ITFDIO then suspends processing to wait for either the RDTRM or the WRTTRM flag to set.


ITFDIO
WRTTRM = 1
YES
NO
SVC 0 WRITE I/O TO TERMINAL
0 → WRTTRM
RDTRM = 1
NO
YES
SVC 0 READ DATA FROM TERMINAL
0 → RDTRM
1 → PRCESS
1 → PRCESS
SVC 7 SUSPEND UNTIL POSTED BY ITTPMN

(A)130114

Figure C-3. ITFDIO Conceptual Flow Chart

C.1.2.3 RESPONSE TO TERMINAL. If ITSUPV, or a subordinate application subroutine, produces data in response to a terminal request, it also sets the WRTTRM flag after filling the terminal buffer with data to be written to the terminal. This flag causes ITTPMN to post the ITFDIO task for that terminal and instructs that task to issue a Write Terminal I/O SVC. If more data is required, RDTRM is set. If both WRTTRM and RDTRM are set, the I/O is performed in the order of write and then read. Neither ITSUPV or the subordinate applications issue I/O SVCs to communicate directly with the terminals.

C.1.2.4 DISPLAY SIZE. During construction of the Terminal List, ITINIT determines the display size of the terminal from the PDT. The display size, plus the format control word, determines the allocated buffer size for the terminal, and also specifies the length field in the PRB when the terminal is opened. Buffer lengths for the terminal devices are as follows:

- Teleprinter - 82 characters
- Teletype - 74 characters
- CRT - variable up to 1922 characters

If available memory cannot support several terminals with a 1922 character display, the PDT can be modified to reduce the display size. The display size for CRTs can be reduced to 80 characters so that the CRT responds as a teleprinter.

C.1.3 POLLED TERMINAL I/O TASK (ITPBIO)

The polled I/O routine (ITPBIO) is a reentrant subroutine that, unlike ITFDIO, does not require a task for each terminal. Instead, ITINIT creates a task for each polling buffer. The number of polling buffers can be significantly less than the number of terminals. Figure C-4 illustrates the functions performed by the subroutine.

C.1.3.1 TASK INITIATION. When ITINIT determines that there is one or more polled terminals assigned to ITS, memory is allocated for a specified number of polling buffers and PRBs. The number of polling buffers specified for the standard ITS is two. Although this number can be increased, two should be adequate for most applications. Unlike full duplex terminals, a task is not created for ITPBIO when a polled terminal is added to the terminal list. However, like full duplex terminals the RDTRM flag is set. After all terminals have been added to the terminal list, ITINIT allocates memory for a buffer and PRB, and creates a task for each polling buffer. The first time through, ITPBIO does not issue a terminal read since ITINIT does not specify which terminals should be polled. Thus control goes immediately to an SVC to suspend ITPBIO until posted by ITTPMN.

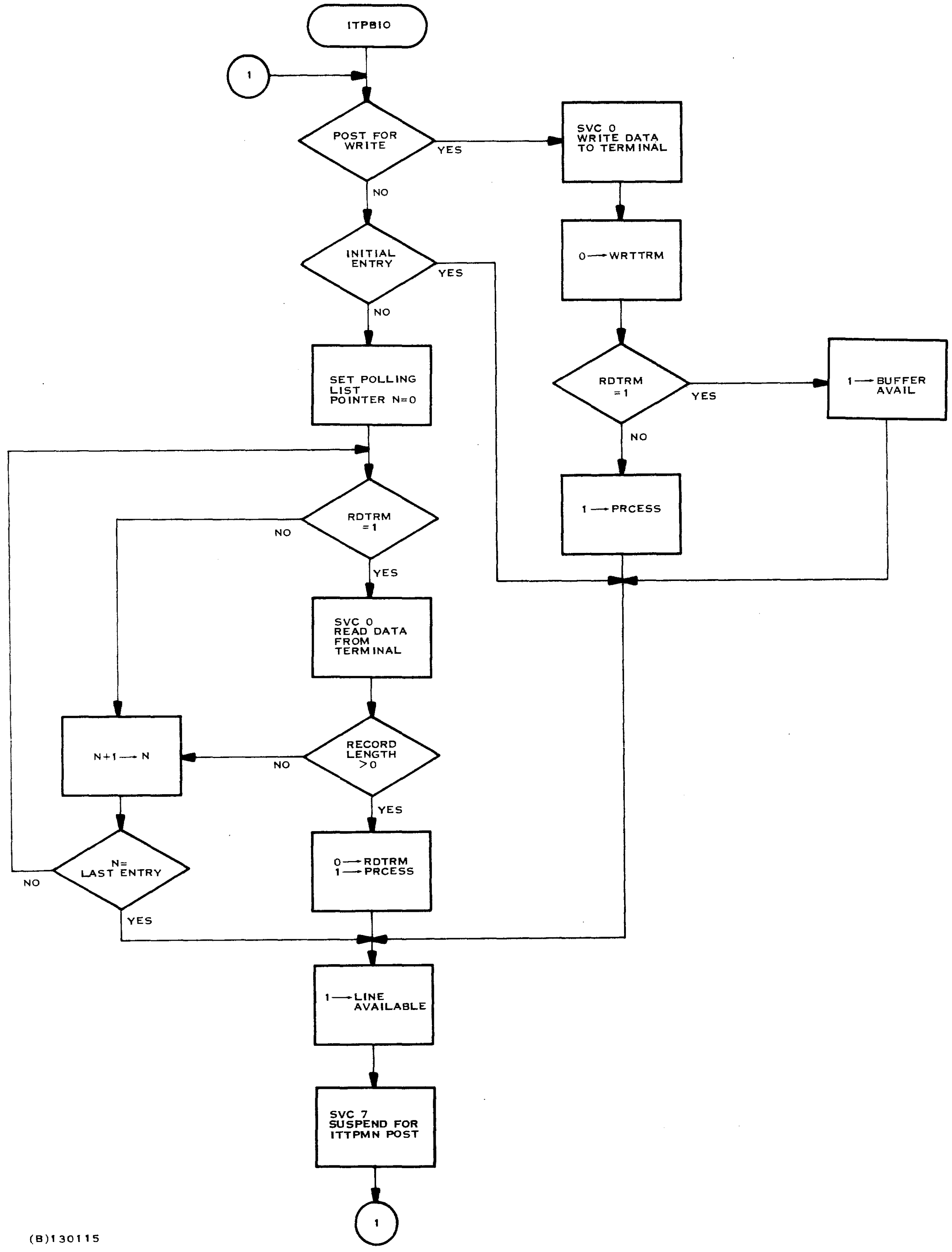


Figure C-4. ITPBIO Conceptual Flow Chart

C.1.3.2 PROCESS REQUEST. ITTPMN checks the status of the polling buffers and the polling lines (communication modules) if:

1) a terminal is ready for processing and ITTPMN calls ITSUPV, or
2) no terminal is ready for processing and ITTPMN suspends itself for one-half second.

If both a polling buffer and a line are free, ITTPMN issues a post for the appropriate ITPBIO together with a pointer to the line to be polled.

After receiving control, ITPBIO issues a read to each terminal on the polling line in succession. If a terminal user has pressed the transmit key prior to issuance of the read to his terminal, the screen data is transferred when the read is issued. If the transmit key has not been pressed, the terminal does not respond and the DSR returns control to ITPBIO with a record length of zero. This record length is treated as a negative response and a read is issued to the next terminal on the line. The first positive response terminates the poll and the PRCESS flag is set for the corresponding terminal. ITPBIO then issues a suspend SVC to wait for another post.

C.1.3.3 RESPONSE TO TERMINAL. The WRTTRM flag can be set by ITSUPV or a subordinate application subroutine. This causes a post to the appropriate ITPBIO task for data transfer. After the data is transferred, ITPBIO determines whether RDTRM is also set. If RDTRM is set, ITPBIO releases the buffer and issues a suspend SVC. If RDTRM is not set, ITPBIO sets PRCESS and does not release the buffer. This process allows a program to maintain a buffer throughout a series of data transfers to a terminal.

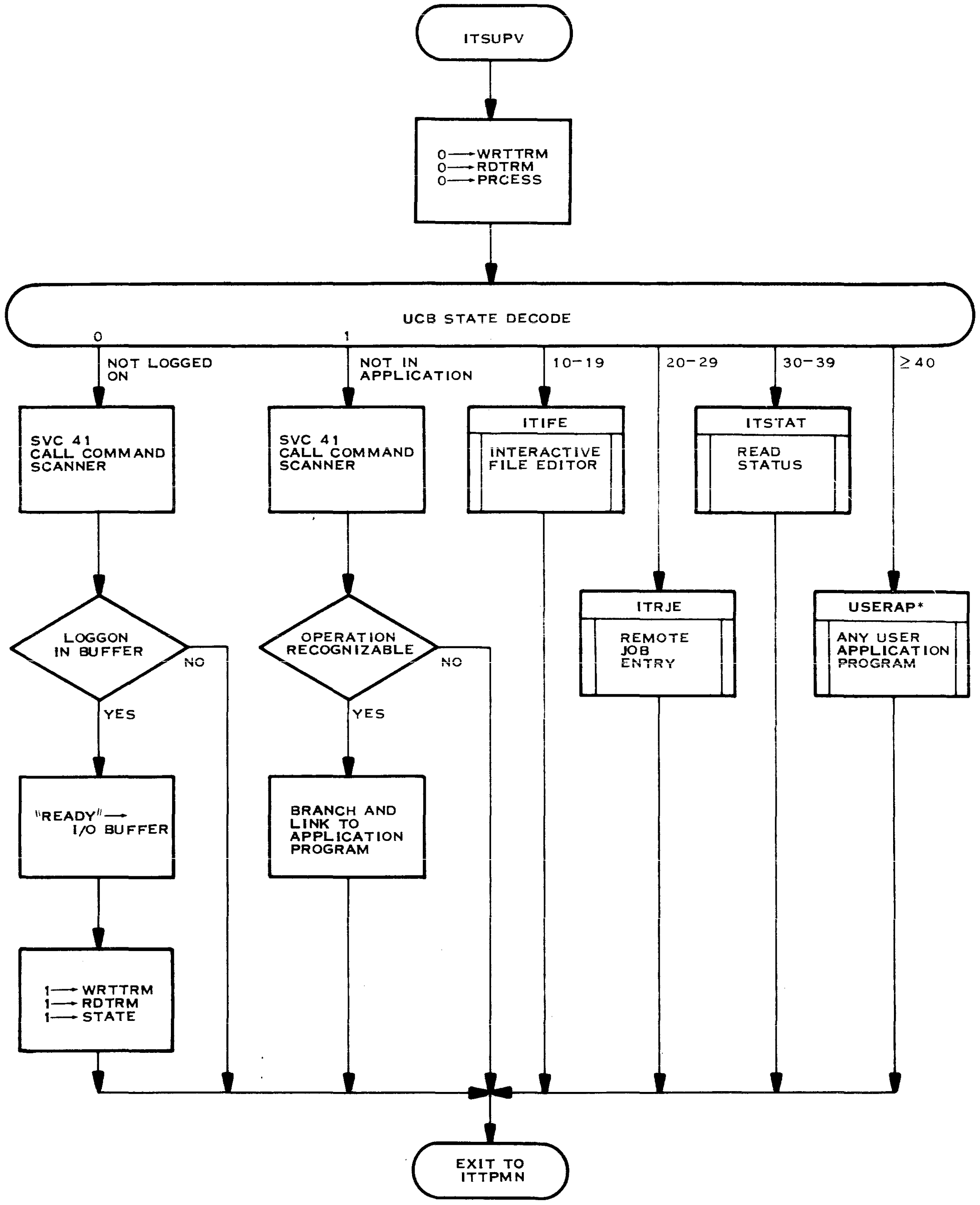
C.1.3.4 DISPLAY SIZE. When ITINIT determines that polling buffers are required, it also determines the largest display size for the terminals to be polled. It then allocates memory to each buffer that corresponds to the maximum size.

C.1.4 SUPERVISOR (ITSUPV)

The ITS Supervisor (ITSUPV) is a combination state and table driven controller that acts as an intermediary between ITTPMN and the application programs that run under ITS. When ITINIT initializes a terminal, it creates a table of parameters, called the User Control Block (UCB), that stays with the terminal as long as it is assigned to ITS. The information within the UCB varies with the user program that is currently using the terminal. Table C-2 defines the information fields within the UCB. When control transfers to ITSUPV to service a particular terminal, it clears the Terminal List flags for that terminal and examines the state-field of the UCB to determine what type of servicing is required. Figure C-5 illustrates the logical sequence of events within ITSUPV. Table C-3 defines the terminal states and the actions required of ITSUPV to service a terminal in that state.

Table C-2. ITS Terminal User Control Block (UCB)

Word	Field Name	Description
0-2	USERID	User Identification: This field contains the <userid> that is entered at the terminal during a Logon operation. Not validated by current ITSUPV. Reserved for future accounting software.
3-4	ACCTNO	Account Number: This field contains the <acctno> that is entered at the terminal during a Logon operation. Not validated by current ITSUPV. Reserved for future accounting software.
5	STATE	Terminal State: This field contains a number that indicates to ITSUPV how to respond to a service request from the terminal.
6	TRMPTR	Terminal Pointer: This field contains the memory address of a table of device characteristics for the terminal.
7	OPTR	Operation Pointer: This field contains space for the application program to load a pointer for its use.
8-9	TIMEON	Clock Time of Logon: This field is not used by the standard ITS.
10-11	DATEON	Calendar Date of Logon: This field is not used by the standard ITS.
12	TRMLNS	Terminal Lines: This field specifies the number of lines on the display unit.
13	LINLEN	Line Length: This field specifies the length of each display line.
14-49	-	These fields provide intermediate storage of parameters for the application program currently using the terminal.



(A)130116

* USER PROGRAM AND OP CODE MUST BE ADDED TO TABLE OF RECOGNIZABLE CODE FOR ITSUPV.

Figure C-5. ITSUPV Conceptual Flow Chart

Table C-3. User Control Block State Definitions

State Number	Terminal Conditions/Requirements
0	Inactive: User has not entered a Logon command to identify himself. ITSUPV checks only for a Logon command.
1	Ready: User has logged-on, but has not as yet requested any application program.
10-19	Edit: User is currently running the Interactive File Editor (ITIFE) application program. ITSUPV calls ITIFE to process request.
20-29	Remote Job Entry: User is currently running the Remote Job Entry (ITRJE) application program. ITSUPV calls ITRJE to process request.
30-39	Status: User is currently running the Status Display (ITSTAT) application program. ITSUPV calls ITSTAT to process request.
$\geqq 40$	User Program: User is currently running an application program not normally supplied with ITS. ITSUPV transfers control to the user program if the program has been entered in the table of recognizable code for ITSUPV.

C.1.4.1 STATE 0. If the UCB State field is zero, the user has not previously entered data from the terminal. Therefore, the first command from the terminal must be a Logon command to identify the user. In State 0 ITSUPV calls the command scanner to examine the input code. If the input code does not contain a Logon command, ITSUPV returns control to ITTPMN without further processing. If the code contains a Logon command, ITSUPV validates the syntax, transfers the pertinent data to the UCB and places a Ready indication in the I/O buffer. It then sets the WRTTRM flag to ensure that the Ready indication is sent to the terminal and sets the RDTRM flag to enable the user to respond by entering the next command. ITSUPV then places a value of 1 in the State field of the UCB before returning control to ITTPMN.

C.1.4.2 STATE 1. If the UCB State field is one, the user has logged-on, but has not as yet indicated an application program. ITSUPV then calls the command scanner to examine the input code (see Section V for a description of the command scanner). The input command must be one of the commands that ITSUPV can recognize. The standard ITS system provides a table of commands that allows ITSUPV to recognize the following input commands and their appropriate arguments:

EDIT

ENTER

LOGOFF

JOB

RUN

STATUS

DELETE

Any user programs that are added to ITS must also add at least one command to this list that will link ITSUPV to the user program. ITSUPV processes the Logoff command. Application programs process all other commands. If an application program processes the command, ITSUPV transfers control to that program with a Branch and Link (BRL) instruction. Pointers to the terminal list entry and to the command scanner data arrays accompany the command to the application program. The application program, therefore, has access to all control blocks for the terminal, plus the command and arguments that were entered at the terminal.

C.1.4.3 OTHER STATES. If the UCB State field is greater than one, ITSUPV transfers control directly to the application program. Terminals in this state have previously used an application program. That program set the state field to a value that returns control to the program for further input, or that logs-off the user from that program. ITSUPV does not process commands in these higher states. Therefore, the application program must decode any commands from the terminal.

C.1.5 APPLICATION PROGRAMS

Application programs that run under ITS are closed subroutines. The arguments that ITSUPV passes to the program are pointers to the appropriate terminal list entry and the command scanner arrays. Figure C-6 shows the relationship between the control blocks and structures.

C.1.5.1 CONTROL TRANSITIONS. When control is passed to an application program, the RDTRM, WRTTRM, and PRCESS flags are all set to zero and the buffer pointer in the PRB points to the first data word in the terminal buffer. If this is the initial entry into the subroutine, STATE is set to 1 and the key and packed arrays in the command scanner structure contain the decoded command line. Before the application program returns control to ITSUPV, the program sets RDTRM, WRTTRM, PRCESS, STATE, and the format control word according to the requirements of the application.

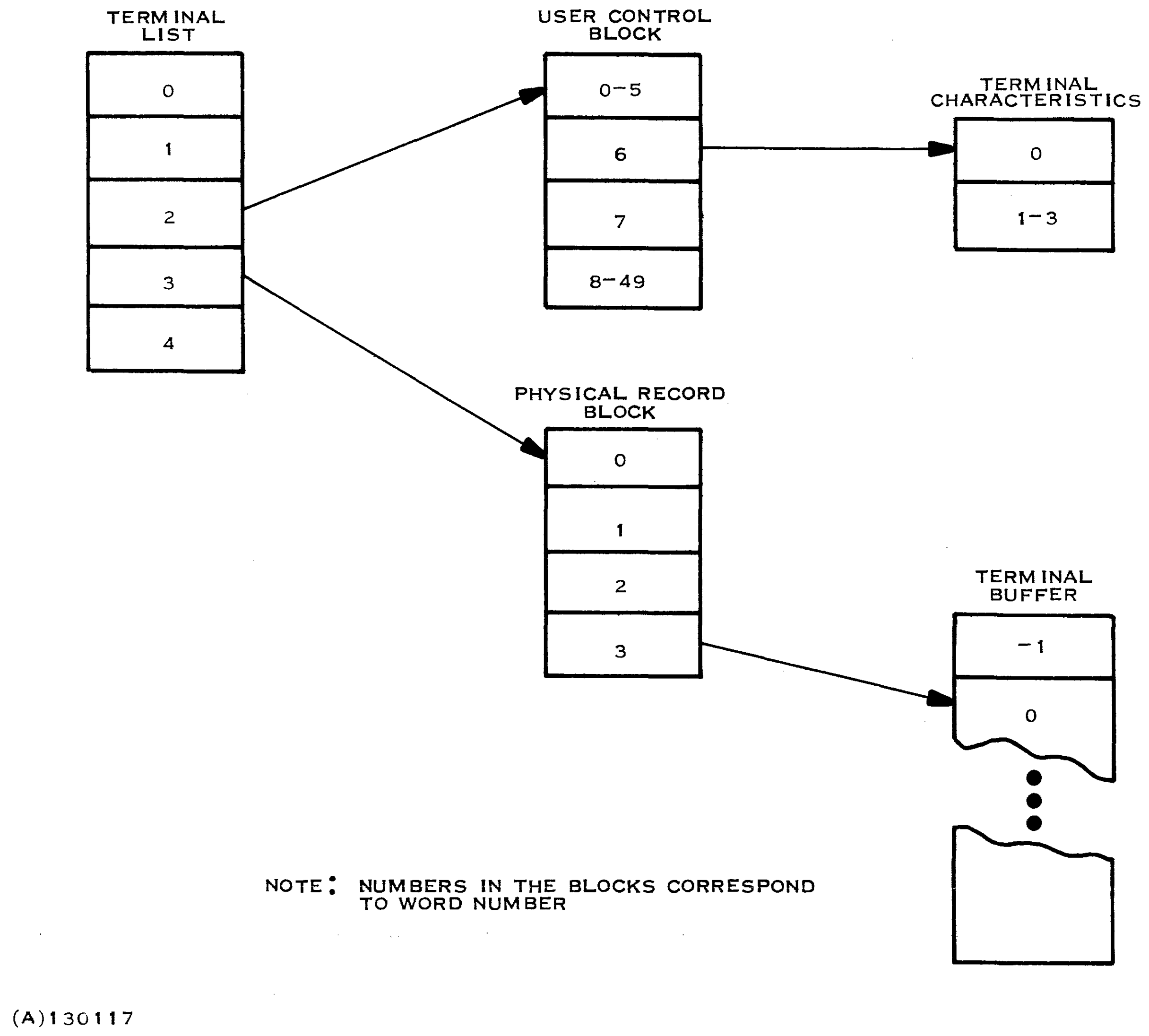


Figure C-6. ITS Control Blocks

C.1.5.2 SAMPLE APPLICATION TRANSFER. The following example from the file editor illustrates the strategy for performing a function under ITS. The terminal is in the Ready state, and the user wants to edit a file by changing the string ABC to XYZ everywhere that it appears. The file contains 50 records. The following sequence performs that operation:

1. User keys EDIT FILE=(1,USER01,MYFILE) and presses RETURN from a teleprinter; ITFDIO then sets the PRCESS flag.

2. ITTPMN detects that processing was requested and calls ITSUPV. ITSUPV calls the command scanner, determines that the command is to be processed by the file editor (ITIFE) and makes the call.

3. ITIFE calls a subroutine to assign and open the file, and read the first file record into the terminal buffer. ITIFE then sets the format control word to $000E_{16}$, RDTRM to 1, WRTTRM to 1, and STATE to 11, and returns control to ITSUPV. ITSUPV returns to ITTPMN to activate the I/O task for the user's terminal.

4. ITFDIO writes the terminal buffer and issues a read to accept more input.

5. Terminal user enters RA 1, 50 /ABC/ /XYZ/ to direct the file editor to replace all strings ABC with the string XYZ in the next 50 records of the file.

6. ITTPMN detects that processing was requested and calls ITSUPV. ITSUPV determines that STATE is between 10 and 19 and calls ITIFE. ITIFE calls the command scanner, determines that the command is a string operation, and calls a subroutine that processes all string commands. The string processor reads 25 records, changing ABC to XYZ wherever found. After processing the 25 records, (the subroutine interrupts itself rather than being arbitrarily cutoff) the string processor changes STATE to 14, leaves RDTRM and WRTTRM reset, sets the PRCESS flag and returns control through ITIFE and ITSUPV to ITTPMN. ITTPMN does not post the I/O routine because neither RDTRM or WRTTRM are set. However, it does continue the terminal scan at the next entry.

7. In time the terminal scan progresses to the same terminal. ITTPMN detects that processing was requested and passes control through to the string processor to process the last 25 records. After processing is complete, the string processor puts the last record processed in the terminal buffer, sets RDTRM and WRTTRM to 1, changes STATE back to 11 and returns. ITIFE sets the format control word to $000E_{16}$ and returns through ITSUPV to ITTPMN to activate the appropriate I/O task. Control then returns to the user. The user can key in either more edit commands for further file processing, a Stop command to get back to the ready state, or a Logoff command to exit the system.

C.2 DESIGN PRINCIPLES

Before adding a new application program to ITS, the user must understand the design features in the subsystem that provide the highest overall efficiency for the subsystem. Observance of these principles when implementing a new program will enhance its performance within the system.

C.2.1 DUAL MODE OPERATION

ITS can run in either the protected or the privileged mode. The first DXOLE control card (see Section VIII) controls the mode of operation. If the card specifies SUBSYSTEM, the program links for privileged operation; if the card specifies NORMAL, the program links for protected operation. Privileged mode is slightly faster than protected mode since it performs less error checking for privileged SVCs. In addition, a privileged mode program can be smaller since it can access the runtime package that is linked into the memory resident portion of the operating system.

Protected mode simplifies debugging a new ITS application program, and guarantees that an error in the new program will not destroy the operating system. When preparing the new program, link it in the protected mode. Then check the program by running a single terminal with the PROT parameter specified in the Execute command (EXEC) to the JCL translatore. When the application is completely checked in this manner, it can be linked with the rest of ITS with SUBSYSTEM specified on the DXOLE control card.

C.2.2 APPLICATIONS OVERLAY

To attain maximum memory efficiency, ITS application programs can be overlayed. The majority of ITS is coded as reusable subroutines. This convention allows the overlays to be the simple, preplanned overlays supported by DXOLE.

C.2.3 PSEUDO TIME SLICING

To avoid monopolizing the system for a single terminal application, all application programs must return control to ITTPMN periodically. To maintain an average access time of three seconds for any terminal and for a total of up to 30 terminals, the average processing time for any one terminal request must be limited to 100 milliseconds. Many terminal requests require less than the 100 millisecond average; long functions may use up to a maximum of 200 milliseconds. The actual interruption of the program, however, is left to the program itself. That is, either the program completes its process within the time limit, or the program interrupts its processing at a logical breakpoint to return to ITTPMN within the time limit. If the program interrupts itself, it returns to ITTPMN with the PRCESS flag set, and with both RDTRM and WRTTRM flags clear. Thus, no I/O operations are initiated and ITTPMN returns to the program during the next Terminal List scan. This process of allowing the program to limit its time instead of being truncated after an arbitrary period is called pseudo time slicing.

Two functions in the standard ITS package, the RJE processor and the Display Status processor, do not follow the pseudo time slicing guidelines. These processors both contain supervisor calls that take longer than 200 milliseconds to perform (Start Job and Stat SVCs, respectively). Control does not return to the processor until the SVC is complete. For each of these processors a small reentrant module appears in the root segment of the ITS overlay. After the Job, Run and Stat commands are syntactically validated, the processor creates a task that points to the reentrant module. The reentrant module issues the SVC. The processor then resets PRCESS, RDTRM and WRTTRM flags and returns normally to ITTPMN. When the SVC returns, the reentrant module sets the PRCESS flag and issues a Delete Task SVC. During the next Terminal List scan cycle, ITTPMN detects the PRCESS flag and reactivates the processor.

C.3 MODIFYING ITS

Calls from ITS to applications programs are driven by a set of tables in the module, ITSTBL. To add an application, the ITSTBL source module must be modified, assembled, and linked with ITS and with any new applications programs. Refer to figure C-7 for a listing of the components in the ITSTBL module. Two sets of tables must be modified to add an application: The State/Call Translation Tables and the Application Names/Initial Entry Tables.

C.3.1 STATE/CALL TRANSLATION TABLES

The State/Call Translation Table, STATET, breaks all of the possible states into State Intervals. Once the interval is determined by ITSUPV, it is used as an index for an indirect branch through a table of application entry addresses (SCALLT). Note that on the standard table states 0 through 1 map into a call to ITCOM (ITS command decode), states 2 through 19 map to ITIFE (although only states 10 through 19 are used), states 20 through 29 map to ITRJE, and states 30 through 39 map to ITSTAT. To add an application that uses states 40 through 49, a 'DATA 49' statement must be inserted after the 'DATA 39' statement in STATET, and a 'DATA application entry point' must be inserted after the 'DATA ITSTAT' statement in SCALLT. This will cause ITSUPV to call the given application for states 40 through 49. Note that a single application may have more than one state interval and entry address if desired.

C.3.2 APPLICATION NAMES/INITIAL ENTRY TABLES

This set of tables map application names to initial application entry points. The first table, RESLAB, is a list of eight character application keywords. The second table, ICALLT, is a list of application entry points that correspond to the application names. New application names must be inserted after the DATA 'JOB', 'DELETE' statement in RESLAB, and new initial entry points must be inserted after the last 'DATA ITIFE' statement in ICALLT. The order of the standard eight commands may not be changed.

```
        IDT   ITSTBL
**************************************************************************
*   ABSTRACT   * THIS TABLE CONTAINS THE USER VARIABLE PARAMETERS
*                FOR THE INTERACTIVE TERMINAL SYSTEM, AND DEFINES
*                THE APPLICATION NAMES AND ENTRY ADDRESSES. THIS
*                MODULE MUST BE MODIFIED IN ORDER TO ADD ANY
*                ADDITIONAL APPLICATIONS ROUTINES.
*
*   ROUTINES
*   CALLED     * LOGON,LOGOFF,ITRJE,ITSTAT,ITIFE,ITCOM
*
*



**************************************************************************
       HED  ITS COMMAND/STATE TABLF
*************************************************************
*      NOTE :                                               *
*         THIS MODULE MUST BE INCLUDED IN THE ROOT SEGMENT  *
*************************************************************
       DEF  INFO,RESLAB,KEYA,PAKSTR,ICALL,SCALL,S3ATET,POLTIM
       REF  LOGON,LOGOFF,ITRJE,ITSTAT,ITIFE,BADST,ITCOM
A      EQU  0
E      EQU  1
X      EQU  2
M      EQU  3
S      EQU  4
L      EQU  5
B      EQU  6
P      EQU  7
BR     EQU  1
BX     EQU  3
       PEJ
*
*      APPLICATION NAMES (RESLAB)
*
RESLAB EQU  $
       DATA 'LOGON   ','LOGOFF  '   ** POSITION SENSITIVE **
       DATA 'RUN     ','STATUS  '   ** POSITION SENSITIVE **
       DATA 'EDIT    ','ENTER   '   ** POSITION SENSITIVE **
       DATA 'JOB     ','DELETE  '   ** POSITION SENSITIVE **
*                        INSERT NEW APPLICATIONS ABOVE
RESD   EQU  $-RESLAB     THIS LABEL MUST FOLLOW LAST
NRES   EQU  RESD/4          APPLICATION NAME
*
*      INITIAL ENTRY ADDRESS TABLE
*
ICALLT EQU  $
       DATA LOGON        0
       DATA LOGOFF       1
       DATA ITRJE        2
       DATA ITSTAT       3
       DATA ITIFE        4
       DATA ITIFE        5
       DATA ITRJE        6
       DATA ITIFE        7
       PEJ
*
```

Figure C-7. ITSTBL Listing (Sheet 1 of 2)

```
*       STATE/CALL TRANSLATION TABLES
*
STATET  EQU  $
        DATA 1              STATES 00-01
        DATA 19                    02-19
        DATA 29                    20-29
        DATA 39                    30-39
*                           INSERT NEW STATES HERE
        DATA -1                    ALL OTHERS
*
*
*
SCALLT  DATA ITCOM          00-02 COMMAND DECODE
        DATA ITIFE          10-19 IFE
        DATA ITRJE          20-29 RJE
        DATA ITSTAT         30-39 STATUS
*                           INSERT NEW CALLS HERE
        DATA BADST          ALL UNDEFINED STATES
        PEJ
*
*       ROUTINE TO MAKE INITIAL CALL BY APPL. NAME
*
ICALL   EQU  $
        RMO  B,A            SAVE BASE
        RMO  M,B            GET SECOND ARGUMENT
        LDX  *2,BR          *
        RMO  A,B            RESTORE BASE
        LDA  ICALLT,X       GET CALL ADDRESS
        RMO  A,P            CALL
*
*       ROUTINE TO MAKE SUBSEQUENT CALLS ACCORDING TO STATE
*
SCALL   EQU  $
        RMO  B,A            SAVE BASE
        RMO  M,B            GET SECOND ARG.
        LDX  *2,BR          *
        RMO  A,B            RESTORE BASE
        LDA  SCALLT,X       GET CALL ADDRESS
        RMO  A,P            CALL
        PEJ
*
*       CRSCAN 'CONTROL' ARRAY
*
INFO    EQU  $
        DATA NPAK           NUMBER OF CHARS. IN 'PAKSTR'
        DATA NKEY           NUMBER OF WORDS IN 'KEYA'
        DATA 0,0            CRSCAN WORKSPACE
        DATA NRES           NUMBER OF LABELS IN 'RESLAB'
        DATA 0              STARTING SCAN POSITION
NPAK    EQU  80             NO OF CHAR. IN PAKSTR
NKEY    EQU  50             NO OF WORDS IN KEY ARRAY
KEYA    BSS  NKEY
PAKSTR  BSS  NPAK/2
        DEF  TRMS,BUFS,LINS
MTRMS   EQU  32             MAX. NO. OF TERMINALS
MLINES  EQU  0              MAX. NO. OF POLLING LINES
MBUFS   EQU  0              MAX. NO. OF POLLING BUFFERS
TRMS    BSS  6*MTRMS        TERMINAL LIST
BUFS    BSS  4*MBUFS        BUFFER LIST
LINS    BSS  36*MLINES      LINE LIST
POLTIM  DATA 500            POLLING INTERVAL (IN MILLISECONDS)
        DEF  UCBSIZ
UCBSIZ  DATA 50             USER CONTROL BLOCK SIZE
        END
```

Figure C-7. ITSTBL Listing (Sheet 2 of 2)

C.3.3 CALLING CONVENTIONS

Calls from ITSUPV follow the standard DX980 calling conventions and are as follows:

- Initial Entry (via Applications Names/Initial Entry tables)

```
@LDM=ARGLST
@BRL APPLIC
     .
     .
     .
ARGLST DATA 2
       DATA TRMLST          pointer to TRMLST entry
       DATA  keyno          pointer to Key Number index
```

- Subsequent Entry (via State/Call Translation Tables)

```
@LDM=ARGLST
@BRL APPLIC
     .
     .
ARGLST DATA 2
       DATA TRMLST
       DATA STATI           pointer to state interval
```

C.3.4 CRSCAN ARGUMENTS

The calling argument, key array, and packed string for the command decode done in states 0 and 1 are in ITSTBL.

C.3.5 TRMS

All TRMLST entries are built in ITSTBL in the buffer 'TRMS'. The size of this area is controlled by the label, MTRMS.

C.3.6 POLLING PARAMETERS

Several parameters may be adjusted to suit the requirements of the user. These parameters are:

- POLTIM - Time interval for polling
- MLINS - Maximum number of polling lines
- MBUFS - Maximum number of polling buffers

C.3.7 USER CONTROL BLOCK SIZE

The user may alter the size of the USER CONTROL BLOCKS by adjusting the value of the statement:

```
UCBSIZ DATA 50
```

Note that the minimum UCB size for IFE is 50 words.

APPENDIX D
ADDING NON-STANDARD DEVICES TO DX980

APPENDIX D

ADDING NON-STANDARD DEVICES TO DX980

D.1 GENERAL

By using DX980 utilities, user supplied routines, and information about the I/O package, up to four non-standard devices may be added to a DX980 configuration. However, non-standard discs cannot be added. To support a new device, the user must design, code and install a device service routine in the system. In addition, he must implement a utility to build and modify I/O associated tables. This appendix provides the necessary information and procedure to accomplish these tasks.

D.2 WRITING A DEVICE SERVICE ROUTINE

Writing a new Device Service Routine (DSR) requires a knowledge of the DX980 input/output structure, coding requirements for a DSR, plus an understanding of the device characteristics to be allowed for in the routine. The following paragraphs provide that background in addition to sample DSR's to be used as a guide.

D.2.1 INPUT/OUTPUT STRUCTURE

A program initiates Input/Output operations by using an I/O supervisor call (SVC) and passing the required operation via a Physical Record Block (PRB). The SVC is actually an illegal instruction that generates an internal interrupt. The internal interrupt decoder passes control to the SVC Processor. After determining that an I/O SVC has been made, the SVC Processor gives control to the I/O Manager. Figure D-1 illustrates the relationship of the I/O Manager within the I/O system.

The I/O Manager performs the required housekeeping of the I/O associated tables, controls the available devices, sets up and controls DSR entry and exit, and performs all other common I/O SVC functions. The following are some of the pre - DSR device independent functions performed by the I/O Manager:

- Control and housekeeping of Open/Close calls
- Device assignment checks
- LUN Open checks
- Control of the share/exclusive capabilities
- Data buffer boundary checks
- Linking PRB's and Logical Device Tables (LDT's) to the Physical Device Tables (PDT's)

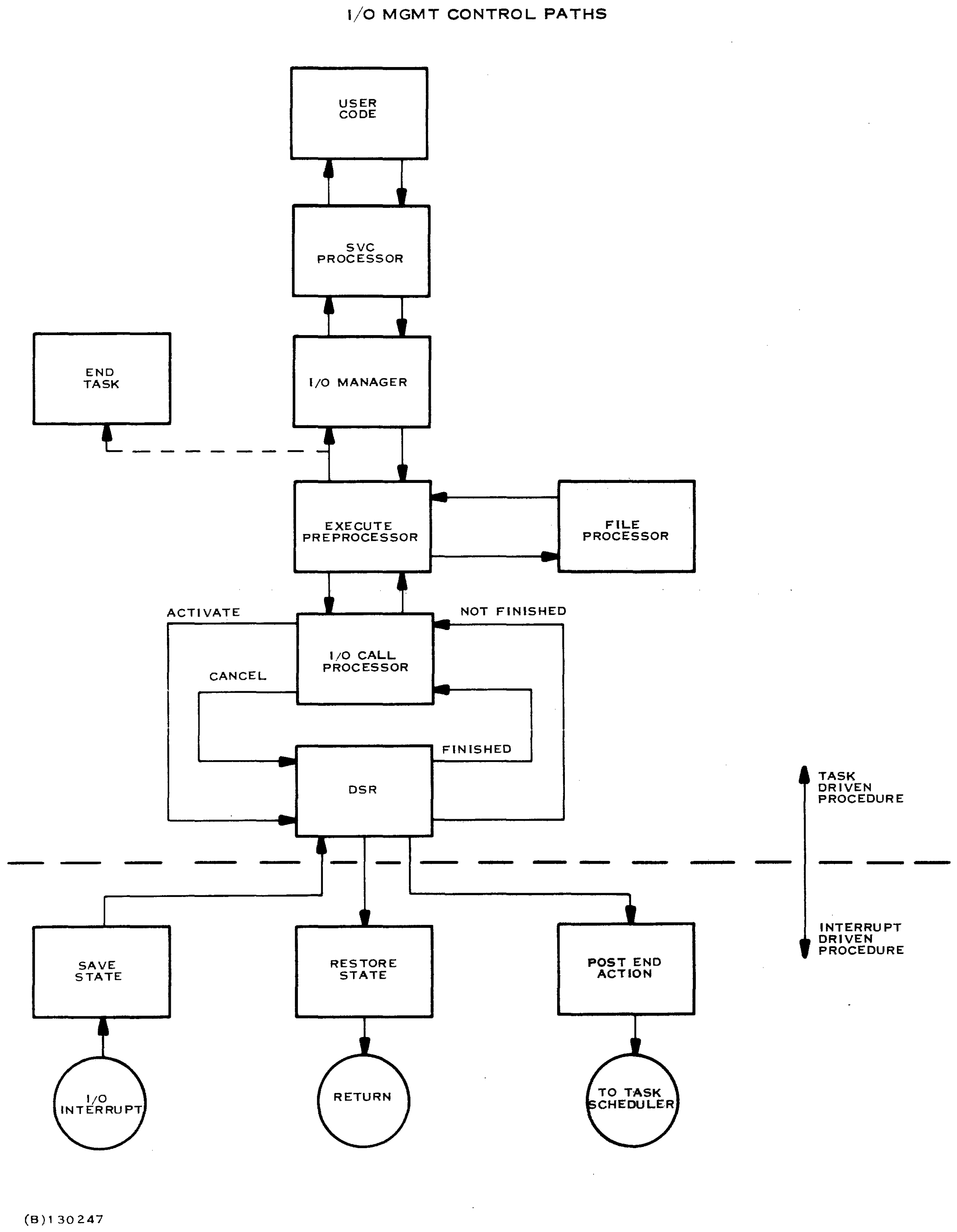


Figure D-1. General I/O Flow

- Queuing operations when a device is busy
- Monitoring system operations for initiate I/O SVC's
- Calling the correct DSR

D.2.2 DEVICE SERVICE ROUTINES (DSR's)

DSR's provide the actual interface with the hardware. They check for illegal operations and for special device dependent and unique conditions not monitored by the I/O Manager. The DSR's have three standard entries: an initial entry, a reset or cancel entry and an interrupt entry.

D.2.2.1 INITIAL AND CANCEL ENTRIES. For initial entry, the I/O Manager issues an SVC to begin an I/O operation. The I/O manager also controls the cancel or reset entry by issuing an SVC to the DSR when the system is trying to terminate an I/O operation in progress. The SVC is as follows:

```
CALL SVC (DSR #, PDT @, TYPE)
```

In this form, the term PDT @ represents a pointer to the PDT containing the I/O information. The term TYPE indicates that the call is either an initial or a cancel entry to the DSR.

Since the I/O Manager calls DSR's with an SVC, the DSR can have only one entry point. For this reason and also to perform some common logic routines, the DSR executes a system routine (ISDSRI) immediately when entered. This call must appear in all DSR's and has the following format:

```
        REF ISDSRI
START   EQU $
        RMO L, A    1st word of code
        @BRL ISDSRI
        DATA interrupt entry pointer
        DATA cancel entry pointer
          .
          .
          .
        (logic for initial entry)
          .
          .
          .
```

The initial entry checks operation validity, starts the I/O operation and, if possible, completes the operation. The reset path terminates the I/O operation by reseting the interrupt logic and, trying to halt the I/O device.

D.2.2.2 INTERRUPT ENTRIES. I/O interrupts can occur during or at the end of an I/O operation from either an I/O Bus or a Direct Memory Access Channel (DMAC) device. I/O bus interrupts can occur following each character transfer while DMAC devices usually interrupt when the operation is complete. The operations following an interrupt resemble the response to a supervisor call. However, the interrupt decoder coordinates the activity instead of the SVC Processor. The interrupt decoder transfers control to the DSR through the interrupt entry. The interrupt entry either completes the operation or prepares for further interrupts.

D.2.2.3 REGISTER INITIALIZATION. Regardless of the type of entry, when the DSR is entered from the entrance utility, the following registers are initialized:

- Register E = 0
- Register S = entry address of the utilities
- Register M = pointer to the PRB
- Register B = pointer to the PDT

In addition, if the entry was an initial entry, registers A and X contain the operation code.

D.2.2.4 EXITS. For any type of entry the DSR may take one of six possible exits (see the DSR EXIT utility). These may be classified into three groups. The NORM exit is the normal exit from the DSR when no abnormal conditions have been detected and the data transfer is not yet complete. The EOR exit indicates an End-of-Record exit when the requested data transfer is complete. Other types of exits are used to indicate that abnormal conditions were encountered.

D.2.2.5 I/O ERRORS. Errors that occur during an I/O operation are described in Section III of this manual. The I/O Error numbers listed in Section III are in the range of 201 - 209. However, the I/O Manager adds 200 to the error number returned by the DSR. Therefore, the DSR need only provide an error number from 1 - 9 to the I/O Manager. An additional error number, 10, is reserved for use by the I/O Manager. When an error occurs that requires return to system control, the DSR performs one of the standard exit utilities.

D.2.2.6 SYSTEM ERRORS. Most I/O errors may require abortion of the user job requesting the I/O. Some types of I/O errors, however, are so severe that they endanger the integrity of the system. For these errors the DSR should halt the system so that the cause of the problem can be determined before the clues are destroyed. An available DSR routine, SCRASH, provides a standard method of bringing the system to a halt gracefully. This routine causes the CPU to IDLE. Information from the routine that called SCRASH may then be displayed. A sample calling sequence is shown below:

```
REF SCRASH
  .
  .
  .
@LDM =ERINFO   Load M Register with Address of Error Information
@BRL SCRASH    Call SCRASH
  .
  .
```

(Continued on next text page)

ERINFO DATA 1, BADERR	Number of Arguments, Address of Arguments
BADERR DATA ERCODE	Code that may be displayed in the A Register at IDLE.

If arguments other than an error code would provide helpful information, the number of arguments may be increased. At IDLE the M Register provides a pointer to this information. See the DX980 System Operation Guide for the standard set of SCRASH codes.

D.2.2.7 INFORMATION RETURNED BY THE DSR. The DSR only sets the Operation Ignored bit in the system set flags (PRB word 0, bit 3) when an operation is not implemented. The I/O manager sets all other system set flags. The DSR may pass device dependent information to the caller through the non-dedicated bits of the PRB. When passing information that requires several words, the DSR must use a buffer that is pointed to by the third word of the PRB. The operation code for this type of call is either 0, 1, 2, 3, 19, or 20. The DSR does not need to set or reset any indicators for the I/O Manager. The type of exit taken indicates a course of action for the I/O Manager.

D.2.3 I/O UTILITY ROUTINES

The I/O utility routines perform common DSR functions with minimal coding in the DSR itself. By using the utilities, the DSR need never know the memory locations of any PDT, PRB or LDT; furthermore it does not need to monitor character input and output buffer indexes. The DSRs use the exit utilities to ensure proper handling of interrupts and other types of exits. Failure to use the optional utilities creates increased DSR size, development time, and maintenance requirements. The I/O Utility Routines perform services frequently required by Device Service Routines (DSRs). The functions available are:

PRB/PDT/LDT Bit Manipulation

PRB/PDT/LDT Word Transfer (via register)

PRB/PDT/LDT Conditional Skip on Bit

READ/WRITE on I/O Bus (via register)

PUT/GET Character from Packed Buffer (via register)

LINKAGE to DSR Exit Routine

The routines generate no task or system errors. All I/O interrupts must be masked and the B Register must point to the PDT when using the routines.

D.2.3.1 UTILITY ROUTINE INTERFACE. The utility routines are serially re-usable. The DSR need not know the location of the utility routines in memory. When the DSR is entered by the I/O call processor or Interrupt Decoder, the S-register contains the I/O utility entry point. The routines are entered by execution of an REX S, P instruction. To enhance DSR readability, the REX S, P instruction is defined as a new instruction, IOCOM, using the OPD assembler directive:

```
     IOCOM    OPD     C7C7,5
```

A two word calling sequence is required to use a utility function:

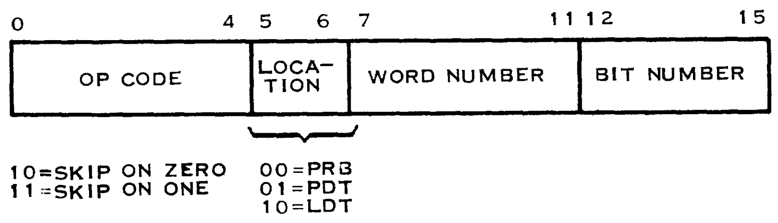
```
     IOCOM

     OPERATION   OPR1,OPR2,....,OPR(N)
```

The I/O Utility ENTRY/EXIT Routine transfers control to the correct opcode processor. This processor decodes the operand(s) and performs the function. On return to the DSR, execution resumes at the instruction following the operation designator (unless a skip was executed. For skip instructions execution is resumed two instructions after the operation designator). No active registers used by the DSR are changed by the common utility routines unless called for in the operation; however, the status register compare indicators are volatile.

All general-purpose DSR's use the utility functions as much as possible, sometimes a special-purpose DSR cannot afford the added overhead of using the Common Utility Functions. To allow for in-line coding where necessary, the DSR is always entered with the PDT location in the B-register and the PRB location in the M-register. Additionally, the E-register is always zeroed when a DSR is entered.

D.2.3.2 SET/CLEAR PRB, PDT, OR LDT BIT. This routine allows the DSR to manipulate bits within the PRB, PDT or LDT. The machine instruction appears in the following format:



The assembler directives that define the instruction are as follows:

```
    BIT      FRM     5,2,5,4
    SET      EQU     1
    CLR      EQU     0
    PRB      EQU     0
    PDT      EQU     1
    LDT      EQU     2
```

Therefore, the general form of the instruction becomes:

BIT $\begin{Bmatrix}\text{SET}\\ \text{CLR}\end{Bmatrix}$, $\begin{Bmatrix}\text{PRB}\\ \text{LDT}\\ \text{PDT}\end{Bmatrix}$, <word number>, <bit number>

For example, the expression:

```
IOCOM
BIT SET, PRB, 1, 5
```

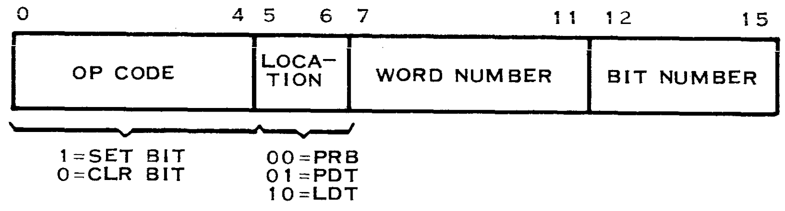
changes bit 5 of PRB Word 1 to a value of 1. Also, the expression:

```
IOCOM
BIT CLR, LDT, 0, 15
```

changes bit 15 of LDT Word 0 to a value of 0.

The utility's execution time is between 26.00 to 29.50 microseconds.

D.2.3.3 CONDITIONAL INSTRUCTION SKIP FROM PRB, PDT OR LDT BIT. This routine allows the DSR to inspect a specific bit of either the PRB, the PDT or the LDT and either skip or not skip depending upon the state of that bit. The machine instruction word appears as follows:



The assembler directives that define the instruction are as follows:

```
BIT     FRM    5,2,5,4
SKIP0   EQU    2
SKIP1   EQU    3
PRB     EQU    0
PDT     EQU    1
LDT     EQU    2
```

Therefore, the general form of the instruction is:

BIT $\begin{Bmatrix}\text{SKIP0}\\ \text{SKIP1}\end{Bmatrix}$, $\begin{Bmatrix}\text{PRB}\\ \text{PDT}\\ \text{LDT}\end{Bmatrix}$, <word number>, <bit number>

For example, the expression:

```
IOCOM
BIT SKIP0, PRB, 3, 15
```

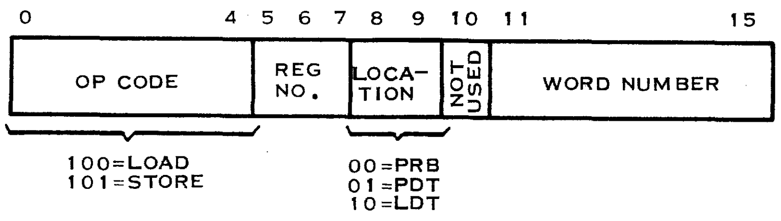
indicates that if bit 15 of PRB word 3 is a zero, then the next instruction should be skipped. Similarly, the expression:

```
IOCOM
BIT SKIP1, LDT, 0, 0
```

indicates that if bit 0 of LDT word 0 is a 1, then the next instruction should be skipped.

The execution time for this utility is between 27.50 to 29.75 microseconds.

D.2.3.4 LOAD/STORE PRB, PDT OR LDT WORD TO/FROM REGISTER. The routine allows the PRB to transfer a word between a designated register and a specified word in either the PRB, PDT or LDT. The machine instruction appears in the following format:



The assembler directives that define the instruction are as follows:

```
REG    FRM  5,3,2,6
LOAD   EQU  4
STORE  EQU  5
PRB    EQU  0
PDT    EQU  1
LDT    EQU  2
```

The general form of the instruction is:

$$\text{REG} \begin{Bmatrix} \text{LOAD} \\ \text{STORE} \end{Bmatrix}, \text{<register number>}, \begin{Bmatrix} \text{PRB} \\ \text{PDT} \\ \text{LDT} \end{Bmatrix}, \text{<word number>}$$

For example, the expression:

```
IOCOM
REG   STORE, A, PDT, 6
```

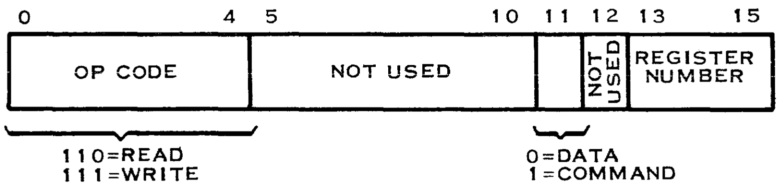
transfers the contents of the A Register to word 6 of the PDT. Similarly, the expression:

```
IOCOM
REG   LOAD, M, LDT, 0
```

transfers the contents of word 0 of the LDT to the M Register.

The execution time for this utility is 20.25 to 23.25 microseconds.

D.2.3.5 READ/WRITE I/O BUS TO/FROM REGISTER. This routine allows the DSR to transfer the contents of a register to the I/O Bus, or to fill a particular register from the I/O Bus. The machine instruction appears in the following format:



The assembler directives that define the instruction are as follows:

```
IOBUS   FRM    5,7,4
READ    EQU    6
WRITE   EQU    7
DATA    EQU    0
CMMD    EQU    1
```

The general form of the instruction is:

$$\text{IOBUS} \begin{Bmatrix} \text{READ} \\ \text{WRITE} \end{Bmatrix}, \begin{Bmatrix} \text{DATA} \\ \text{CMMD} \end{Bmatrix}, \text{<register number>}$$

For example, the expression:

```
IOCOM
IOBUS   READ, DATA, X
```

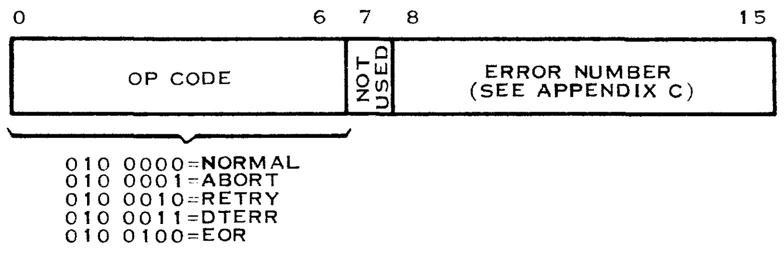
transfers data from the I/O Bus to the X Register. Similarly, the expression:

```
IOCOM
IOBUS   WRITE, CMMD, A
```

transfers a command from the A Register to the I/O Bus.

The execution time for this utility is 29.00 to 40.50 microseconds.

D.2.3.6 EXIT DSR. This routine links the DSR to the proper routine to handle the conditions that exist when the DSR is completed. The machine instruction for this routine appears in the following format:



The assembler directives that define the instruction are as follows:

```
EXIT    FRM   7, 9
NORM    EQU   20₁₆
ABORT   EQU   21₁₆
RETRY   EQU   22₁₆
DTERR   EQU   23₁₆
EOR     EQU   24₁₆
```

The general form of the instruction is

EXIT {type} , {error number}   Error number is ignored for NORM and EOR exits.

For example, the expression:

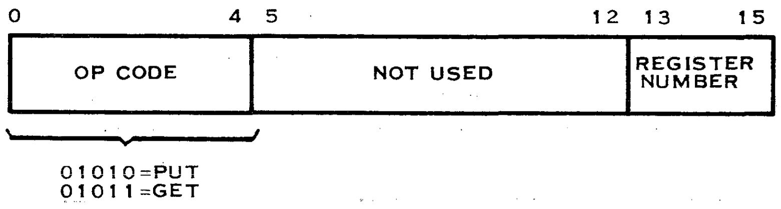
```
IOCOM
EXIT NORM, 0
```

indicates a normal exit from the DSR. Also, the expression:

```
IOCOM
EXIT ABORT, 9
```

indicates that the operation was aborted due to an attempt to execute an illegal I/O operation to the device (error code 9).

D.2.3.7 PUT/GET. This routine performs character transfer between a packed buffer and a specified register. The PUT routine transfers a character from the right half of the register to the packed buffer and increments the PRB character count. The routine may also set bit 0 of the register to indicate that storing that character filled the buffer space. If the buffer space is already full, the routine sets bit 1 of the register to indicate that the character was not stored. The GET routine transfers a character from the buffer to the right half of the specified register and increments the output count contained in the first word of the temporary storage area of the PDT. The routine may also set bit 0 of the register to indicate that fetching the current character emptied the buffer. If the buffer is already empty, the routine sets bit 1 to indicate that no character is available. The machine instruction for this routine appears in the following format:



The assembler directives that define the instruction are as follows:

```
CHAR   FRM   5,11
PUT    EQU   A16
GET    EQU   B16
```

The general form of the instruction is:

$$\text{CHAR} \begin{Bmatrix} \text{PUT} \\ \text{GET} \end{Bmatrix}, \text{<register number>}$$

For example, the expression:

```
IOCOM
CHAR  PUT,M
```

transfers a character from the M Register to a packed buffer. Similarly, the expression:

```
IOCOM
CHAR  GET,E
```

transfers a character from a packed buffer to the E Register.

The execution time for the put and get utilities between 58.25 to 67.00 microseconds and 49.50 to 55.50 microseconds, respectively.

D.2.4 PHYSICAL DEVICE TABLES

A Physical Device Table (PDT) contains parameters, such as device addresses and special attributes, that are necessary for control and performance of an I/O operation. Figure D-2 illustrates the format for a standard PDT and Table D-1 defines each of the fields. Device Service Routines use the PDT's for temporary data storage between interrupts. Each device within a system has a PDT containing information exclusive to that device. However, one common DSR may be used to coordinate I/O operations for a group of identical devices.

WORD		
0	NEXT PDT ADDRESS	
1	FLAGS	
2	INTERNAL DEVICE NUMBER	COMMIT PRIORITY
3	ASSIGN COUNT	OPEN COUNT
4	LOCKING LDT POINTER	
5	ERROR CODE	SVC INDEX
6	DEVICE ATTRIBUTES	
7	DEVICE	
8	NAME	
9		
10	POINTER TO EXTENDED PDT OR EXTERNAL REGISTER	
11	I/O LOAD FACTOR	TIMEOUT
12	DSR INTERRUPT ENTRY	
13	PRB ADDRESS	
14	LDT ADDRESS	
15	I/O DONE EVENT LINK	
16	DATA BUFFER ADDRESS	
17	UNSOLICITED INTERRUPT PROCESSING	
18	OUTPUT CHARACTER COUNT	
19 ↓ n	DSR TEMPORARY STORAGE AREA (VARIES FOR EACH DEVICE)	

(A)130248

Figure D-2. Physical Device Table General Structure

Table D-1. Standard PDT Field Definitions

Word	Bits	Definition
0	0-15	PDT chain word, pointer to the next PDT. Zero in this word indicates that this is the last PDT in the system.
1	0	Device Busy Flag - initially zero and set by I/O manager.
	1-2	RESERVED - initially zero.
	3	Exclusive access/shared access flag set by assignment. Initially zero (shared access) by Job Manager.
	4	Exclusive access/shared access flag set at OPEN time by I/O Manager. Initially zero (shared access).
	5	Locked/not Locked flag. Initially zero (not locked) and set by I/O Manager.
	6	ONLINE/OFFLINE status set by online processor.
	7	Not sharable/sharable attribute set at system generation.
	8	Extended PDT indicator . When set, word 10 contains pointer to an Extended PDT for a central controller.
	9	RESERVED - initially zero.
1	10-11	System/user mode indicator - used only for system console data terminal by I/O manager and DSR to determine whether device is in a system or user mode. Initially zero.
	12-15	DSR used flags; I/O Manager resets to zero at every initial DSR entry.
2	0-7	Internal device identification number - Each PDT has a unique number. Identification numbers 0-20, inclusive, are reserved and should not be used. All others are assigned according to table 2-2.
	8-15	Commit priority - initially zero.
3	0-7	Assign count, initially zero.
	8-15	Open count, initially zero
4	0-15	Locking LDT pointer, initially zero.
5	0-7	This byte indicates the operation status upon completion; initially zero, it is controlled by the I/O manager and DSR exit utilities.

Table D-1. Standard PDT Field Definitions (Continued)

Word	Bits	Definition
5 (con't)	8-15	SVC index - Determines which DSR is to be called. The value is determined by the memory image phase (MIP) containing the DSR, as follows:
		MIP / SVC Index
		181 / 22
		182 / 23
		183 / 24
		184 / 25
6		Device attributes. Returned in the PRB when the device is opened.
	0	System console only
	1	Dummy device only
	2	Rewindable
	3	Device can be forward spaced
	4	Device can be back spaced
	5	Printing Device
	6	ASR 733 cassette
	7	Data Terminal or CRT
	8	Disc
	9	Input
	10	Output
	11	USASCII Device
	12	Binary Device
	13	Polled CRT
	14-15	00 - Non-disc Device 01 - Linked Sequential Disc File 10 - Relative Record Disc File 11 - Indexed Disc File
7-9	0-15	Device Name - This may be any valid USASCII 6 character string which has not previously been used as a device name.

Table D-1. Standard PDT Field Definitions (Continued)

Word	Bits	Definition
10	0-15	If the Extended PDT flag is set, then this word contains the address of the Extended PDT. If this is a data bus device and the I/O utilities are to be used, this word contains two possible device addresses. Refer to I/O Utilities description in this section. Otherwise, this word is available for DSR use.
11	0-7	I/O load factor.
	8-15	Device Timeout - can be selected by loading value less than 255 into the timeout count byte of the PDT. Counting begins when an I/O Call is initially processed. If a device is timed out before the data transfer is complete, a correctable I/O error occurs. Each count corresponds to one second to allow a maximum timeout of 4 minutes and 14 seconds. Some devices will require resetting the last record. Keyboard type devices do not normally have a timeout. The timeout in the PDT's should be set to FF_{16} to specify no timeout.
12	0-15	DSR interrupt entry - Initially points to a set of code that performs a required task when an I/O operation is not currently being done to this device. Normally the only logic required clears the interrupt and returns control to the system. This word is controlled by the entry and exit utilities. This code may reside in the Extended PDT or temporary storage area. This entry normally handles unexpected hardware interrupts.
13	0-15	PRB Pointer for the PRB currently being processed by a DSR. Initially zero.
14	0-15	Logical Device Table (LDT) pointer for the LDT currently being processed by the DSR. Initially zero.
15	0-15	I/O done event link controlled by the exit utilities and the system task scheduler. Initially zero.
16	0-15	If the operation currently being processed by a DSR is a Read Binary, Read USASCII, Write USASCII, Write Binary, Write Direct, or Read Direct, then this word contains the data buffer address associated with the operation. This word is initially zero.

Table D-1. Standard PDT Field Definitions (Continued)

Word	Bits	Definition
17		Same as word 12.
18	0-15	Used by the utilities to maintain an output character count. Should be set to zero by the DSR at the start of character retrievals from the data buffer. Otherwise, this word is available for DSR use.
19	0-15	This area contains any temporary data or device dependent information required by the DSR.

Controllers that handle more than one identical device (moving head disc controller or magnetic tape controller for example) are not completely separate physical devices. Therefore, each multiple unit controller has an Extended PDT in addition to the regular PDT's for the separate devices connected to the controller. Figure D-3 illustrates the format for an extended PDT. Table D-2 defines each of the fields. The Extended PDT helps the I/O Manager determine the busy status of the controller and its individual devices, and also indicates which of several PDT's associated with the bus address applies to a generated interrupt. When an I/O operation is initiated, the I/O Manager sets the PDT pointer into Word 1 of the Extended PDT. The PDT address table contains a pointer to the Extended PDT instead of the PDT itself. When an interrupt occurs from one of the controller's devices, the Interrupt Decoders check the PDT Address Table to determine the address of the Extended PDT. The decoders then use data in the temporary storage area of the Extended PDT to determine which device PDT is associated with the interrupting device.

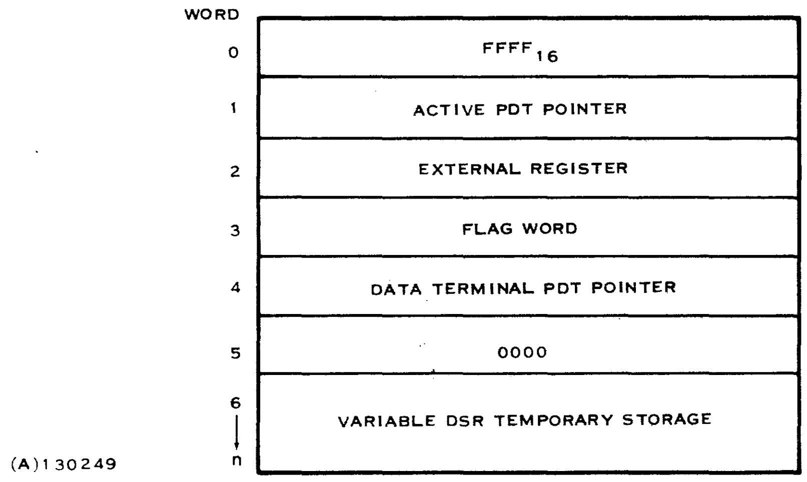


Figure D-3. Extended PDT General Structure

Table D-2. Extended PDT Field Definitions

Word	Definition
0	Must be $FFFF_{16}$
1	Active PDT Pointer - controlled by I/O Manager if the Extended PDT bit is set in the device PDT (Word 1, Bit 8).
2	External Register - used for command or data register by I/O bus device. Available for DSR use if device is on the DMAC.
3	Bit 0 - set to indicate controller busy.
	Bit 1 - set to indicate a Silent 700 Series Data Terminal Extended PDT. Word should be initially set to zero.
4	Pointer to Data Terminal PDT if Word 3, Bit 1 is set. Otherwise may be used as temporary storage.
5	Mode control for 733 ASR Cassette if Word 3, Bit 1 is set. Otherwise may be used as temporary storage. Initially zero.

Bit 8 of Word 1 in the device PDT indicates to the I/O Manager that an Extended PDT exists for a central controller. If that bit is set, the I/O Manager retrieves the Extended PDT Pointer from Word 10 of the device PDT to access the Extended PDT. It then examines Bit 0 of Word 3 of the Extended PDT to determine if the controller is busy.

D.2.5 LOGICAL DEVICE TABLE

The Logical Device Table (LDT) equates logical device numbers to physical devices or files. Figure D-4 illustrates the table format. Table D-3 provides a detailed breakdown of the fields. Job Management creates and deletes LDT's. When an I/O call is made, the I/O Call Processor searches the linked LDT's for a LUN definition. If it finds the LUN, it prepares

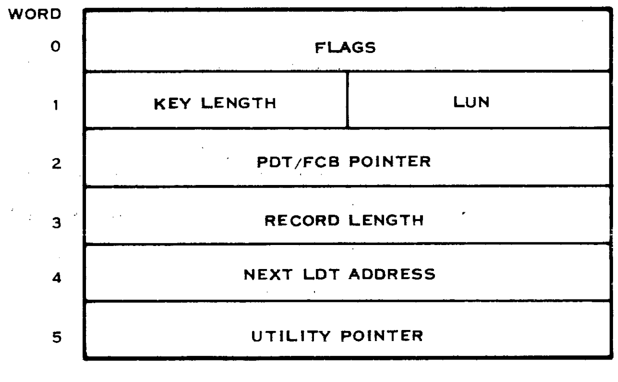


Figure D-4. DX980 Logical Device Table Format

Table D-3. LDT Field Definitions

Word	Bit	Definition
0	0	Device open
	1	File
	2	Blocked
	3	Reserved
	4	Return on I/O Error
	5	Busy
	6	Pass LDT
	7	Delete LDT
	8	Return on Retry
	9	Password - Owner
	10	Password Protected
	11	No Password Protection
	12	Temporary
	13	User/Creator Access
	14-15	Not Used
1	0-7	Key length - for files only
	8-15	LUN
2	0-15	Pointer to PDT for devices. Pointer to File Control Block for files.
3	0-15	Record length
4	0-15	Pointer to next LDT - zero if last LDT; Anchor is in Job Control Block.
5		Pointer to Record Control Block for files.

for either a DSR or a File Management entry, and relinguishes control to the correct DSR or File Management Routine. If it does not find a definition for the LUN, an error condition exists and the I/O call cannot be executed. Open and close calls change the LDT Flags and Pointers as necessary.

D.2.6 DEVICE TIMING FACTORS

Many I/O Bus interrupts are time-critical, whereas DMAC interrupts generally are not. Two factors affect I/O Bus interrupt performance:

1. The worst-case time to service the high-priority device,
2. The percentage of CPU time required to service interrupts.

D.2.5.1 WORST-CASE I/O BUS INTERRUPT LATENCY TIME. The worst-case latency time to service high-priority I/O Bus device is the sum of the following:

	Factor	Maximum Time Required
1.	The longest code sequence when interrupts can be masked. This occurs immediately after a previous interrupt has been decoded and includes set-up for DSR entry, DSR execution, DSR exit and return to the correct machine state.	400 microseconds
2.	Interrupt execution time that is a hardware function and does not add to the latency time.	N/A
3.	Retaining the current state of the machine and identifying the interrupt, plus setting-up for DSR entry. These factors are controlled by the I/O Bus Interrupt Decoder.	40 microseconds
4.	DSR execution to the internal point of I/O is dependent upon the DSR being used.	60 microseconds
	Total worst case latency time to honor a high priority I/O Bus interrupt	=505 microseconds

Table D-4 lists the character interval times of some standard I/O Bus peripheral devices.

D.2.6.2 INTERRUPT HANDLING CAPACITY. The worst-case percentage of CPU time required by a I/O Bus device is calculated by:

$$\frac{\text{WORST-CASE INTERRUPT SERVICE TIME}}{\text{TIME BETWEEN INTERRUPTS}} \text{ X } 100$$

The worst-case percentage of CPU time required by a DMAC device during the actual transfer is:

WORDS PER SECOND X 0.000075

Table D-4. I/O Data Rates and CPU Loading

Device	Character Time	Maximum System Load (Worst Case)
9600 Baud Modem	1040 μS.	29.0 %
1200 Baud Modem	8333 μS.	3.5 %
300 Baud Modem	33, 333 μS.	.91%
110 Baud Modem	100, 000 μS.	.3 %
300 CPM Card Reader	1760 μS.	17. %
Moving Head Disc	(DMAC)	7.25%
Model 979 Tape Transport	(DMAC)	1.25%
DS330 Disc System	(DMAC)	36. %
High Speed Paper Tape Reader	3333 μS.	9.1 %
High Speed Paper Tape Punch	13, 333 μS.	2.3 %
2310 Line Printer	(DMAC)	.02%

The worst-case I/O Bus interrupt service time is the sum of the worst cases of:

1. Interrupt identification and DSR entry time (45 microseconds)
2. DSR execution time (350 microseconds)
3. DSR exit until interrupts are re-enabled (40 μS. microseconds)

The total worst case time is 435 microseconds. The system can handle any combination of devices concurrently until the available CPU time is exceeded. The best performance occurs if the higher transfer rate devices are assigned to the high priorities.

D.2.6.3 SYSTEM OVERLOAD PREVENTION. The maximum system load percentage for each device is represented in the individual Physical Device Tables. The I/O Manager routines monitor the system I/O load at all times. If an I/O call is made that causes the load to exceed 90%, the task is queued until more CPU time is available. Higher-speed DSR's are designed with system load in mind and frequently show a significant improvement over the worst-case percentages.

D.2.7 CONFIGURATION LIMITATIONS

The Model 980 Computer has four catagories of interrupts: Internal, Priority Option, Direct Memory Access Channel (DMAC), and I/O Bus. Internal interrupts are not directly related to I/O transfers and are handled as a separate function. The I/O interrupt decoders assign the I/O Bus interrupts a higher priority than DMAC interrupts (this is different than the hardware priority assignment). Within the I/O Bus and DMAC interrupt levels, there is also a definite priority structure. I/O Bus interrupt handling has the greatest impact on overall system performance.

Versions of the I/O Bus interrupt decoder support the following I/O configurations:

I/O Configuration	Maximum I/O Ports With (Without) Internal Expansion
Internal Only	13 (4)
1 Expander	22 (13)
2 Expanders	31 (22)
3 Expanders	40 (31)
4 Expanders	49 (40)

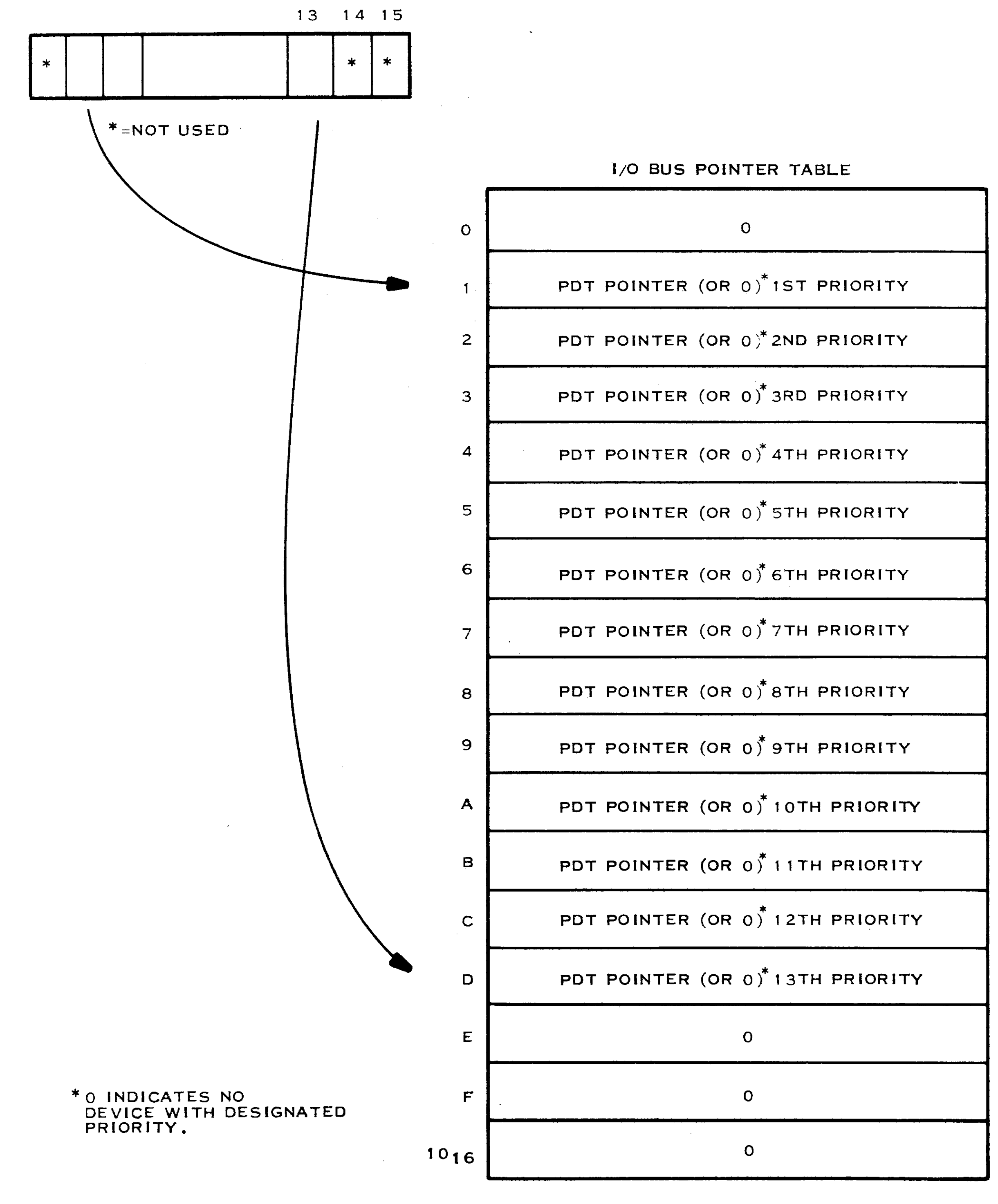
Figure D-5 illustrates priority the structure for internal expansion only. Figure D-6 illustrates the maximum I/O Bus configuration. The PDT pointer tables list the location of the PDT or Extended PDT associated with that device. If less than the maximum configuration is used, expanders should be added in the numerical order indicated. Internal expansion may or may not be included. If an I/O expander is not included, its internal port may be used for an I/O device. Figure D-7 shows the DMAC expander table.

The following equation determines the maximum number of ports per hardware configuration:

(# of expanders x 10) + (13 - # of expanders)

D.2.8 DX980 CONVENTIONS

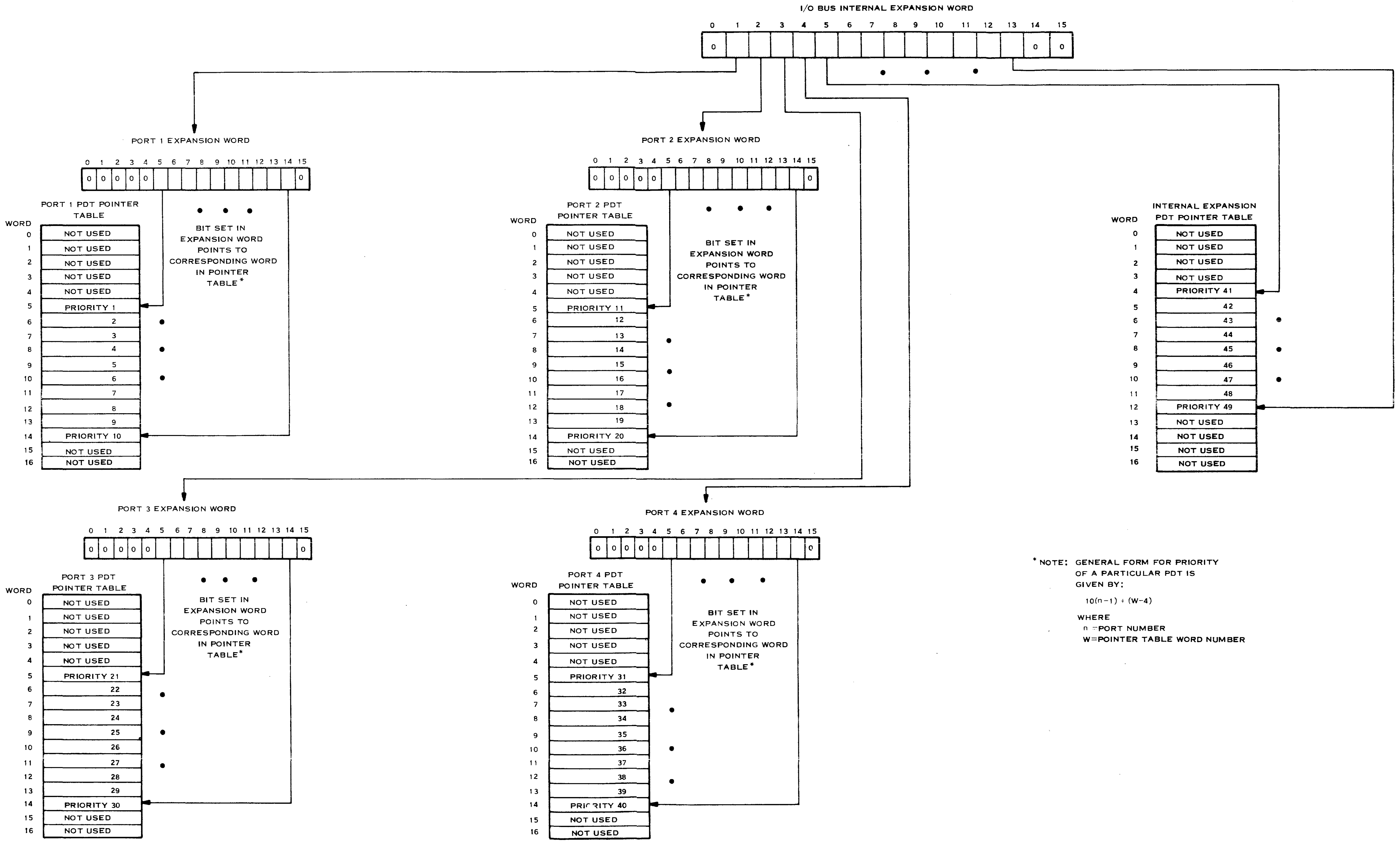
Some general rules apply to all sequential I/O devices as well as sequential disc files. Devices capable of both reading and writing cannot arbitrarily switch between Read/Write modes. The restrictions vary from device to device. Do not close and re-open a device to circumvent these restrictions; the results are unpredictable and different for each device. All DSR's should ignore I/O opcode 17 and 18 to allow for expansion with new functions. Opcodes above 20 are generally in error, although some are legal for random access to disc files and other device-dependent functions. For opcodes 0-3 and 19-20, the I/O Call Processor checks for allowable data buffers. Use these opcodes for data transfer functions. Use opcodes above 29 for special functions that do not directly involve a data transfer. The basic functions, Open and Close are processed by all standard Device Service Routines. All sequential writing DSR's provide a command to write an end-of-file. For most devices this is a /* record. The end-of-file command should always be used in lieu of writing a /* for that purpose.



13 14 15
*
*
*
*=NOT USED
I/O BUS POINTER TABLE
0
0
1
PDT POINTER (OR 0)* 1ST PRIORITY
2
PDT POINTER (OR 0)* 2ND PRIORITY
3
PDT POINTER (OR 0)* 3RD PRIORITY
4
PDT POINTER (OR 0)* 4TH PRIORITY
5
PDT POINTER (OR 0)* 5TH PRIORITY
6
PDT POINTER (OR 0)* 6TH PRIORITY
7
PDT POINTER (OR 0)* 7TH PRIORITY
8
PDT POINTER (OR 0)* 8TH PRIORITY
9
PDT POINTER (OR 0)* 9TH PRIORITY
A
PDT POINTER (OR 0)* 10TH PRIORITY
B
PDT POINTER (OR 0)* 11TH PRIORITY
C
PDT POINTER (OR 0)* 12TH PRIORITY
D
PDT POINTER (OR 0)* 13TH PRIORITY
E
0
F
0
10₁₆
0
*0 INDICATES NO DEVICE WITH DESIGNATED PRIORITY.

(A)130251

Figure D-5. PDT Branch Table with Internal Expansion Only



*NOTE: GENERAL FORM FOR PRIORITY OF A PARTICULAR PDT IS GIVEN BY:

10(n−1) + (W−4)

WHERE
n =PORT NUMBER
W=POINTER TABLE WORD NUMBER

Figure D-6. Priority Scheme for Maximum I/O Expansion

D.2.9 DSR WRITING PROCEDURE

The recommended method for writing a DSR uses a defined, consistent structure. The approach breaks up the DSR into unique processing steps. Each step is controlled by a branch vector, similar to the operation of a hardware state controller. The branch vector serves as the next state identifier. This technique is not suitable for all DSR's. DSR's that do not have several unique processing steps are inappropriate for this convention.

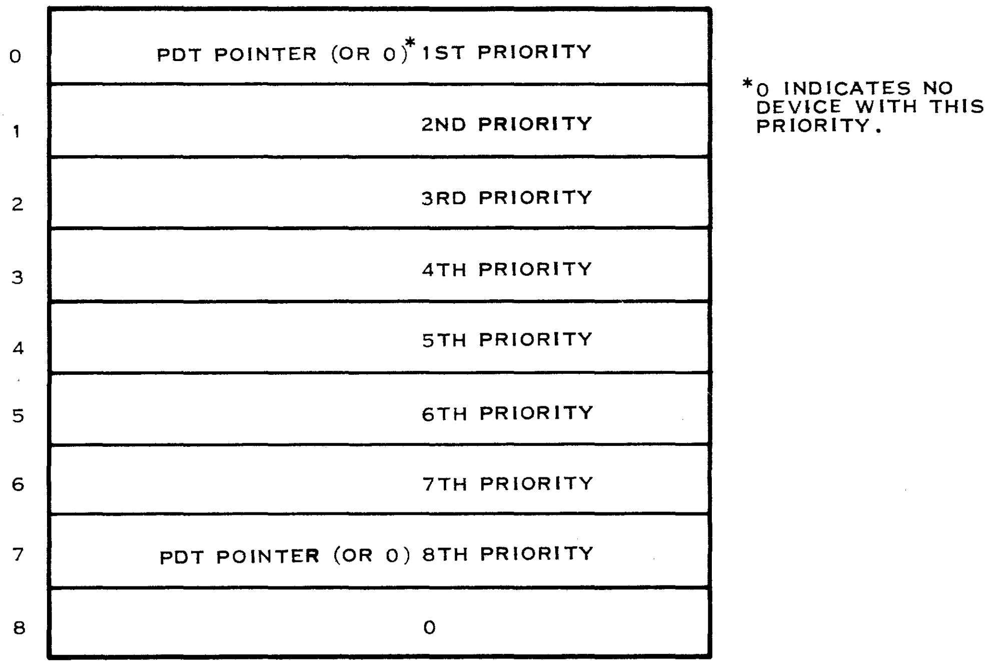
D.2.9.1 PREPARATION. Before designing a DSR, understand completely how the hardware interface works. Research sufficient interface documentation (usually available) to develop a familiarity with the interface. When developing new hardware, design both the interface and DSR before constructing either one. Very often, a simple hardware addition can save a considerable amount of software, and vice-versa. In some cases, special features that look desirable when only considering the hardware may actually make the software more complicated and are best eliminated.

D.2.9.2 REENTRANT DSR'S. DX980 DSR's can service multiple identical devices with one copy of the procedure. A data base consisting of several I/O tables described earlier is required for each individual device. The DSR's also appear to service multiple devices simultaneously because when an I/O interrupt occurs, it is serviced very quickly for the interrupting device. Once the interrupt service has begun, however, another interrupt cannot be serviced until the service for the current one has been completed.

All registers and DSR local storage are lost between interrupts for a device. All data needed for a subsequent execution of a DSR is saved in a temporary storage area of the device's Physical Device Table (PDT). This temporary storage area is whatever size is needed by the DSR and begins at the end of those entries required by the I/O Manager. Do not save data locally in a DSR even when only one device is serviced by it. If a second device is ever added, the DSR will not work. Furthermore, PDT storage usually takes the same amount of code and memory as local DSR storage. Therefore, whether or not the DSR's appear to be reentrant, code them as if they were; with all data storage being done in the PDT.

D.2.9.3 STRUCTURING THE DSR. First determine how many interrupt entry points will be necessary. To do this requires breaking the I/O transfer operation into its major components. This is not always an obvious decision. Hold the number of states to a reasonable level.

For example, consider the high-speed paper tape punch interface. A write operation involves sending the requested data on a character basis to the interface. The DSR is initially in an idle state that clears interrupts and returns control back to the system. When an output request is initiated by the IO Manager, the DSR sends a character to the interfaces, sets its next state to one that continues sending characters upon interrupt, and exits normally.


0
PDT POINTER (OR 0)* 1ST PRIORITY
1
2ND PRIORITY
2
3RD PRIORITY
3
4TH PRIORITY
4
5TH PRIORITY
5
6TH PRIORITY
6
7TH PRIORITY
7
PDT POINTER (OR 0) 8TH PRIORITY
8
0
*0 INDICATES NO DEVICE WITH THIS PRIORITY.

(A)130253

Figure D-7. DMAC Expansion Table PDT Pointer

When the DSR senses that it is about to send the last character, it changes state to exit the DSR with an End-of-Record exit. When this state is executed, the state is again set to idle and the DSR is exited. This procedure involves changing the state of the DSR several times. Upon an interrupt entry, the DSR only needs to clear the interrupt and branch via the next state vector.

Figure D-8 and table D-5 provide a flowchart and listing, respectively, of the paper tape punch DSR. Following that, figure D-9 and table D-6 provide a more complex example, the data terminal DSR.

D.2.9.4 CODING PDT BUILDER. When configuring the DX980 system at Initial Program Load (IPL) time include all the hardware configuration that is going to be used on the total system. This allows adding a device by adding only the PDT pointer to PDT branch tables since these tables already exist. The PDT Builder utility then has access to these tables using pointers and labels defined in table D-7. Figure D-10 outlines the steps required to add the new PDT using the PDT Builder utility. The remainder of this paragraph describes these events.

01/16/74 AUTUFLOW CHART SET - PAGE 01

CHART TITLE - PAPER TAPE PUNCH DEVICE SERVICE ROUTINE

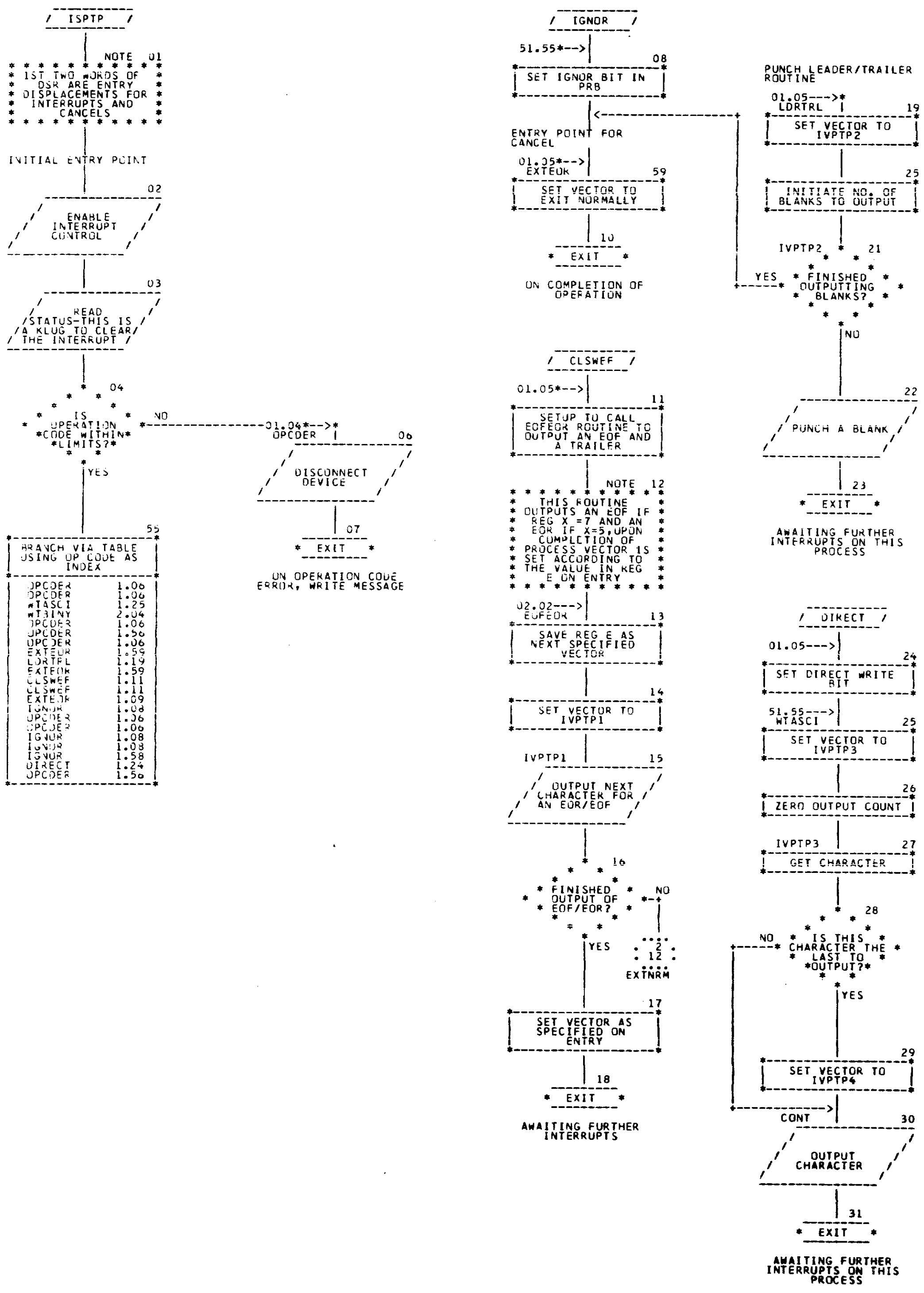

ISPTP
NOTE 01
1ST TWO WORDS OF DSR ARE ENTRY DISPLACEMENTS FOR INTERRUPTS AND CANCELS
INITIAL ENTRY POINT
02
ENABLE INTERRUPT CONTROL
03
READ STATUS-THIS IS A KLUG TO CLEAR THE INTERRUPT
04
IS OPERATION CODE WITHIN LIMITS?
NO
YES
01.04*-->
OPCDER
06
DISCONNECT DEVICE
07
EXIT
ON OPERATION CODE ERROR, WRITE MESSAGE
55
BRANCH VIA TABLE USING OP CODE AS INDEX
OPCDER 1.06
OPCDER 1.06
WTASCI 1.25
WTBINY 2.04
OPCDER 1.06
OPCDER 1.56
OPCDER 1.06
EXTEOR 1.59
LDRTRL 1.19
EXTEOR 1.59
CLSWEF 1.11
CLSWEF 1.11
EXTEOR 1.09
IGNOR 1.08
OPCDER 1.06
OPCDER 1.06
IGNOR 1.08
IGNOR 1.08
IGNOR 1.58
DIRECT 1.24
OPCDER 1.56
IGNOR
51.55*-->
08
SET IGNOR BIT IN PRB
ENTRY POINT FOR CANCEL
01.05*-->
EXTEOR
59
SET VECTOR TO EXIT NORMALLY
10
EXIT
ON COMPLETION OF OPERATION
CLSWEF
01.05*-->
11
SETUP TO CALL EOFEOR ROUTINE TO OUTPUT AN EOF AND A TRAILER
NOTE 12
THIS ROUTINE OUTPUTS AN EOF IF REG X =7 AND AN EOR IF X=5, UPON COMPLETION OF PROCESS VECTOR IS SET ACCORDING TO THE VALUE IN REG E ON ENTRY
02.02--->
EOFEOR
13
SAVE REG E AS NEXT SPECIFIED VECTOR
14
SET VECTOR TO IVPTP1
IVPTP1
15
OUTPUT NEXT CHARACTER FOR AN EOR/EOF
16
FINISHED OUTPUT OF EOF/EOR?
NO
YES
2
12
EXTNRM
17
SET VECTOR AS SPECIFIED ON ENTRY
18
EXIT
AWAITING FURTHER INTERRUPTS
PUNCH LEADER/TRAILER ROUTINE
01.05--->*
LDRTRL
19
SET VECTOR TO IVPTP2
25
INITIATE NO. OF BLANKS TO OUTPUT
IVPTP2
21
FINISHED OUTPUTTING BLANKS?
YES
NO
22
PUNCH A BLANK
23
EXIT
AWAITING FURTHER INTERRUPTS ON THIS PROCESS
DIRECT
01.05--->
24
SET DIRECT WRITE BIT
51.55--->
WTASCI
25
SET VECTOR TO IVPTP3
26
ZERO OUTPUT COUNT
IVPTP3
27
GET CHARACTER
28
IS THIS CHARACTER THE LAST TO OUTPUT?
NO
YES
29
SET VECTOR TO IVPTP4
CONT
30
OUTPUT CHARACTER
31
EXIT
AWAITING FURTHER INTERRUPTS ON THIS PROCESS

Figure D-8. Paper Tape Punch DSR (Sheet 1 of 2)

```
01/16/74                              AUTOFLOW CHART SET -                                  PAGE 02

CHART TITLE - PAPER TAPE PUNCH DEVICE SERVICE ROUTINE
```

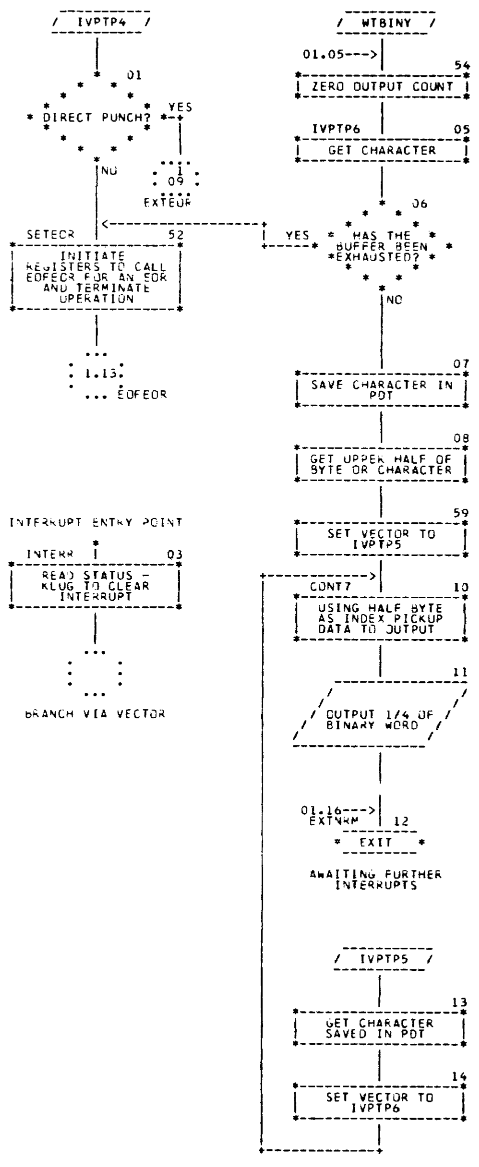

IVPTP4
01
DIRECT PUNCH?
YES
NO
1 09
EXTEOF
SETECR 52
INITIATE REGISTERS TO CALL EOFEOR FOR AN EOR AND TERMINATE OPERATION
1.13
EOFEOR
INTERRUPT ENTRY POINT
INTERR 03
READ STATUS - KLUG TO CLEAR INTERRUPT
BRANCH VIA VECTOR
WTBINY
01.05--->
54
ZERO OUTPUT COUNT
IVPTP6 05
GET CHARACTER
06
HAS THE BUFFER BEEN EXHAUSTED?
YES
NO
07
SAVE CHARACTER IN PDT
08
GET UPPER HALF OF BYTE OR CHARACTER
59
SET VECTOR TO IVPTP5
CONT7 10
USING HALF BYTE AS INDEX PICKUP DATA TO OUTPUT
11
OUTPUT 1/4 OF BINARY WORD
01.16--->
EXTNRM 12
EXIT
AWAITING FURTHER INTERRUPTS
IVPTP5
13
GET CHARACTER SAVED IN PDT
14
SET VECTOR TO IVPTP6

Figure D-8. Paper Tape Punch DSR (Sheet 2 of 2)

Table D-5. Paper Tape Punch DSR Listing

```
SAP R2LC
PTP DSR - INITIAL ENTRY                                          SHEET  0001

      *                                                   943117
      *
      *
      *  TEXAS INSTRUMENTS,INC.     PART#943117-9901
      *
      *
           IDT  IDPTP
      ***************************************************************************
      *
      *
      *  TITLE      = HIGH SPEED PAPER TAPE PUNCH DEVICE SERVICE R
      *               'ISPTP'
      *
      *  AUTHOR     = ERNEST BONUGLI, 01/74
      *
      *  SYSTEM     = DX980
      *
      * ABSTRACT          THIS ROUTINE PROVIDES THE INTERFACE
      *                   BETWEEN THE PAPER TAPE PUNCH CONTROLLER
      *                   AND THE DX980 SYSTEM.
      * MAJOR FUNCTIONS   1)  THE HIGH SPEED PAPER TAPE PUNCH
      *                       DSR PUNCHES DATA ON 8 LEVEL PAPER
      *                       TAPES IN AN ASCII, BINARY, OR DIRECT
      *                       MODE.
      *                   2)  ACSII RECORDS ARE WRITTEN AS ONE
      *                       FRAME PER CHARACTER, THE DATA
      *                       PUNCHED DIRECTLY FROM THE DATA BUF-
      *                       FER WITH NO CONVERSIONS.
      *                   3)  EACH RECORD IS TERMINATED WITH A
      *                       READER OFF CHARACTER (X-OFF) AND 4
      *                       DELETE (RUBOUT) CHARACTERS.
      *                   4)  BINARY RECORDS ARE PUNCHED AS FOUR
      *                       FRAMES PER WORD.
      *                   5)  ON WRITE DIRECT NO END OF RECORD
      *                       INDICATOR IS PUNCHED.
      * ERRORS            NO PUNCH ERRORS ARE DETECTABLE
      *                   OPERATION CODE ERRORS - ABORT
      *
      *  INPUT      = THE FOLLOWING ARGUMENT IS PASSED
      *
      *                   (PDT#, INITIAL/CANCEL CALL INDICATOR)
      *
      *             ALL DSR DO A CALL TO ISDSRI TO OBTAIN THE
      *             FOLLOWINGS:
      *             E = 0
      *             M = PRB #
      *             B = PDT #
      *             A AND X = OPERATION CODE
      *             S = UTILITY ENTRY
      *                      BRANCH INTO INITIAL OR CANCEL ENTR
      *
      *  OUTPUT     = NONE
      *
```

Table D-5. Paper Tape Punch DSR Listing (Continued)

```
SAP R2LC
PTP DSR - INITIAL ENTRY                                           SHEET  0002

             *   ROUTINES
             *   CALLED     = I/O UTILITIES
             *
             *   SVCS USED = NONE
             *
             *
             *
             **********************************************************
```

Table D-5. Paper Tape Punch DSR Listing (Continued)

```
SAP R2LC
PTP DSR - INITIAL ENTRY                                            SHEET  0003

                 PEJ
                 REF   ISDSRT
                 DEF   ISPTP
         ***************************************************
         *
         *     I/O UTILITY EQUATES
         *
         ***************************************************
         IOCOM   OPD   >C7C7,5
0021     ABORT   EQU   >21
0009     OPDERR  EQU   9
         BIT     FRM   5,2,5,4
0001     SET     EQU   1
0000     CLEAR   EQU   0
0001     PDT     EQU   1
0000     PRB     EQU   0
0002     LDT     EQU   2
0002     SKIP0   EQU   2
0003     SKIP1   EQU   3
         REG     FRM   5,3,2,6
0004     LOAD    EQU   4
0005     STORE   EQU   5
         IOBUS   FRM   5,7,4
0006     READ    EQU   6
0007     WRITE   EQU   7
0000     DATA    EQU   0
0001     CMMD    EQU   1
         CHAR    FRM   5,11
000A     PUT     EQU   >A
000B     GET     EQU   >B
         EXIT    FRM   7,9
0020     NORM    EQU   >20
0024     EOR     EQU   >24
```

Table D-5. Paper Tape Punch DSR Listing (Continued)

```
SAP R2LC
PTP DSR - INITIAL ENTRY                                      SHEET  0004

                PEJ
       ************************************************************
       *
       *  REGISTER EQUATES
       *
       ************************************************************
0000   A        EQU  0
0001   E        EQU  1
0002   X        EQU  2
0003   M        EQU  3
0004   S        EQU  4
0005   L        EQU  5
0006   B        EQU  6
0007   P        EQU  7
0001   BP       EQU  1
       ************************************************************
       *
       *  PRB
       *
       ************************************************************
0000   PRBSFL EQU  0             SYSTEM FLAG WORD
       *
0002   PRBEOF EQU  2              EOF SYSTEM FLAG
0003   PRBOPI EQU  3              OPERATION IGNORED FLAG
       *
0001   PRBOPC EQU  1             OPERATION CODE (RIGHT HLAF)
0002   PRBDRL EQU  2             DATA RECORD LENGTH
       ************************************************************
       *
       * PDT
       *
       ************************************************************
0001   PDTFLG EQU  1             FLAG WORD
       *
000F   DIRCTP EQU  15            DIRECT PUNCH BIT
       *
0012   OUTCNT EQU  18            OUTPUT COUNT USED BY UTILITIES
0012   TEMP   EQU  18
0013   TEMP1  EQU  19
0014   VECT   EQU  20            WRITE VECTOR USED ON INTERRUPT ENT
       ************************************************************
       *
       * WRITE COMMANDS FOR PTP INTERFACE
       *
       ************************************************************
0200   WCDISC EQU  >0200         DISCONNECT DEVICE
0100   WCENIN EQU  >0100         ENABLE INTERRUPTS
       ************************************************************
       *
       * CONTROL FOR EOFEOR SUBROUTINE
       *
       ************************************************************
0007   WTEOF  EQU  7
```

Table D-5. Paper Tape Punch DSR Listing (Continued)

```
SAP R2LC
PTP DSR - INITIAL ENTRY                                                    SHEET  0005

          0005  0148  WTEOR   EQU  5
                0149          HED  PTP DSR - INITIAL ENTRY
```

Table D-5. Paper Tape Punch DSR Listing (Continued)

```
SAP R2LC
PTP DSR - INITIAL ENTRY                                               SHEET  0006

                         PEJ
                  ******************************************************
                  *
                  * A. - DETERMINE IF LEGAL OPCODE,TAKE ERROR PATHE IF NOT
                  * B. - BRANCH VIA BRANCH TABLE ACCORDING TO OPCODE
                  *
                  ******************************************************
         0000     ISPTP  EQU  $
  0000   C550            RMO  L,A             SAVE SVC RETURN #
  0001   7400            *BPL ISDSRI          INITIALIZE SUB PROGRAM
         0000
X 0002   0000
P 0003   0063            DATA INTERR          INTERRUPT ENTRY ADDRESS
P 0004   0026            DATA CANCEL          CANCEL ENTRY ADDRESS
  0005   1800            *LDM =WCENIN         ENABLE INTERRUPT CONTROL
  0006   0100
  0007   C7C7            IOCOM
  0008   3813            IOBUS WRITE,CMMD,M
  0009   C7C7            IOCOM
  000A   3013            IOBUS READ,CMMD,M
  000B   6F13            CPA  =19
  000C   CD80            SGE
  000D   781D            BRU  OPCDER
  000E   1201            LDX  $+2,X
  000F   C527            RMO  X,P
P 0010   002B            DATA OPCDER          0  READ ASCII
P 0011   002B            DATA OPCDER          1  READ BINARY
P 0012   004E            DATA WTASCI          2  WRITE ASCII
P 0013   0066            DATA WTBINY          3  WRITE BINARY
P 0014   002B            DATA OPCDER          4  REWIND
P 0015   002B            DATA OPCDER          5  BACKSPACE RECORD
P 0016   002B            DATA OPCDER          6  FORWARD SPACE RECORD
P 0017   0026            DATA EXTEOR          7  OPEN
P 0018   0042            DATA LDRTRL          8  OPEN REWIND
P 0019   0026            DATA EXTEOR          9  CLOSE
P 001A   0031            DATA CLSWEF          10 CLOSE WRITE EOF
P 001B   0031            DATA CLSWEF          11 WRITE EOF
P 001C   0026            DATA EXTEOR          12 CHANGE RECORD LENGTH
P 001D   0024            DATA IGNOR           13 READ DEVICE STATUS
P 001E   002B            DATA OPCDER          14 BACK SPACE FILE
P 001F   002B            DATA OPCDER          15 FORWARD SPACE FILE
P 0020   0024            DATA IGNOR           16 UNLOAD
P 0021   0024            DATA IGNOR           17 UNASSIGNED
P 0022   0024            DATA IGNOR           18 UNASSIGNED
P 0023   004D            DATA DIRECT          19 WRITE DIRECT
                         HED  PTP DSR
```

Table D-5. Paper Tape Punch DSR Listing (Continued)

```
SAP R2LC
PTP DSR                                                              SHEET  0007

                 0193          PEJ
          0024   0194   IGNOR  EQU  $
   0024   C7C7   0195          IOCOM               SET OPCODE IGNOR BIT
   0025   0803   0196          BIT  SET,PRB,PRBSFL,PRBOPI
          0026   0197   CANCEL EQU  $
          0026   0198   EXTEOR EQU  $
   0026   0000   0199          *LDA =EXTNRM
P  0027   0075
   0028   8114   0200          STA  VECT,BR
   0029   C7C7   0201          IOCOM               EXIT ON COMPLETION OF REQUEST
   002A   4800   0202          EXIT  EOR,0
          002B   0203   OPCDER EQU  $
   002B   0000   0204          *LDA =WCDISC
   002C   0200
   002D   C7C7   0205          IOCOM
   002E   3810   0206          IOBUS WRITE,CMMD,A
   002F   C7C7   0207          IOCOM
   0030   4200   0208          EXIT ABORT,OPDERR
          0031   0209   CLSWEF EQU  $
   0031   1707   0210          LDX  =WTEOF
   0032   0800   0211          *LDE =LDRTPL
P  0033   0042
                 0212          HED  PTP DSR - WRITE EOF/EOR
```

Table D-5. Paper Tape Punch DSR Listing (Continued)

```
SAP R2LC
PTP DSR - WRITE EOF/EOR                                              SHEET  0008

                   0213         REJ
                   0214  ***************************************************************
                   0215  *
                   0216  *INPUT IS E=NEXT ENTRY VECTOR ANDX=2 IF EOR OR X=4 IF EOF
                   0217  * A. - SET VECTOR TO IVPTP1
                   0218  * B. - GET CHARACTER
                   0219  * C. - IS IT LAST ONE TO PUNCH? ES THEN TO TO F
                   0220  * D. - OUTPUT CHARACTER
                   0221  * E. - EXIT NORMALLY
                   0222  * F. - SET VECTOR AS SPECIFIED ON ENTRY
                   0223  *
                   0224  ***************************************************************
            0034   0225  EOFEOR EQU $
     0034   8913   0226         STE  TEMP1,BR
     0035   0000   0227         *LDA =IVPTP1
   P 0036   0039
     0037   8114   0228         STA  VECT,BR
     0038   9112   0229         STX  TEMP,BR
            0039   0230  IVPTP1 EQU  $
     0039   1112   0231         LDX  TEMP,BR
     003A   023F   0232         LDA  EOF,X
     003B   C7C7   0233         IOCOM
     003C   3800   0234         IOBUS WRITE,DATA,A
     003D   4912   0235         DMT  TEMP,BR
     003E   7838   0236         BRU  EXTNRM
     003F   1113   0237         LDX  TEMP1,BR
     0040   9114   0238         STX  VECT,BR
     0041   7833   0239         BRU  EXTNRM
                   0240         HED  PTP DSR - PUNCH LEADER/TRAILER ROUTINE
```

Table D-5. Paper Tape Punch DSR Listing (Continued)

```
SAP R2LC
PTP DSR - PUNCH LEADER/TRAILER ROUTINE                             SHEET  0009

                    0241         PEJ
                    0242   ************************************************************
                    0243   *
                    0244   * A. - SET INTERRUPT VECTOR TO ENTER AT B
                    0245   * B. - DONE IT 100 TIMES? YES THEN GO TO F
                    0246   * C. - PUNCH A BLANK
                    0247   * D. - EXIT NORMALLY
                    0248   * E. - DISCONNECT DEVICE AND EXIT EOR
                    0249   *
                    0250   ************************************************************
              0042  0251   LDRTRL EQU   $
  0042  0000  0252               *LDA  =IVPTP2
P 0043  0047
  0044  8114  0253               STA   VECT,BR
  0045  0764  0254               LDA   =100
  0046  8112  0255               STA   TEMP,BR
              0047  0256   IVPTP2 EQU   $
  0047  4912  0257               DMT   TEMP,BR
  0048  7801  0258               BRU   $+2
  0049  78DC  0259               BRU   CANCEL
  004A  C7C7  0260               IOCOM
  004B  3801  0261               IOBUS WRITE,DATA,E
  004C  7828  0262               BRU   EXTNRM
                    0263         HED PTP DSR - WRITE ASCII
```

Table D-5. Paper Tape Punch DSR Listing (Continued)

```
SAP R2LC
PTP DSR - WRITE ASCII                                                  SHEET  0010

                  0264         REJ
                  0265   ***********************************************************
                  0266   *
                  0267   * A. - SET VECTOR
                  0268   * B. - ZERO PDT OUTPUT COUNT
                  0269   * C. - GET CHARACTER
                  0270   * D. - IS IT LAST CHARACTER TO WRITE ? NO THEN GO TO F
                  0271   * E. - SET VECTOR TO IVPTP4
                  0272   * F. - EXIT NORMALLY
                  0273   *
                  0274   ***********************************************************
           004D   0275   DIRECT EQU  $              ENTRY FOR DIRECT PUNCH
    004D   5101   0276          IMO  PDTFLG,BR      SET DIRECT BIT FLAG
           004E   0277   WTASCI EQU  $
    004E   0800   0278          @LDA =IVPTP3
  P 004F   0052
    0050   8114   0279          STA  VECT,BR
    0051   8912   0280          STE  OUTCNT,BR
           0052   0281   IVPTP3 EQU  $
    0052   C7C7   0282          IOCOM
    0053   5800   0283          CHAR GET,A
    0054   DB10   0284          TABO 0
    0055   7803   0285          BRU  CONT
    0056   0800   0286          @LDE =IVPTP4
  P 0057   005C
    0058   8914   0287          STE  VECT,BR
           0059   0288   CONT   EQU  $
    0059   C7C7   0289          IOCOM
    005A   3800   0290          IOBUS WRITE,DATA,A
    005B   7819   0291          BRU  EXTNRM
           005C   0292   IVPTP4 EQU  $
    005C   0101   0293          LDA  PDTFLG,BR
    005D   DB0F   0294          TABZ DIRCTP
    005E   78C7   0295          BRU  EXTEOR         BRANCH IF DOING DIRECT PUNCH
           005F   0296   SETEOR EQU  $
    005F   1705   0297          LDX  =WTEOR
    0060   0800   0298          @LDE =EXTEOR
  P 0061   0026
    0062   78D1   0299          BRU  EOFEOR
                  0300          HED PTP DSR - INTERRUPT ENTRY
```

Table D-5. Paper Tape Punch DSR Listing (Continued)

```
SAP R2LC
PTP DSR - INTERRUPT ENTRY                                                   SHEET  0011

                         PEJ
                 ************************************************************
                 *
                 * A. - CLEAR INTERRUPT
                 * B. - BRANCH VIA VECTOR
                 *
                 ************************************************************
           0063  INTERR EQU  $
  0063    C7C7           IOCOM
  0064    3010           IOBUS READ,CMMD,A
  0065    7014           BRU  *VECT,BR
                         HED PTP DSR - WRITE BINARY
```

Table D-5. Paper Tape Punch DSR Listing (Continued)

```
SAP R2LC
PTP DSR - WRITE BINARY                                              SHEET  0012

                          PEJ
                   ***********************************************************
                   *
                   * A. - GET CHARACTER FROM BUFFER
                   * B. - HAS BUFFER BEEN EXHAUSTED? YES GO TO IVPTP4
                   * C. - SAVE CHARACTER IN PDT
                   * D. - SET VECTOR TO IVPTP5 (STATEMENT G)
                   * E. - OUTPUT OBJECT CHAR
                   * F. - EXIT NORMALLY
                   * G. - GET NEXT OBJECT CHAR FORM PDT
                   * H. - SET VECTOR TO IVPTP6 (STATEMENT A)
                   * I. - GO TO F
                   *
                   ***********************************************************
        0066       WTBINY EQU  $
  0066  8012              STF  OUTCNT,BR
        0067       IVPTP6 EQU  $
  0067  C7C7              TOCOM
  0068  5800              CHAR GET,A
  0069  DB01              TABZ 1
  006A  78F4              BRU  SETEOF
  006B  8113              STA  TEMP1,BR
  006C  CA04              CRA  4
  006D  0800              *LDE =IVPTP5
P 006E  0077
        006F       CONT7  EQU  $
  006F  8914              STF  VECT,BR
  0070  3F0F              AND  =>F
  0071  C502              RMO  A,X
  0072  020F              LDA  TAB,X
  0073  C7C7              TOCOM
  0074  3800              TOBUS WRITE,DATA,A
        0075       IVPTP  EQU  $
        0075       EXTNRM EQU  $
  0075  C7C7              TOCOM
  0076  4000              EXIT NORM,0
        0077       IVPTP5 EQU  $
  0077  0113              LDA  TEMP1,BR
  0078  0800              *LDE =IVPTP6
P 0079  0067
  007A  78F4              BRU  CONT7
        007A       EOF    EQU  $-1
  007B  00FF              DATA  >FF
  007C  00FF              DATA  >FF
  007D  00FF              DATA  >FF
  007E  00FF       NULL   DATA >FF              DELETE
  007F  0093       XOFF   DATA >93              X-OFF
  0080  00AA              DATA >AA              *
  0081  00AF              DATA >AF              /
  0082  0010       TAB    DATA >10
  0083  0001              DATA >01
  0084  0002              DATA >02
  0085  0083              DATA >83
```

Table D-5. Paper Tape Punch DSR Listing (Continued)

```
SAP R2LC
PTP DSR - WRITE BINARY                                                  SHEET  0013

  0086  0004  0364             DATA >04
  0087  0015  0365             DATA >15
  0088  0016  0366             DATA >16
  0089  0097  0367             DATA >97
  008A  0098  0368             DATA >98
  008B  0019  0369             DATA >19
  008C  001A  0370             DATA >1A
  008D  009B  0371             DATA >9B
  008E  001C  0372             DATA >1C
  008F  009D  0373             DATA >9D
  0090  009E  0374             DATA >9E
  0091  001F  0375             DATA >1F
        000C  0376             END   ISPTP
```

Table D-5. Paper Tape Punch DSR Listing (Continued)

```
SAP R2LC
PTP DSR - WRITE BINARY                                                    SHEET  0014

   A         0003        ABORT     0021     R B         0006        BIT       8508
   BR        0001        CANCEL    0026       CHAR      8400     R  CLEAR     0000
   CLSWEF    0031        CMMD      0001       CONT      0050        CONT7     006F
   DATA      0000        DIRCTP    003F       DIRECT    004D        E         0001
   EOF       007A        EOFFOR    0034       EOR       0024        EXIT      8100
   EXTEOR    0026        EXTNRM    0075       GET       000B        IGNOR     0024
   INTERR    0063        IOBUS     8408       IOCOM     C7C7        ISDSRI    0000
   ISPTP     0000     R  IVPTP     0075       IVPTP1    0039        IVPTP2    0047
   IVPTP3    0052        IVPTP4    005C       IVPTP5    0077        IVPTP6    0067
   L         0005        LDRTRL    0042     R LDT       0002     R  LOAD      0004
   M         0003        NORM      0020     R NULL      007F        OPCDER    002B
   OPDERR    0009        OUTCNT    0012       P         0007     R  PDT       0001
   PDTFLG    0001        PRB       0000     R PRBDRL    0002     R  PRBEOF    0002
R  PRBOPC    0001        PRBOPI    0003       PRBSFL    0000     R  PUT       000A
   READ      0006        REG       84A0     R S         0004        SET       0001
   SETEOR    005F     R  SKIP0     0002     R SKIP1     0003     R  STORE     0005
   TAB       0082        TEMP      0012       TEMP1     0013        VECT      0014
   WCDISC    0200        WCENIN    0100       WRITE     0007        WTASCI    004E
   WTBINY    0066        WTEOF     0007       WTEOR     0005        X         0002
R  XOFF      007F

0000 ERRORS
```

01/17/74 AUTOFLOW CHART SET - PAGE 01

CHART TITLE - DATA TERMINAL DSR - 980A

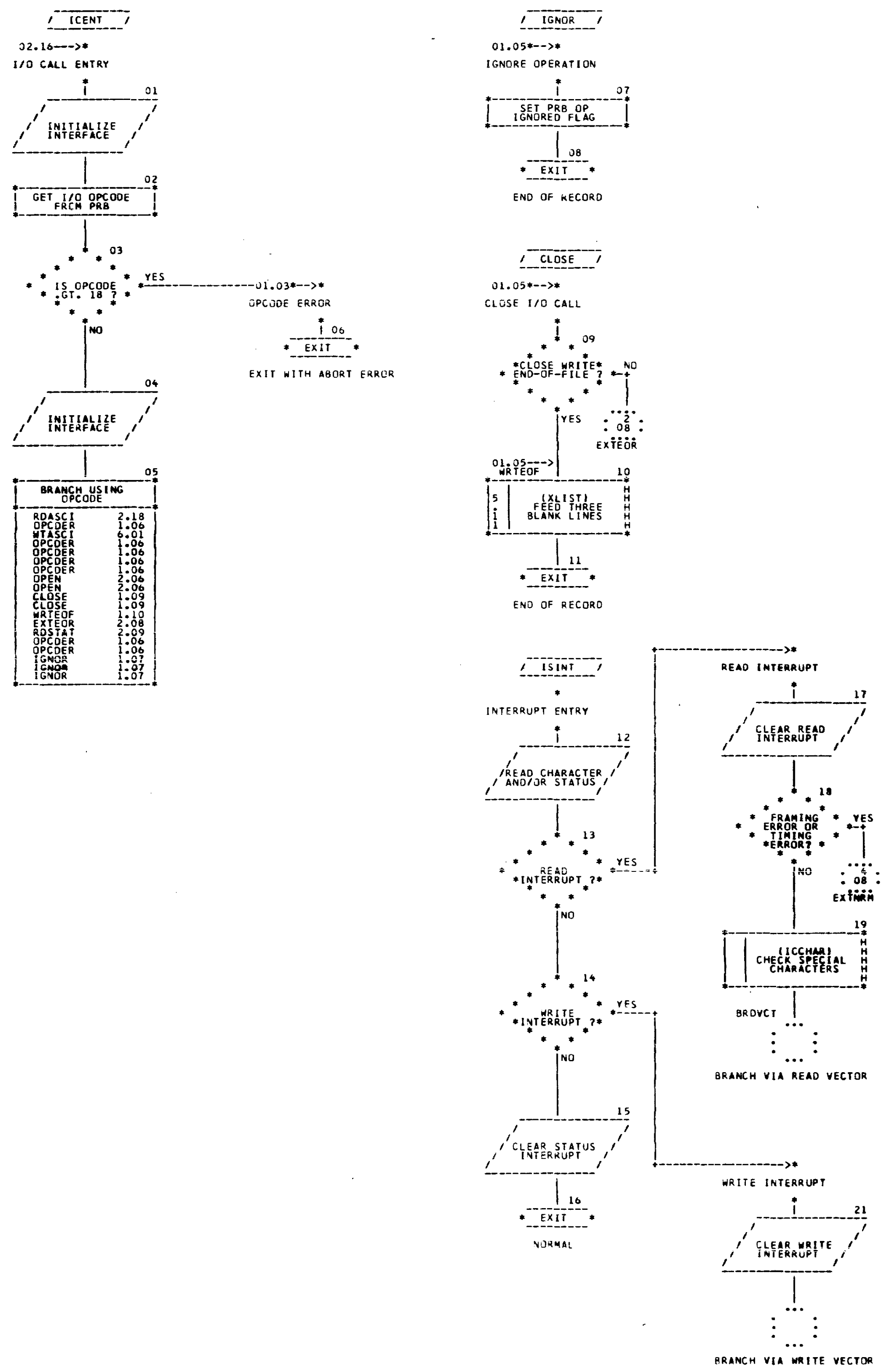


Figure D-9. Data Terminal DSR (Sheet 1 of 6)

01/17/74 AUTOFLOW CHART SET - PAGE 02

CHART TITLE - DATA TERMINAL DSR - 980A

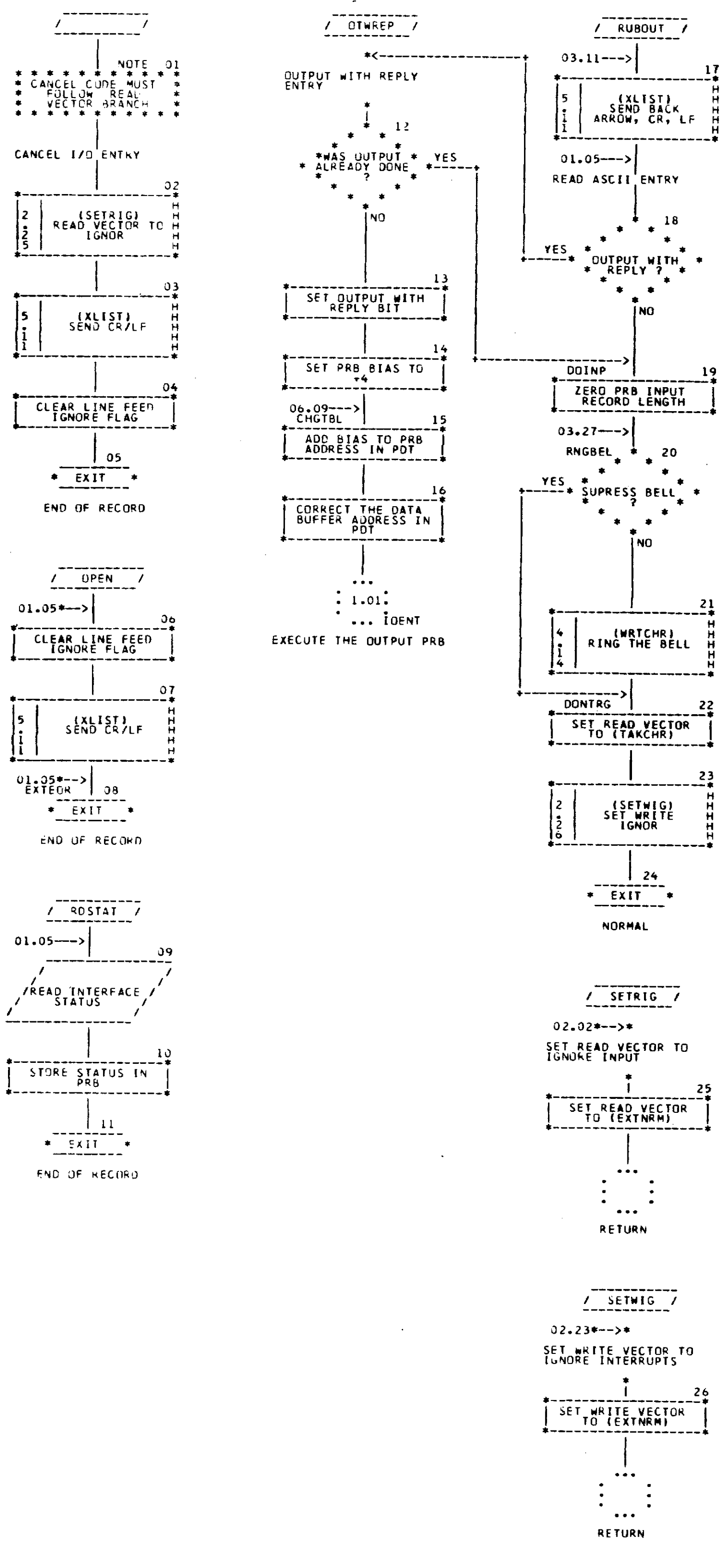
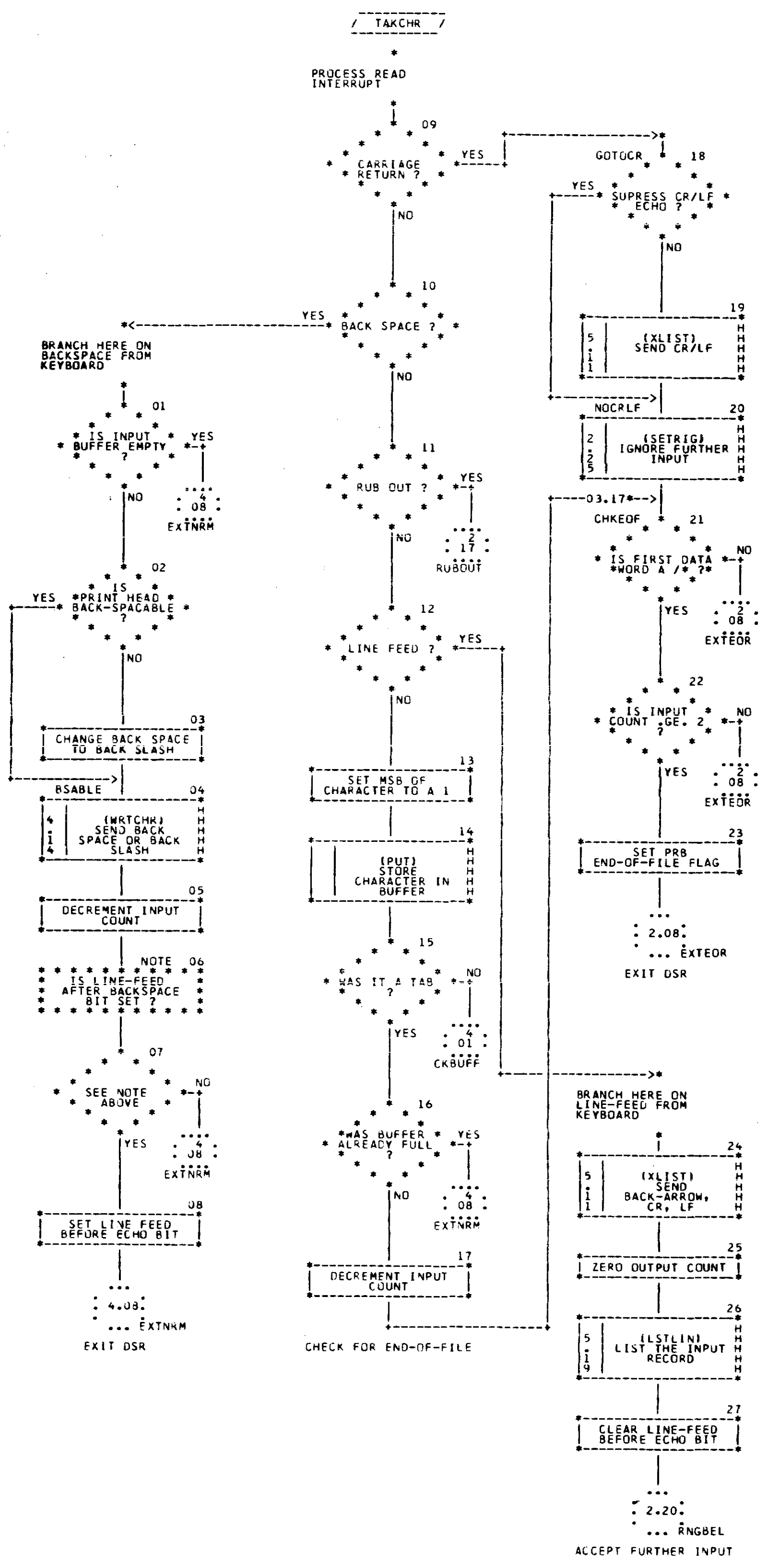


Figure D-9. Data Terminal DSR (Sheet 2 of 6)

01/17/74 AUTOFLOW CHART SET - PAGE 03

CHART TITLE - DATA TERMINAL DSR - 980A

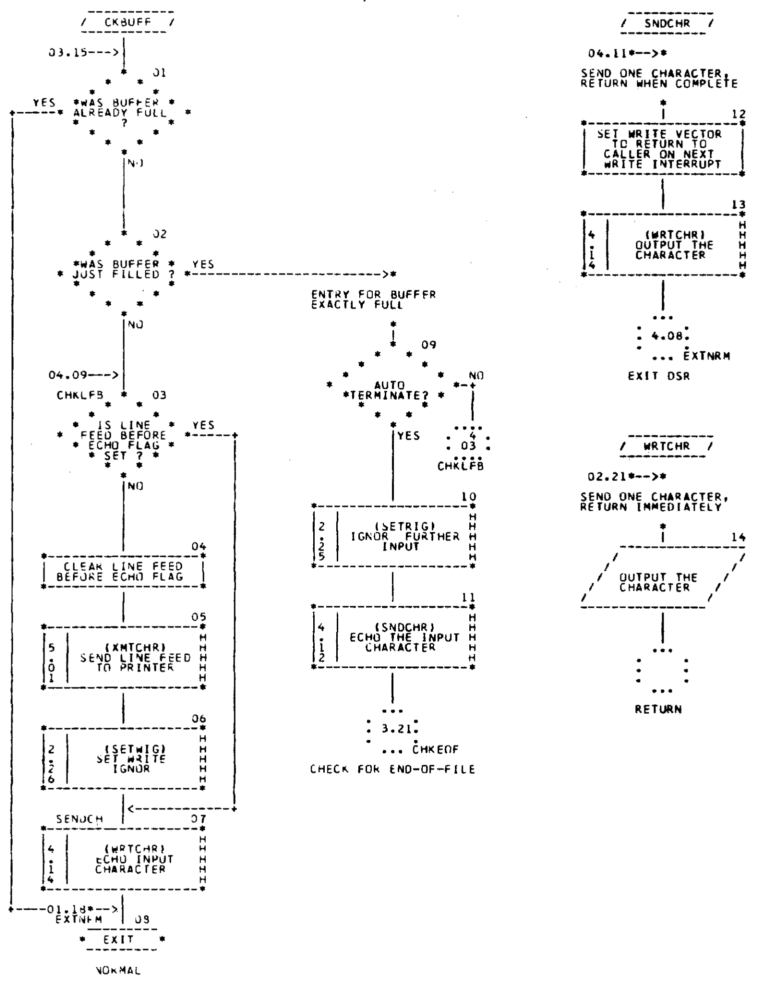


TAKCHR
PROCESS READ INTERRUPT
09 CARRIAGE RETURN ?
YES
NO
GOTOCR 18 SUPRESS CR/LF ECHO ?
19 (XLIST) SEND CR/LF
5.11
NOCRLF
20 (SETRIG) IGNORE FURTHER INPUT
2.25
03.17
CHKEOF 21 IS FIRST DATA WORD A /* ?
2 08 EXTEOR
22 IS INPUT COUNT .GE. 2 ?
23 SET PRB END-OF-FILE FLAG
2.08 EXTEOR
EXIT DSR
10 BACK SPACE ?
BRANCH HERE ON BACKSPACE FROM KEYBOARD
01 IS INPUT BUFFER EMPTY ?
4 08 EXTNRM
02 IS PRINT HEAD BACK-SPACABLE ?
03 CHANGE BACK SPACE TO BACK SLASH
BSABLE 04 (WRTCHR) SEND BACK SPACE OR BACK SLASH
4.14
05 DECREMENT INPUT COUNT
NOTE 06 IS LINE-FEED AFTER BACKSPACE BIT SET ?
07 SEE NOTE ABOVE
08 SET LINE FEED BEFORE ECHO BIT
4.08 EXTNRM
EXIT DSR
11 RUB OUT ?
2 17 RUBOUT
12 LINE FEED ?
13 SET MSB OF CHARACTER TO A 1
14 (PUT) STORE CHARACTER IN BUFFER
15 WAS IT A TAB ?
4 01 CKBUFF
16 WAS BUFFER ALREADY FULL ?
4 08 EXTNRM
17 DECREMENT INPUT COUNT
CHECK FOR END-OF-FILE
BRANCH HERE ON LINE-FEED FROM KEYBOARD
24 (XLIST) SEND BACK-ARROW, CR, LF
5.11
25 ZERO OUTPUT COUNT
26 (LSTLIN) LIST THE INPUT RECORD
5.19
27 CLEAR LINE-FEED BEFORE ECHO BIT
2.20 RNGBEL
ACCEPT FURTHER INPUT

Figure D-9. Data Terminal DSR (Sheet 3 of 6)

01/17/74 AUTOFLOW CHART SET - PAGE 04

CHART TITLE - DATA TERMINAL DSR - 980A



CKBUFF
03.15--->
01
WAS BUFFER ALREADY FULL ?
YES
NO
02
WAS BUFFER JUST FILLED ?
YES
NO
ENTRY FOR BUFFER EXACTLY FULL
09
AUTO TERMINATE?
NO
YES
4 03 CHKLFB
04.09--->
CHKLFB 03
IS LINE FEED BEFORE ECHO FLAG SET ?
YES
NO
04
CLEAR LINE FEED BEFORE ECHO FLAG
05
(XMTCHR) SEND LINE FEED TO PRINTER
5.01
06
(SETWIG) SET WRITE IGNOR
2.26
SENDCH 07
(WRTCHR) ECHO INPUT CHARACTER
4.14
01.18*--> EXTNRM 08
EXIT
NORMAL
10
(SETRIG) IGNOR FURTHER INPUT
2.25
11
(SNDCHR) ECHO THE INPUT CHARACTER
4.12
3.21 CHKEOF
CHECK FOR END-OF-FILE
SNDCHR
04.11*-->*
SEND ONE CHARACTER, RETURN WHEN COMPLETE
12
SET WRITE VECTOR TO RETURN TO CALLER ON NEXT WRITE INTERRUPT
13
(WRTCHR) OUTPUT THE CHARACTER
4.14
4.08 EXTNRM
EXIT DSR
WRTCHR
02.21*-->*
SEND ONE CHARACTER, RETURN IMMEDIATELY
14
OUTPUT THE CHARACTER
RETURN

Figure D-9. Data Terminal DSR (Sheet 4 of 6)

01/17/74 AUTOFLOW CHART SET - PAGE 05

CHART TITLE - DATA TERMINAL DSR - 980A

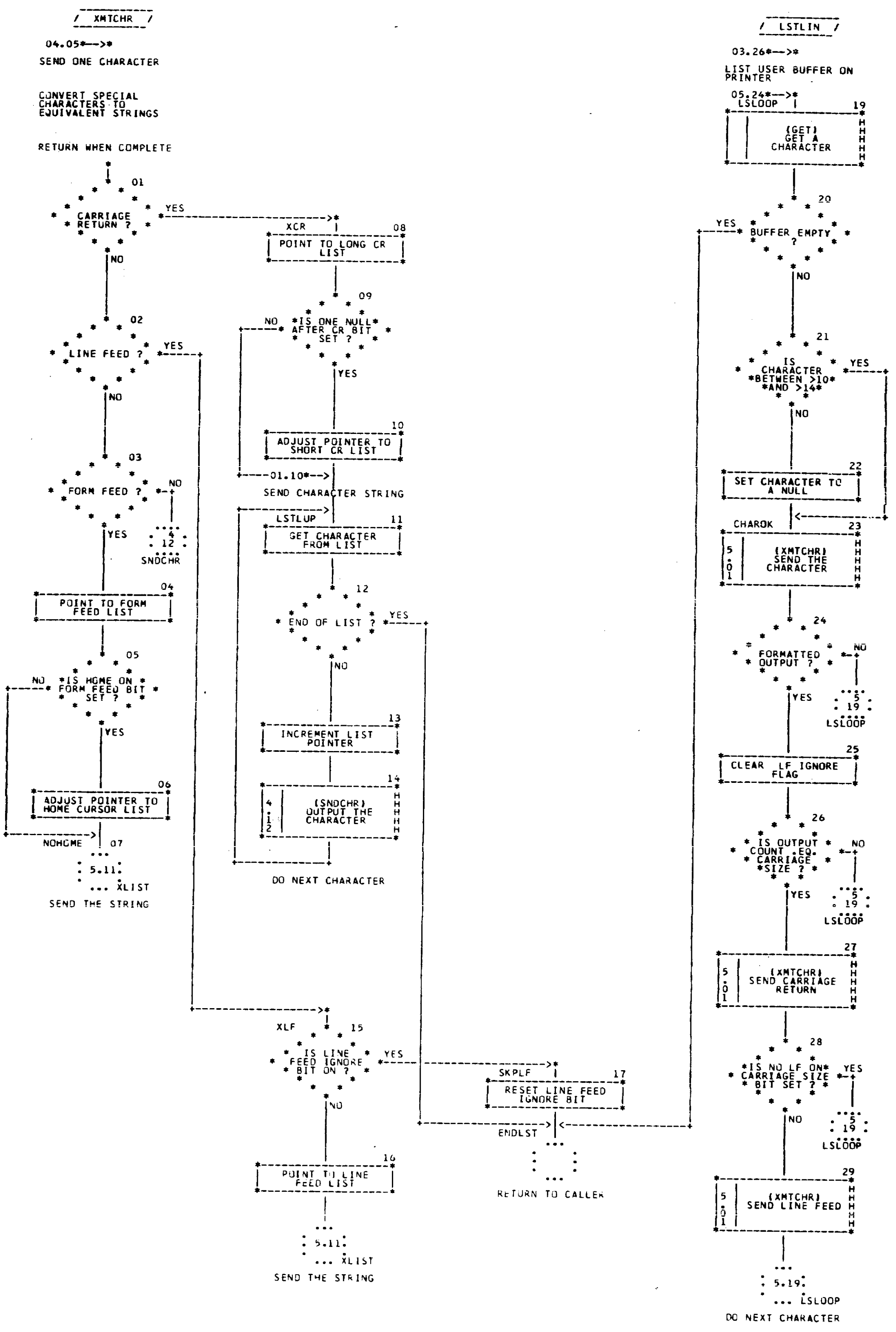

/ XMTCHR /
04.05*—>*
SEND ONE CHARACTER
CONVERT SPECIAL CHARACTERS TO EQUIVALENT STRINGS
RETURN WHEN COMPLETE
01 CARRIAGE RETURN ?
YES
NO
XCR 08
POINT TO LONG CR LIST
09 IS ONE NULL AFTER CR BIT SET ?
NO
YES
10 ADJUST POINTER TO SHORT CR LIST
01.10*—>
SEND CHARACTER STRING
LSTLUP 11
GET CHARACTER FROM LIST
12 END OF LIST ?
YES
NO
13 INCREMENT LIST POINTER
14 (SNDCHR) OUTPUT THE CHARACTER
4.12
DO NEXT CHARACTER
02 LINE FEED ?
YES
NO
03 FORM FEED ?
NO
YES
4 12 SNDCHR
04 POINT TO FORM FEED LIST
05 IS HOME ON FORM FEED BIT SET ?
NO
YES
06 ADJUST POINTER TO HOME CURSOR LIST
NOHOME 07
5.11 XLIST
SEND THE STRING
XLF 15 IS LINE FEED IGNORE BIT ON ?
YES
NO
16 POINT TO LINE FEED LIST
5.11 XLIST
SEND THE STRING
SKPLF 17
RESET LINE FEED IGNORE BIT
ENDLST
RETURN TO CALLER
/ LSTLIN /
03.26*—>*
LIST USER BUFFER ON PRINTER
05.24*—>*
LSLOOP 19
(GET) GET A CHARACTER
20 BUFFER EMPTY ?
YES
NO
21 IS CHARACTER BETWEEN >10 AND >14
YES
NO
22 SET CHARACTER TO A NULL
CHAROK 23
(XMTCHR) SEND THE CHARACTER
5.01
24 FORMATTED OUTPUT ?
NO
YES
5 19 LSLOOP
25 CLEAR LF IGNORE FLAG
26 IS OUTPUT COUNT .EQ. CARRIAGE SIZE ?
NO
YES
5 19 LSLOOP
27 (XMTCHR) SEND CARRIAGE RETURN
5.01
28 IS NO LF ON CARRIAGE SIZE BIT SET ?
YES
NO
5 19 LSLOOP
29 (XMTCHR) SEND LINE FEED
5.01
5.19 LSLOOP
DO NEXT CHARACTER

Figure D-9. Data Terminal DSR (Sheet 5 of 6)

01/17/74 AUTOFLOW CHART SET - PAGE 06

CHART TITLE - DATA TERMINAL DSR - 980A

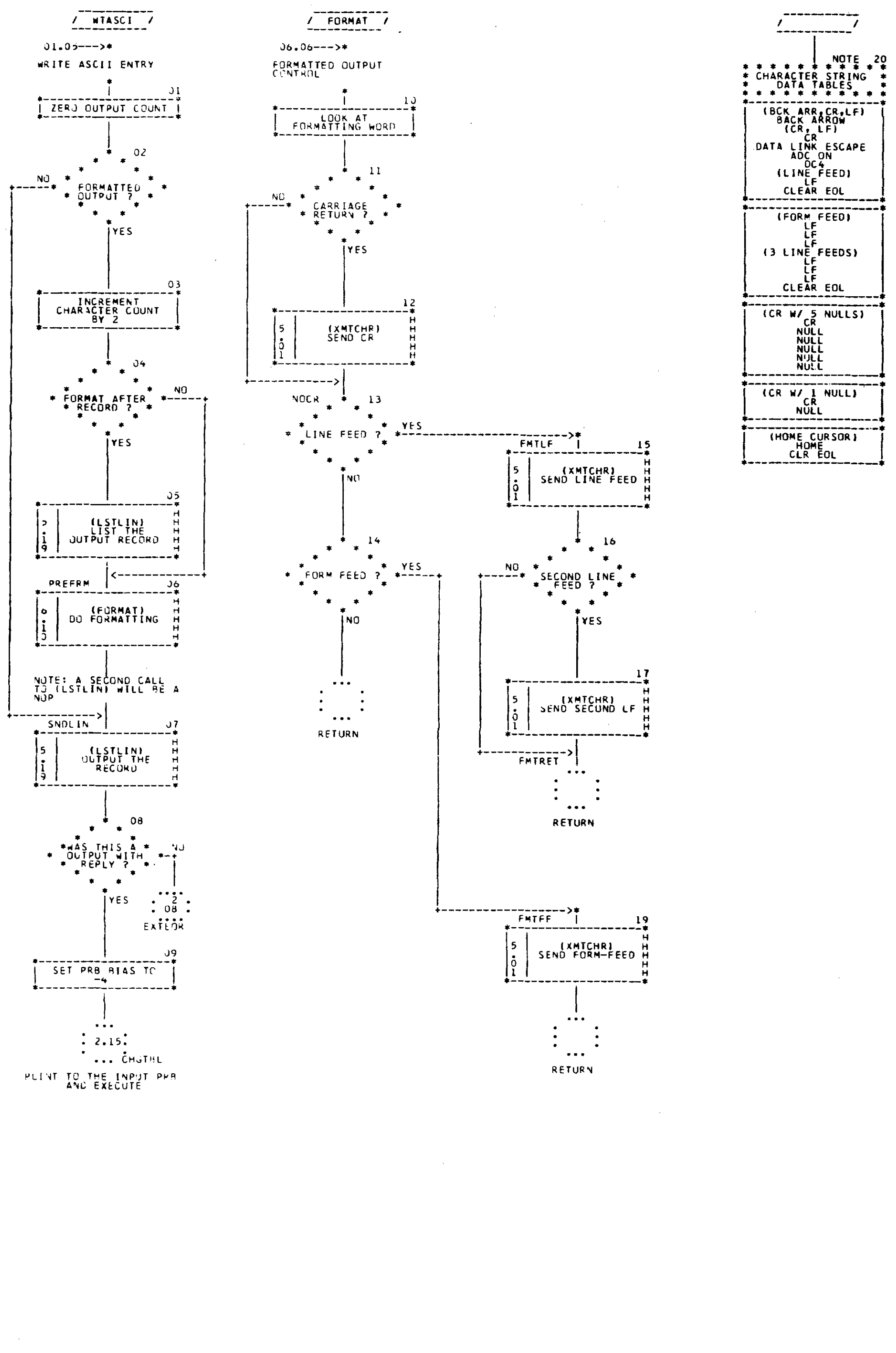


Figure D-9. Data Terminal DSR (Sheet 6 of 6)

Table D-6. Data Terminal DSR Listing

```
SAP R2LC
980A DATA TERMINAL DEVICE SERVICE ROUTINE                            SHEET  0001

        .#CREATE IDDT,SAP,'DATA TERMINAL DSR'.
              IDT  IDDT
              HED 980A DATA TERMINAL DEVICE SERVICE ROUTINE
        *
        *  TITLE=IDDT - 980A DATA TERMINAL DSR
        *
        *  AUTHOR=WAYNE DOHNAL
        *
        *  REVISIONS=NO REVISIONS
        *
        *  COMPUTER=980A,SAP
        *
        * ABSTRACT= THIS IS THE DEVICE SERVICE ROUTINE PROVIDING
        *           THE NECESSARY INTERFACE BETWE N THE DATA
        *           TERMINALS TIED TO COMMUNICATION MODULE
        *           AND THE DX980 SYSTEM.
        *
        *           DETAILED OPERATIONAL DOCUMENTATION IS SEPARATE
        *
        *  STATISTICS=THE MAXIMUM PATH WITH INTERRUPTS MASKED IS
        *             ????? US. PLUS TIME FOR THE FOLLOWING UTILITY
        *             FUNCTIONS:
        *
        *
        *
              HED  DATA TERMINAL DSR - GENERAL EQUATES
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - GENERAL EQUATES                                 SHEET  0002

                  PEJ
       *****************************************************************
       *
       *    EXTERNAL REFERENCES AND DEFINITIONS
       *
       *****************************************************************
       *
                  REF   ISDSRT
                  DEF   ISDT               ENTRY POINT
                  REF   ICCHAR             CONTROL CHARACTER ROUTINE
       *
       *****************************************************************
       *
       *    I/O UTILITY EQUATES
       *
       *****************************************************************
       *
       IOCOM      OPD   >C7C7,5
       BIT        FRM   5,2,5,4
0001   SFT        EQU   1
0000   CLEAR      EQU   0
0001   PDT        EQU   1
0000   PRB        EQU   0
0002   LDT        EQU   2
0002   SKIP0      EQU   2
0003   SKIP1      EQU   3
       REG        FRM   5,3,2,6
0004   LOAD       EQU   4
0005   STORE      EQU   5
       IOBUS      FRM   5,7,4
0006   READ       EQU   6
0007   WRITE      EQU   7
0000   DATA       EQU   0
       CHAR       FRM   5,11
000A   PUT        EQU   >A
000B   GET        EQU   >B
       EXIT       FRM   7,9
0020   NORM       EQU   >20
0024   EOR        EQU   >24
0021   ABORT      EQU   >21
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - GENERAL EQUATES                                    SHEET  0003

                  PEJ
             ******************************************************************
             *
             *     PRB WORD EQUATES
             *
             ******************************************************************
             *
0000         PRBSFL EQU   0                      PRB SYSTEM SET FLAGS
0001         PRBUFL EQU   1                      USER SET FLAGS
0001         PRBOPC EQU   1                      I/O OPCODE (RIGHT HALF)
0002         PRBDRL EQU   2                      DATA RECORD LENGTH
0003         PRBDBA EQU   3                      DATA RECORD LENGTH
             *
             ******************************************************************
             *
             *     PRB FLAG BIT EQUATES
             *
             ******************************************************************
             *
0002         PRBEOF EQU   2                      END OF FILE
0003         PRBOPI EQU   3                      OPERATION IGNORED
0001         PRBOWR EQU   1                      OUTPUT WITH REPLY
0002         PRBATM EQU   2                      AUTO RECORD TERMINATE
0003         PRBFAC EQU   3                      FORMATTED ASCII OUTPUT
0004         PRBSCR EQU   4                      SUPPRESS AUTO CR-LF ON INPUT
0003         PRBSBL EQU   3                      SUPPRESS BELL ON INPUT
             *
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - GENERAL EQUATES                                  SHEET  0004

                   PEJ
             ****************************************************************
             *
             *      PDT WORD EQUATES
             *
             ****************************************************************
             *
0001         PDTFLG EQU  1              FLAGS
000D         PDTPRB EQU  13             PRB ADDRESS
0010         PDTDBA EQU  16             DATA BUFFER ADR
0012         PDTOCT EQU  18         OUTPUT COUNT
0014         PDTMXO EQU  20         20 MAX CHARACTER OUTPUT/RECORD
0015         PDTSTA EQU  21         INITIAL STATUS
0016         PDTRVC EQU  22         READ VECTOR
0017         PDTWVC EQU  23         WRITE VECTOR
0018         TEMP1  EQU  24         TEMPORARY STORAGE
0019         TEMP2  EQU  25         TEMPORARY STORAGE
001A         TEMP3  EQU  26         TEMPORARY STORAGE
001B         PDTDEV EQU  27         DEVICE DESCRIPTION
             *
             ****************************************************************
             *
             *      PDT FLAG BIT EQUATES
             *
             ****************************************************************
             *
0000         PDTBSY EQU  0              DEVICE BUSY
0002         PDTLFE EQU  2              LF BEFORE ECHO AFTER BS BIT
0003         PDTLFT EQU  3              IGNORE NEXT LINE FEED BIT
0004         PDTFFH EQU  4              HOME ON FORM FEED BIT
0005         PDT1NL EQU  5              ONE NULL AFTER CR BIT
0006         PDTBCK EQU  6              BACK SPACABLE PRINT HEAD BIT
0007         PDTLFA EQU  7              AUTO LF ON CARRIAGE SIZE BIT
000E         PDTLFB EQU  14             LINE FEED BEFORE ECHO BIT
000F         PDTOWR EQU  15             OUTPUT WITH REPLY IN PROGRESS
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - GENERAL EQUATES                                    SHEET  0005

                   PEJ
             ************************************************************
             *
             *     REGISTER EQUATES
             *
             ************************************************************
             *
      0000   A        EQU   0
      0001   E        EQU   1
      0002   X        EQU   2
      0003   M        EQU   3
      0005   L        EQU   5
      0006   B        EQU   6
      0007   P        EQU   7
      0001   BR       EQU   1
             *
             ************************************************************
             *
             *     CHARACTER EQUATES
             *
             ************************************************************
             *
      001F   CLR      EQU   >1F
      001A   CURUP    EQU   >1A                CURSOR UP
      0000   NULL     EQU   >0
      0002   HOME     EQU   >2
      0007   BELL     EQU   >07
      0008   BAKSP    EQU   >08
      000A   LF       EQU   >0A
      000C   FF       EQU   >C
      000D   CR       EQU   >0D
      0010   DLE      EQU   >10
      0014   DC4      EQU   >14
      0017   CLRLIN   EQU   >17                CLEAR LINE
      003A   RDCON    EQU   >3A
      005C   BKSLSH   EQU   >5C
      005F   BCKARR   EQU   >5F
      007F   RUBOUT   EQU   >7F
      0089   TAB      EQU   >89
      AFAA   ENDFIL   EQU   >AFAA
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - GENERAL EQUATES                          SHEET  0008

                     PEJ
              ************************************************************
              *
              *    OUTPUT FORMAT CONTROL FLAG BIT EQUATES
              *
              ************************************************************
              *
000C          POSFRM EQU  12                  FORMAT AFTER RECORD
000D          CRFORM EQU  13                  CARRIAGE RETURN
000E          LFFORM EQU  14                  LINE FEED
000F          FFFORM EQU  15                  FORM FEED
000F          LF2FRM EQU  15                  2ND LINE FEED
              *
              ************************************************************
              *
              *    OTHER EQUATES
              *
              ************************************************************
              *
0003          LDTOPC EQU  3
0009          ILLOPC EQU  9             ILLEGAL OPERATION
              *
                     HED  DATA TERMINAL DSR - I/O CALL ENTRY
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - I/O CALL ENTRY                                    SHEET  0007

                    0192             REJ
            000F    0193    ISDT     EQU  $
  0000      C550    0194             RMO  L,A              SAVE SVC RETURN @
  0001      7430    0195             @BRL ISDSRT           INITIALIZE SUB PROGRAM
            0000
X 0002      0000
P 0003      0031    0196             DATA ISINT                 INTERRUPT ENTRY
P 0004      004B    0197             DATA ISRST                 RESET ENTRY
  0005      7072    0198             BRL  SETRIG           INITIALIZE READ VECTOR TO IGNOR
  0006      7074    0199             BRL  SETWIG           INITIALIZE WRITE VECTOR TO IGNORE
            0007    0200    IOENT    EQU  $                     STARTING POINT FOR I/O CALLS
  0007      C7C7    0201             IOCOM                      GET I/O OPCODE
  0008      2001    0202             REG  LOAD,A,PRB,PRBOPC
  0009      3FFF    0203             AND  =>FF
  000A      C502    0204             RMO  A,X
  000B      6F12    0205             CPA  =18
  000C      CD80    0206             SGE                        LEGAL OPCODE ?
  000D      781B    0207             BRU  OPCDER                NO, FATAL ERROR
  000E      0115    0208           LDA  PDTSTA,BR     GET INITIAL STATUS
  000F      C7C7    0209             IOCOM                      INITIALIZE INTERFACE
  0010      3B00    0210             IOBUS WRITE,DATA,A
  0011      0201    0211             LDA  $+2,X
  0012      C507    0212             RMO  A,P                   BRANCH VIA TABLE
P 0013      0069    0213             DATA RDASCI                00 READ ASCII
P 0014      0026    0214             DATA OPCDER                01 READ BINARY
P 0015      013A    0215             DATA WTASCI                02 WRITE ASCII
P 0016      0026    0216             DATA OPCDER                03 WRITE BINARY
P 0017      0026    0217             DATA OPCDER                04 REWIND
P 0018      0026    0218             DATA OPCDER                05 BACK SPACE RECORD
P 0019      0026    0219             DATA OPCDER                06 FORWARD SPACE RECORD
P 001A      0051    0220             DATA OPEN                  07 OPEN
P 001B      0051    0221             DATA OPEN                  08 OPEN REWIND
P 001C      002B    0222             DATA CLOSE                 09 CLOSE
P 001D      002B    0223             DATA CLOSE                 10 CLOSE WRITE END-OF-FILE
P 001E      002D    0224             DATA WRTEOF                11 WRITE END OF FILE
P 001F      0055    0225             DATA EXTEOR                12 CHANGE RECORD LENGTH
P 0020      0163    0226             DATA RDSTAT                13 READ DEVICE STATUS
P 0021      0026    0227             DATA OPCDER                14 BACK SPACE FILE
P 0022      0026    0228             DATA OPCDER                15 FORWARD SPACE FILE
P 0023      002F    0229             DATA IGNOR                 16 UNLOAD
P 0024      002F    0230             DATA IGNOR                 17
P 0025      002F    0231             DATA IGNOR                 18
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - I/O CALL ENTRY                                  SHEET  0008

                              PEJ
          0026        OPCDER  EQU   $                      FATAL ERROR ENTRY
  0026    C7C7                IOCOM                        FATAL ERROR
  0027    4200                EXIT  ABORT,ILLOPD
          0028        IGNOR   EQU   $
  0028    C7C7                IOCOM                        SET PRB IGNORE FLAG
  0029    0803                BIT   SET,PRB,PRBSFL,PRBOPI
  002A    782A                BRU   EXTEOR
          002B        CLOSE   EQU   $
  002B    CCC2                SEV   X                      CLOSE WRITE EOF ?
  002C    782B                BRU   EXTEOF                 NO
          002D        WRTEOF  EQU   $
  002D    0000                *LDA  =LF3LST                YES, SEND 3 LINE FEEDS
P 002E    01C9
  002F    7429                BRL   *XXLIST
  0030    7824                BRU   EXTEOR
                      *
                              HED   DATA TERMINAL DSR - INTERRUPT ENTRY POINT
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - INTERRUPT ENTRY POINT                        SHEET 0009

                          PEJ
      0031     ISINT      EQU   $                   INTERRUPT ENTRY POINT
 0031 1701                LDX   =1                  INITIALIZE TO CLEAR INTERRUPT
 0032 C7C7                IOCOM                     READ INTERRUPT I.D.
 0033 3000                IORUS READ,DATA,A
 0034 0B00                TABZ  0                   READ INTERRUPT ?
 0035 780A                BRU   RDINT               YES
 0036 0B01                TABZ  1                   WRITE INTERRUPT ?
 0037 7804                BRU   WRTINT              YES
 0038 CA43                CRX   3                   ASSUME STATUS CHANGE INTERRUPT
 0039 C7C7                IOCOM
 003A 3802                IORUS WRITE,DATA,X
 003B 7866                BRU   EXTNRM
      003C     WRTINT     EQU   $
 003C CA44                CRX   4                   CLEAR WRITE INTERRUPT
 003D C7C7                IOCOM
 003E 3802                IORUS WRITE,DATA,X
 003F 7017                BRU   *PDTWVC,BR          BRANCH VIA WRITE VECTOR
      0040     RDINT      EQU   $
 0040 CA41                CRX   1                   CLEAR READ INTERRUPT
 0041 C7C7                IOCOM
 0042 3802                IORUS WRITE,DATA,X
 0043 0B03                TABZ  3                   TEST FRAMING ERROR
 0044 785D                BRU   EXTNRM
 0045 0B04                TABZ  4                   TEST TIMING ERROR
 0046 785B                BRU   EXTNRM
 0047 3F7F                AND   =>7F                MASK STATUS BITS
 0048 7400                *BRL  IOCHAR              CHECK SPECIAL CHARACTERS
      0001
X0049 0000
 004A 7016                BRU   *PDTRVC,BR          NO, BRANCH TO READ VECTOR
               *
                          HED   DATA TERMINAL DSR - OPEN AND OPEN REWIND
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP 02LC
DATA TERMINAL DSR - OPEN AND OPEN REWIND                          SHEET  0010

                 0280          PEJ
                 0281   * NOTE: ISRST MUST FOLLOW @BRL TCCHAR & BRU *PDTRVC,BR
           004B  0282   ISRST  EQU   $                   CANCEL I/O ENTRY
      004B 702C  0283          BRL   SETRIG              SET READ VECTOR TO IGNORE
      004C 000B  0284          LDA   XXBKAR
      004D 740B  0285          BRL   *XXLIST             SEND CR/LF
      004E C7C7  0286          IOCOM                     CLEAR LINE FEED IGNORE BIT
      004F 0383  0287          PIT   CLEAR,PDT,PDTDEV,PDTLFI
      0050 7804  0288          BRU   EXTEOR              EXIT THE DSR
                 0289   *
           0051  0290   OPEN   EQU   $
      0051 C7C7  0291          IOCOM                     CLEAR IGNORE LINE FEED FLAG
      0052 0383  0292          PIT   CLEAR,PDT,PDTDEV,PDTLFI
      0053 0003  0293          LDA   XXCRLF
      0054 7404  0294          BRL   *XXLIST             SEND CR/LF
           0055  0295   EXTEOR EQU   $
      0055 C7C7  0296          IOCOM                     DSR END-OF-RECORD EXIT
      0056 4B00  0297          EXIT  EOR,0
                 0298   *
                 0299   *
                 0300   *
    P 0057 018D  0301   XXCRLF DATA  XCRLF
    P 0058 018C  0302   XXBKAR DATA  XBCKAR
    P 0059 010B  0303   XXLIST DATA  XLIST
                 0304   *
                 0305   *
                 0306          HED   DATA TERMINAL DSR - READ ASCII
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - READ ASCII                                        SHEET  0011

                              PEJ
            005A      OTWREP  EQU   $
    005A    C7C7              IOCOM                          SEE IF ALREADY THROUGH LOOP
    005B    121F              BIT   SKIP0,PDT,PDTFLG,PDTOWR
    005C    780F              BRU   DOINP                    YES, GET THE REPLY
    005D    C7C7              IOCOM                          NO, SET OUTPUT WITH REPLY BIT
    005E    0A1F              BIT   SET,PDT,PDTFLG,PDTOWR
    005F    0704              LDA   =4                       SET UP PRB BIAS
            0060      CHGTBL  EQU   $                        SET UP NEW PRB AND DB ADRS
    0060    C083              RAD   A,M                      SET UP NEW PRB ADDRESS
    0061    C530              RMO   M,A
    0062    810C              STA   PDTPRB,RR                SAVE IN PDT
    0063    C7C7              IOCOM                          GET NEW DATA BUFFER ADDRESS
    0064    2003              REG   LOAD,A,PRB,PRBDBA
    0065    8110              STA   PDTDBA,RR                SAVE IN PDT
    0066    78A0              BRU   IOENT
            0067      RUBOUT  EQU   $                        ENTER HERE IF RUBOUT INPUT
    0067    00F0              LDA   XXBKAR
    0068    74F0              BRL   *XXLIST                  SEND BACK ARROW, CR, LF
            0069      RDASCI  EQU   $
    0069    C7C7              IOCOM                          OUTPUT WITH REPLY ?
    006A    1011              BIT   SKIP0,PRB,PRBUFL,PRBOWR
    006B    78EF              BRU   OTWREP                   YES
            006C      DOINP   EQU   $
    006C    C7C7              IOCOM                          ZERO INPUT RECORD LENGTH
    006D    2902              REG   STORE,E,PRB,PRBDRL
            006E      RNGBEL  EQU   $
    006E    C7C7              IOCOM                          SUPRESS BELL ?
    006F    1013              BIT   SKIP0,PRB,PRBUFL,PRBSBL
    0070    7802              BRU   DONTRG                   YES
    0071    0707              LDA   =BELL
    0072    7079              BRL   WRTCHR                   WRITE THE CHARACTER
            0073      DONTRG  EQU   $
    0073    0000              @LDA  =TAKCHR                  SET UP READ VECTOR
P   0074    016B
    0075    8116              STA   PDTRVC,RR                TO ACCEPT NEXT CHARACTER
    0076    7004              BRL   SETWIG
    0077    782A              BRU   EXTNRM
                      *
            0078      SETRIG  EQU   $
    0078    1005              LDX   XEXNRM                   SET READ VECTOR
    0079    9116              STX   PDTRVC,RR                TO IGNORE FURTHER CHARACTERS
    007A    C557              RMO   L,P
                      *
                      *
            007B      SETWIG  EQU   $
    007B    0002              LDA   XEXNRM                   SET WRITE VECTOR
    007C    8117              STA   PDTWVC,RR                TO IGNORE FURTHER INTERRUPTS
    007D    C557              RMO   L,P
                      *
P   007E    00A2      XEXNRM  DATA  EXTNRM
                              HED   D.T. DSR-CHARACTER INTERRUPT INPUT , CONTINUED
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
D.T. DSR-CHARACTER INTERRUPT INPUT , CONTINUED                          SHEET  0012

                 0358            PEJ
          007F   0359   CLRSCR   EQU   $
  007F    706C   0360            BRL   WRTCHR
  0080    C7C7   0361            IOCOM
  0081    021F   0362            BIT   CLEAR,PDT,PDTFLG,PDTLFB
  0082    7BE6   0363            BRU   RDASCT
          0083   0364   TAKCH1   EQU   $
  0083    6F7F   0365            CPA   =RUBOT              RUB OUT ?
  0084    C0A0   0366            SNE
  0085    7BE1   0367            BRU   RUBOUT              YES
  0086    0P5B   0368            SABO  8                   SET MSB OF CHARACTER TO 1
  0087    C7C7   0369            IOCOM                     STORE IN USER'S BUFFER
  0088    5000   0370            CHAR  PUT,A
  0089    6789   0371            CPL   =TAB                TAB ?
  008A    CD20   0372            SEQ
  008B    7807   0373            BRU   CKBUFF              NO
  008C    DB01   0374            TABZ  1                   WAS BUFFER ALREADY FULL ?
  008D    7814   0375            BRU   EXTNRM              YES, DON'T DO ANYTHING
  008E    C7B6   0376            REX   M,B                 NO, POINT BASE TO PRB
  008F    4902   0377            DMT   PRBDRL,BR           REDUCE CHARACTER COUNT BY 1
  0090    7800   0378            BRU   $+1                 NECESSARY NOP IN CASE OF SKIP
  0091    C786   0379            REX   M,B                 RESTORE BASE TO PDT
  0092    781C   0380            BRU   CHKEOF              CHECK FOR END-OF-FILE
          0093   0381   CKBUFF   EQU   $
  0093    DB01   0382            TABZ  1                   WAS BUFFER ALREADY FULL ?
  0094    780D   0383            BRU   EXTNRM              YES, DON'T DO ANYTHING
  0095    DB00   0384            TABZ  0                   DID WE JUST FILL BUFFER ?
  0096    780C   0385            BRU   FULLBF              YES, SEE IF AUTO TERMINATE
          0097   0386   CHKLFB   EQU $
  0097    C7C7   0387            IOCOM                     SEE IF LF BEFORE ECHO
  0098    1A1F   0388            BIT   SKIP1,PDT,PDTFLG,PDTLFB
  0099    7807   0389            BRU   SENDCH              NO, ECHO THE CHARACTER
  009A    811B   0390            STA   TEMP1,BR            SAVE THE ECHO CHARACTER
  009B    C7C7   0391            IOCOM                     CLEAR LF BEFORE ECHO FLAG
  009C    021F   0392            BIT   CLEAR,PDT,PDTFLG,PDTLFB
  009D    070A   0393            LDA   =LF
  009E    7052   0394            BRL   XMTCHR              SEND THE LF
  009F    70DB   0395            BRL   SETWIG              SET WRITE VECTOR TO IGNORE
  00A0    011B   0396            LDA   TEMP1,BR            RESTORE THE ECHO CHARACTER
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
D.T. DSR-CHARACTER INTERRUPT INPUT , CONTINUED                        SHEET  0013

                          PEJ
      00A1       SENDCH   EQU   $
00A1  704A                BRL   WRTCHR              ECHO CHARACTER TO TERMINAL
      00A2       IVWDT    EQU   $
      00A2       IVRDT    EQU   $
      00A2       EXTNRM    EQU  $
00A2  C7C7                IOCOM                     NORMAL DSR EXIT
00A3  4000                EXIT  NORM,0
      00A4       FULLBF   EQU   $                   COME HERE IF BUFFER JUST FULL
00A4  C7C7                IOCOM                     AUTO-TERMINATE ?
00A5  1812                BIT   SKIP1,PRB,PRBUFL,PRBATM
00A6  78F0                BRU   CHKLFR              NO
00A7  70D0                BRL   SETRIG              DISCARD FURTHER CHARACTERS
00A8  703F                BRL   SNDCHR              ECHO THE CHARACTER
00A9  7805                BRU   CHKEOF              CHECK FOR END-OF-FILE
                 *
                 *
                 *
      00AA       GOTOCR   EQU   $                   ENTER HERE ON CR INPUT
00AA  00AC                LDA   XYCRLF
00AB  C7C7                IOCOM                     SUPRESS CR/LF ECHO ?
00AC  1814                BIT   SKIP1,PRB,PRBUFL,PRBSCR
00AD  705D                BRL   XLIST               SEND CR/LF
00AE  70C0                BRL   SETRIG              IGNORE FURTHER INPUT
      00AF       CHKEOF   EQU   $
00AF  0510                LDA   *PDTDBA,BR          LOOK AT FIRST WORD IN BUFFER
00B0  6800                *CPA  =ENDFIL             END-OF-FILE ?
00B1  AFAA
00B2  CD20                SEQ
00B3  78A1                BRU   EXTEOR              NO, EXIT DSR
00B4  C7C7                IOCOM                     LOOK AT INPUT COUNT
00B5  2002                REG   LOAD,A,PRB,PRBDRL
00B6  6F01                CPA   =1                  IS IT AT LEAST 2 ?
00B7  CD00                SLT
00B8  789C                BRU   EXTEOR              NO, EXIT DSR
00B9  C7C7                IOCOM                     YES, SET EOF FLAG IN PRB
00BA  0802                BIT   SET,PRB,PRBSFL,PRBEOF
00BB  7899                BRU   EXTEOR              EXIT DSR
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
D.T. DSR-CHARACTER INTERRUPT INPUT , CONTINUED                    SHEET  0014

                           PEJ
      00BC        BACKSP   EQU  $                     ENTER HERE IF BACKSPACE INPUT
00BC  C7C7                 IOCOM                      LOOK AT INPUT COUNT
00BD  2102                 REG  LOAD,E,PRB,PRBDRL
00BE  CC81                 SNZ  E                     IS IT ZERO ?
00BF  78E2                 BRU  EXTNRM                YES, EXIT DSR
00C0  C7C7                 IOCOM                      IS PRINTER BACK SPACABLE ?
00C1  1BB6                 BIT  SKIP1,PDT,PDTDEV,PDTBCK
00C2  075C                 LDA  =BKSLSH               NO, USE BACK SLASH INSTEAD
00C3  7024                 BRL  SNDCHR                SEND BACK SPACE OR SLASH
00C4  70B6                 BRL  SETWIG
00C5  C7C7                 IOCOM                      LOOK AT INPUT COUNT
00C6  2102                 REG  LOAD,E,PRB,PRBDRL
00C7  C711                 RDE  E,E                   DECREMENT CHARACTER COUNT
00C8  C7C7                 IOCOM                      PUT BACK IN PRB
00C9  2902                 REG  STORE,E,PRB,PRBDRL
00CA  C7C7                 IOCOM                      SEE IF LF AFTER BS
00CB  1BB2                 BIT  SKIP1,PDT,PDTDEV,PDTLFE
00CC  7802                 BRU  BACKS1        DON'T SET LF BEFORE ECHO BIT
00CD  C7C7                 IOCOM                      SET LF BEFORE ECHO BIT
00CE  0A1F                 BIT  SET,PDT,PDTFLG,PDTLFB
      00CF        BACKS1   EQU  $
00CF  CC81                 SNZ  E
00D0  78D1                 BRU  EXTNRM
00D1  C7C7                 IOCOM              CRT?
00D2  1BB4                 BIT  SKIP1,PDT,PDTDEV,PDTFFH
00D3  78CF                 BRU  EXTNRM        NO
00D4  011B                 LDA  PDTDEV,RR
00D5  3800                 *AND =>FF          MASK WIDTH
00D6  00FF
00D7  2F03                 SUB  =>3
00D8  8702                 STA  $+3
00D9  0700                 LDA  =0
00DA  5800                 *DIV =$-$
00DB  0000
00DC  CC01                 SZE  E
00DD  78C4                 BRU  EXTNRM
00DE  071A                 LDA  =CURUP
00DF  78C1                 BRU  SENDCH
                  *
                  *
      00E0        LINFED   EQU  $                     ENTER HERE IF LF INPUT
00E0  0400                 *LDA  XXBKAR
P 00E1  005B
00E2  702B                 BRL  XLIST                 SEND BACK ARROW, CR, LF
00E3  8912                 STE  PDTOCT,RR             ZERO OUTPUT COUNT
00E4  7032                 BRL  LSTLIN                LIST THE INPUT RECORD
00E5  C7C7                 IOCOM                      CLEAR LF BEFORE ECHO BIT
00E6  021F                 BIT  CLEAR,PDT,PDTFLG,PDTLFB
00E7  78B6                 BRU  RNGREL                ACCEPT FURTHER INPUT
                  *
                  *
                           HED  DATA TERMINAL DSR - CHARACTER TRANSMISSION
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - CHARACTER TRANSMISSION                     SHEET  0015

                        PEJ
                  *****************************************************************
                  *
                  * SUBROUTINE TO SEND ONE CHARACTER FROM THE A-REGISTER.
                  * RETURN IS WHEN CHARACTER TRANSMISSION IS COMPLETED.
                  *
                  *****************************************************************
                  *
      00E8        SNDCHR EQU   $
00E8  C551               RMO   L,E
00E9  8917               STE   PDTWVC,BR          SAVE RETURN IN WRITE VECTOR
00EA  7001               BRL   WRTCHR             OUTPUT THE CHATACTER
00EB  78B6               BRU   EXTNRM             EXIT DSR
                  *
                  *****************************************************************
                  *
                  * SUBROUTINE TO SEND ONE CHARACTER FROM THE A-REGISTER.
                  * RETURN IS IMMEDIATE.
                  *
                  *****************************************************************
      00EC        WRTCHR EQU   $
00EC  3F7F               AND   =>7F
00ED  0B51               SABO 1                   SET INTERFACE TRANSMIT BIT
00EE  C7C7               IOCOM
00EF  3B00               IOBUS WRITE,DATA,A
00F0  C557               RMO   L,P                RETURN
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - CHARACTER TRANSMISSION                         SHEET  0016

                         PEJ
                  ***************************************************************
                  *
                  * SUBROUTINE TO SEND SINGLE CHARACTERS.  USAGE IS SIMILAR
                  * TO SNDCHR, EXCEPT SPECIAL CHARACTERS ARE CONVERTED TO
                  * THE NECESSARY CHARACTER STRINGS.  MAY ALSO BE ENTERED AT
                  * XLIST TO SEND A PRE-SPECIFIED STRING.
                  *
                  ***************************************************************
       00F1       XMTCHR EQU   $
  00F1 3F7F              AND   =>7F
  00F2 6F0D              CPA   =CR                  CARRIAGE RETURN ?
  00F3 CDA0              SNE
  00F4 7811              BRU   XCR                  YES
  00F5 6F0A              CPA   =LF                  LINE FEED ?
  00F6 CDA0              SNE
  00F7 7809              BRU   XLF                  YES
  00F8 6F0C              CPA   =FF                  FORM FEED ?
  00F9 CD20              SEQ
  00FA 78E0              BRU   SNDCHR               NO, JUST TRANSMIT AS IS
  00FB 0000              *LDA  =FFLST
P 00FC 01C6
  00FD C7C7              TOCOM                      SEE IF HOME ON FORM FEED
  00FE 1384              BIT   SKIP0,PDT,PDTDEV,PDTFFH
  00FF 2714              ADD   =HMLST-FFLST         CHANGE CHARACTER LIST ADR
  0100 780A              BRU   XLIST                SEND FORM FEED EQUIVALENT
       0101       XLF    EQU   $
  0101 011B              LDA   PDTDEV,BR
  0102 D803              TABZ  PDTLFT               SEE IF LF IGNORE BIT SET
  0103 7810              BRU   SKPLF                IGNORE LINE FEED
  0104 0234              LDA   XLFLST               SEND LINE FEED EQUIVALENT
  0105 7805              BRU   XLIST
       0106       XCR    EQU   $
  0106 0000              *LDA  =CR1LST
P 0107 01D0
  0108 C7C7              TOCOM                      SEE IF ONE NULL AFTER CR
  0109 1385              BIT   SKIP0,PDT,PDTDEV,PDT1NL
  010A 2707              ADD   =CR2LST-CR1LST
       010B       XLIST  EQU   $                    SEND THE CHARACTER LIST
  010B 8119              STA   TEMP2,BR             SAVE LIST INDEX
  010C C551              RMO   L,E
  010D 891A              STE   TEMP3,BR             RETURN ADR
       010E       LSTLUP EQU   $
  010E 0519              LDA   *TEMP2,BR            GET A CHARACTER
  010F D801              TABZ  1                    END OF LIST ?
  0110 7D1A              BRU   *TEMP3,BR            YES, RETURN
  0111 5119              IMO   TEMP2,BR             NO, INCREMENT INDEX
  0112 70D5              BRI   SNDCHR               AND SEND THE CHARACTER
  0113 78FA              BRU   LSTLUP               DO NEXT CHARACTER
       0114       SKPLF  EQU   $
  0114 D843              SABZ  PDTLFT               RESET LF IGNORE BIT
  0115 811B              STA   PDTDEV,BR
  0116 C557              RMO   L,P                  RETURN
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - CHARACTER TRANSMISSION                        SHEET  ØØ17

                          REJ
        Ø117      LSTLIN   EQU   $                      LIST USER BUFFER ON TERMINAL
  Ø117  C55Ø               RMO   L,A
  Ø118  8118               STA TEMP1,BR                 SAVE RETURN
  Ø119  C7C7               TOCOM                        GET A CHARACTER
  Ø11A  58ØØ               CHAR GET,A
  Ø11B  DBØ1               TABZ 1                       BUFFER EMPTY ?
  Ø11C  7D1B               BRU   *TEMP1,BR              YES, EXIT ROUTINE
  Ø11D  3F7F               AND   =>7F
  Ø11E  6F1Ø               CPA   =>1Ø                   IF THE CHARACTER IS IN THE
  Ø11F  CDCØ               SLE                          RANGE OF >1Ø TO >14
  Ø12Ø  78Ø3               BRU   CHAROK                 CHANGE IT TO A NULL
  Ø121  6F14               CPA   =>14
  Ø122  CDØØ               SLT
  Ø123  Ø7ØØ               LDA   =NULL
        Ø124      CHAROK   EQU   $
  Ø124  7ØCC               BRL   XMTCHR                 NO, SEND THE CHARACTER
  Ø125  C7C7               TOCOM                        SEE IF FORMATTED OUTPUT
  Ø126  1813               BIT   SKIP1,PRB,PRBUFL,PRBFAC
  Ø127  78F1               BRU   LSTLIN+2
  Ø128  Ø11B               LDA   PDTDEV,BR              LOOK AT DEVICE DESCRIPTION
  Ø129  DB43               SABZ PDTLFT                  CLEAR THE LF IGNORE FLAG
  Ø12A  811B               STA   PDTDEV,BR
  Ø12B  3FFF               AND   =>FF                   CHECK CARRIAGE SIZE
  Ø12C  6912               CPA   PDTOCT,BR              EQUAL TO OUTPUT COUNT ?
  Ø12D  CD2Ø               SEQ
  Ø12E  78EA               BRU   LSTLIN+2               NO
  Ø12F  Ø7ØD               LDA   =CR
  Ø13Ø  7ØCØ               BRL   XMTCHR                 SEND CARRIAGE RETURN
  Ø131  Ø8Ø7               LDA   XLFLST                 POINT TO LINE FEED LIST
  Ø132  C7C7               TOCOM                        DID DEVICE DO ITS OWN LF ?
  Ø133  1387               BIT   SKIPØ,PDT,PDTDEV,PDTLFA
  Ø134  27Ø1               ADD   =1                     YES, SKIP THE LF
  Ø135  7ØD5               BRL   XLIST                  SEND LF OR LF/CLR EOL
  Ø136  C7C7               TOCOM                        SET THE LF IGNORE FLAG
  Ø137  Ø883               BIT   SET,PDT,PDTDEV,PDTLFI
  Ø138  78EØ               BRU   LSTLIN+2               GO DO MORE
                  *
                  *
P Ø139  Ø1C3      XLFLST   DATA  LFLST
                  *
                           HED   DATA TERMINAL DSR - WRITE ASCII
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - WRITE ASCII                                          SHEET  0018

                   0603          PEJ
          013A     0604  WTASCI  EQU   $
 013A     8912     0605          STF   PDTOCT,BR          ZERO OUTPUT COUNT
 013B     C7C7     0606          IOCOM                    FORMATTED OUTPUT ?
 013C     1813     0607          BIT   SKIP1,PRB,PRBUFL,PRBFAC
 013D     7806     0608          BRU   SNDLIN             NO FORMATTING
 013E     5112     0609          IMO   PDTOCT,BR          INCREMENT PAST FORMAT WORD
 013F     5112     0610          IMO   PDTOCT,BR
 0140     0510     0611          LDA   *PDTDBA,BR         LOOK AT FORMATTING WORD
 0141     DB0C     0612          TABZ  POSFRM             POST FORMATTING ?
 0142     70D4     0613          BRL   LSTLIN             YES, LIST THE RECORD
 0143     7007     0614          BRL   FORMAT             DO FORMATTING
          0144     0615  SNDLIN  EQU   $
 0144     70D2     0616          BRL   LSTLIN             DUMMY CALL IF ALREADY DONE
 0145     0101     0617          LDA   PDTFLG,BR          LOOK AT PDT FLAGS
 0146     DB1F     0618          TABO  PDTOWR             WAS THIS AN OUTPUT WITH REPLY
 0147     7821     0619          BRU   EXEORX             NO, WE ARE FINISHED
 0148     07FC     0620          LDA   *-4                RESTORE ORIGINAL PRB ADR
 0149     7C00     0621          @BRU  CHGTBL             RESET PRB AND DB ADRS
P 014A    0060
                   0622  *
                   0623          HED   DATA TERMINAL DSR - FORMATTED OUTPUT / READ STA
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - FORMATTED OUTPUT / READ STATUS                    SHEET  ØØ19

               Ø624          PEJ
         Ø14B  Ø625  FORMAT  EQU  $
 Ø14B  C55Ø    Ø626          RMO  L,A
 Ø14C  811B    Ø627          STA  TEMP1,BR              SAVE RETURN
 Ø14D  Ø51Ø    Ø628          LDA  *PDTDBA,BR            LOOK AT FORMAT WORD
 Ø14E  DB1D    Ø629          TABO CRFORM                CARRIAGE RETURN ?
 Ø14F  7BØ2    Ø63Ø          BRU  NOCR                  NO
 Ø15Ø  Ø7ØD    Ø631          LDA  =CR                   YES
 Ø151  7Ø9F    Ø632          BRL  XMTCHR                YES, SEND IT OUT
         Ø152  Ø633  NOCR    EQU  $
 Ø152  Ø51Ø    Ø634          LDA  *PDTDBA,BR            LOOK AT FORMAT WORD AGAIN
 Ø153  DBØF    Ø635          TABZ LFFORM                LINE FEED ?
 Ø154  78Ø3    Ø636          BRU  FMTLF                 YES
 Ø155  DBØF    Ø637          TABZ FFFORM                FORM FEED ?
 Ø156  78Ø9    Ø638          BRU  FMTFF                 YES
 Ø157  7D1B    Ø639          BRU  *TEMP1,BR             NOTHING AT ALL, RETURN
         Ø158  Ø64Ø  FMTLF   EQU  $                     SEND #1 LINE FEED
 Ø158  Ø7ØA    Ø641          LDA  =LF                   NO
 Ø159  7Ø97    Ø642          BRL  XMTCHR                SEND THE LINE FEED
 Ø15A  Ø51Ø    Ø643          LDA  *PDTDBA,BR            LOOK AT THE FORMAT WORD
 Ø15B  DB1F    Ø644          TABO LF2FRM                SECOND LINE FEED ?
 Ø15C  7D1B    Ø645          BRU  *TEMP1,BR             NO, RETURN TO CALLER
 Ø15D  Ø7ØA    Ø646          LDA  =LF                   YES
 Ø15E  7Ø92    Ø647          BRL  XMTCHR                YES, SEND IT OUT
 Ø15F  7D1B    Ø648          BRU  *TEMP1,BR             THEN RETURN
         Ø16Ø  Ø649  FMTFF   EQU  $
 Ø16Ø  Ø7ØC    Ø65Ø          LDA  =FF
 Ø161  7Ø8F    Ø651          BRL  XMTCHR                SEND THE FORM FEED
 Ø162  7D1B    Ø652          BRU  *TEMP1,BR
               Ø653  *
               Ø654  *
         Ø163  Ø655  RDSTAT  EQU  $
 Ø163  C7C7    Ø656          IOCOM                      READ INTERFACE STATUS WORD
 Ø164  3ØØØ    Ø657          IOBUS READ,DATA,A
 Ø165  38ØØ    Ø658          @AND =>8Ø1F
 Ø166  8Ø1F
 Ø167  C7C7    Ø659          IOCOM                      RETURN IN PRB
 Ø168  28Ø2    Ø66Ø          REG  STORE,A,PRB,PRBDRL
         Ø169  Ø661  EXEDRX  EQU  $
 Ø169  C7C7    Ø662          IOCOM
 Ø16A  48ØØ    Ø663          EXIT EOR,Ø
               Ø664  *
               Ø665          HED  DATA TERMINAL DSR - CHARACTER INTERRUPT INPUT
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - CHARACTER INTERRUPT INPUT                    SHEET  0020

                  0666           PEJ
           016B   0667  TAKCHR   EQU  $
    016B   6F20   0668           CPA  =>20
    016C   CD40   0669           SGT
    016D   7C4D   0670           BRU  *XTAKCH
    016E   C502   0671           RMO  A,X
    016F   1201   0672           LDX  $+2,X
    0170   C527   0673           RMO  X,P
P   0171   00A2   0674  XEXTNR   DATA EXTNRM         00  NULL
P   0172   00A2   0675           DATA EXTNRM         01
P   0173   00A2   0676           DATA EXTNRM         02  HOME CURSOR
P   0174   00A2   0677           DATA EXTNRM         03
P   0175   00A2   0678           DATA EXTNRM         04
P   0176   00A2   0679           DATA EXTNRM         05
P   0177   00A2   0680           DATA EXTNRM         06
P   0178   00A2   0681           DATA EXTNRM         07  BELL
P   0179   008C   0682           DATA BACKSP         08  BACKSPACE OR CURSOR LEFT
P   017A   0083   0683           DATA TAKCH1         09  HORIZONTAL TAB
P   017B   0196   0684           DATA DWNARR         0A  LINE FEED OR CURSOR DOWN
P   017C   00A2   0685           DATA EXTNRM         0B
P   017D   00A2   0686           DATA EXTNRM         0C  FORM FEED
P   017E   00AA   0687           DATA GOTOCR         0D  CARRIAGE RETURN
P   017F   00E0   0688           DATA LINFED         0E  LIST INPUT LINE
P   0180   00A2   0689           DATA EXTNRM         0F
P   0181   00A2   0690           DATA EXTNRM         10  DLE
P   0182   00A2   0691           DATA EXTNRM         11
P   0183   00A2   0692           DATA EXTNRM         12  PRINT
P   0184   00A2   0693           DATA EXTNRM         13
P   0185   00A2   0694           DATA EXTNRM         14  DC4
P   0186   00A2   0695           DATA EXTNRM         15
P   0187   00A2   0696           DATA EXTNRM         16
P   0188   00A2   0697           DATA EXTNRM         17  CLEAR END OF LINE
P   0189   00A2   0698           DATA EXTNRM         18
P   018A   00A2   0699           DATA EXTNRM         19
P   018B   01AB   0700           DATA UPARRO         1A  CURSOR UP
P   018C   00A2   0701           DATA EXTNRM         1B
P   018D   0191   0702           DATA RIGARR         1C  CURSOR RIGHT
P   018E   00A2   0703           DATA EXTNRM         1D
P   018F   00A2   0704           DATA EXTNRM         1E
P   0190   007F   0705           DATA CLRSCR         1F  CLEAR SCREEN
           0191   0706  RIGARR   EQU  $
    0191   C502   0707           RMO  A,X
    0192   C7C7   0708           IOCOM
    0193   2002   0709           REG  LOAD,A,PRB,PRBDRL
    0194   C300   0710           RIN  A,A
    0195   7808   0711           BRU  DWNAR1
           0196   0712  DWNARR   EQU  $
    0196   C502   0713           RMO  A,X
    0197   C7C7   0714           IOCOM
    0198   2102   0715           REG  LOAD,F,PRB,PRBDRL
    0199   011B   0716           LDA  PDTDEV,BR
    019A   3B00   0717           @AND =>FF           MASK WIDTH
    019B   00FF
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - CHARACTER INTERRUPT INPUT                    SHEET  0021

  019C  2F02  0718         SUB  =>2
  019D  C090  0719         RAD  E,A
        019E  0720  DWNAR1 EQU  $
  019E  C7C7  0721         IOCOM             CRT?
  019F  1BB4  0722         BIT  SKIP1,PDT,PDTDEV,PDTFFH
  01A0  7CD0  0723         BRU  *XEXTNR
  01A1  C7C7  0724         IOCOM
  01A2  2183  0725         REG  LOAD,E,LDT,LDTOPC
  01A3  C410  0726         RCA  E,A
  01A4  CD40  0727         SGT
  01A5  7CCB  0728         BRU  *XEXTNR
        01A6  0729  DWNAR2 EQU  $
  01A6  C7C7  0730         IOCOM
  01A7  2B02  0731         REG  STORE,A,PRB,PRBDRL
  01A8  C520  0732         RMO  X,A
  01A9  7C00  0733         *BRU SENDCH
P 01AA  00A1
        01AB  0734  UPARRO EQU  $
  01AB  C7C7  0735         IOCOM             CRT?
  01AC  1BB4  0736         BIT  SKIP1,PDT,PDTDEV,PDTFFH
  01AD  7CC3  0737         BRU  *XEXTNR      NO
  01AE  C7C7  0738         IOCOM
  01AF  2102  0739         REG  LOAD,E,PRB,PRBDRL
  01B0  011B  0740         LDA  PDTDEV,BR
  01B1  3B00  0741         *AND =>FF         MASK WIDTH
  01B2  00FF
  01B3  2F02  0742         SUB  =>2
  01B4  C410  0743         RCA  E,A
  01B5  C080  0744         SGE
  01B6  7CBA  0745         BRU  *XEXTNR
  01B7  C001  0746         RSU  A,E
  01B8  C510  0747         RMO  E,A
  01B9  171A  0748         LDX  =CURUP
  01BA  7BEB  0749         BRU  DWNAR2
P 01BB  0083  0750  XTAKCH DATA TAKCH1
              0751         HED  DATA TERMINAL DSR - SPECIAL CHARACTER LISTS
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - SPECIAL CHARACTER LISTS                     SHEET  0022

                    0752          PEJ
                    0753   ************************************************************
                    0754   *
                    0755   * LISTS USED BY THE XLIST SUBROUTINE.  TRANSMISSION
                    0756   * IS TERMINATED BY BIT 1 OF THE WORD FOLLOWING THE LAST
                    0757   * CHARACTER TO BE SENT BEING SET.
                    0758   *
                    0759   ************************************************************
                    0760   *
          01BC      0761   XBCKAR EQU  $
01BC      005F      0762          DATA BCKARR
          01BD      0763   XCRLF  EQU  $
01BD      000D      0764          DATA CR
01BE      0000      0765          DATA NULL
01BF      0000      0766          DATA NULL
01C0      0000      0767          DATA NULL
01C1      0000      0768          DATA NULL
01C2      0014      0769          DATA DC4
          01C3      0770   LFLST  EQU  $
01C3      000A      0771          DATA LF
01C4      0017      0772         DATA CLRLIN
01C5      4000      0773          DATA >4000
          01C6      0774   FFLST  EQU  $
01C6      000A      0775          DATA LF
01C7      000A      0776          DATA LF
01C8      000A      0777          DATA LF
          01C9      0778   LF3LST EQU  $
01C9      000A      0779          DATA LF
01CA      0017      0780          DATA CLRLIN
01CB      000A      0781          DATA LF
01CC      0017      0782          DATA CLRLIN
01CD      000A      0783          DATA LF
01CE      0017      0784          DATA CLRLIN
01CF      4000      0785          DATA >4000
          01D0      0786   CR1LST EQU  $
01D0      000D      0787          DATA CR
01D1      0000      0788          DATA NULL
01D2      0000      0789          DATA NULL
01D3      0000      0790          DATA NULL
01D4      0000      0791          DATA NULL
01D5      0000      0792          DATA NULL
01D6      4000      0793          DATA >4000
          01D7      0794   CR2LST EQU  $
01D7      000D      0795          DATA CR
01D8      0000      0796          DATA NULL
01D9      4000      0797          DATA >4000
          01DA      0798   HMLST  EQU  $
01DA      0002      0799          DATA HOME
01DB      0017      0800          DATA CLRLIN
01DC      4000      0801          DATA >4000
                    0802          HED  DATA TERMINAL DSR - AUTOFLOW SOURCE
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAP R2LC
DATA TERMINAL DSR - AUTOFLOW SOURCE                                    SHEET  0023

               0803            PEJ
        0000   0804            END   ISDT
```

Table D-6. Data Terminal DSR Listing (Continued)

```
SAF R2LC
DATA TERMINAL DSR - AUTOFLOW SOURCE                                  SHEET 0024

   A        0002        ABORT    0021        B        0006        BACKS1   00CF
   BACKSP   00BC      R BAKSP    0008        BCKARR   005F        BELL     0007
   BTT      850B        BKSLSH   005C        BR       0001        CHAR     8400
   CHAROK   0124        CHGTBL   0060        CHKFOF   00AF        CHKLFB   0097
   CKBUFF   0093        CLEAR    0000        CLOSE    002B      R CLR      001F
   CLRLIN   0017        CLRSCR   007F        CR       000D        CR1LST   01D0
   CR2LST   0107        CRFORM   000D        CURUP    001A        DATA     0000
   DC4      0014      R DLE      0010        DOINP    006C        DONTRG   0073
   DWNAR1   019F        DWNAR2   01A6        DWNARR   0196        E        0001
   ENDFIL   AFAA        FOR      0024        EXEORY   0169        EXIT     8100
   EXTFOF   0055        EXTNRM   00A2        FF       000C        FFFORM   000F
   FFLST    01C6        FMTFF    0160        FMTLF    015B        FORMAT   014B
   FULLBF   00A4        GET      000B        GOTOCR   00AA        HMLST    01DA
   HOME     0002        ICCHAR   0001        IGNOR    002B        ILLOPD   0009
   IOBUS    840B        IOCOM    C7C7        IOENT    0007        ISDSRT   0000
   ISOT     0000        ISINT    0031        ISRST    004B      R IVRDT    00A2
 R IVWDT    00A2        L        0005        LDT      0002        LDTOPC   0003
   LF       000A        LF2FRM   000F        LF3LST   01C9        LFFORM   000E
   LFLST    01C3        LINFED   00E0        LOAD     0004        LSTLIN   0117
   LSTLUP   010E        M        0003        NOCR     0152        NORM     0020
   NULL     0000        OPCDER   0026        OPEN     0051        OTWREP   005A
   P        0007        PDT      0001        PDT1NL   0005        PDTBCK   0006
 R PDTBSY   0000        PDTDBA   0010        PDTDEV   001B        PDTFFH   0004
   PDTFLG   0001        PDTLFA   0007        PDTLFB   000E        PDTLFE   0002
   PDTLFT   0003      R PDTMXC   0014        PDTOCT   0012        PDTOWR   000F
   PDTPRB   000D        PDTRVC   0016        PDTSTA   0015        PDTWVC   0017
   POSFRM   000C        PRB      0000        PRBATM   0002        PRBDBA   0003
   PRBDRL   0002        PRBEOF   0002        PRBFAC   0003        PRBOPC   0001
   PRBOPT   0003        PRBOWR   0001        PRBSBL   0003        PRBSCR   0004
   PRBSFL   0000        PRBUFL   0001        PUT      000A        RDASCI   0069
 R RDCON    003A        RDINT    0040        RDSTAT   0163        READ     0006
   REG      84A0        RIGARR   0191        RNGBEL   006E        RUBOT    007F
   RUBOUT   0067        SENDCH   00A1        SET      0001        SETRIG   0078
   SETWIG   007B        SKIP0    0002        SKIP1    0003        SKPLF    0114
   SNDCHR   00EB        SNDLIN   0144        STORE    0005        TAB      0089
   TAKCH1   0083        TAKCHR   016B        TEMP1    001B        TEMP2    0019
   TEMP3    001A        UPARRO   01AB        WRITE    0007        WRTCHR   00EC
   WRTEOF   002D        WRTINT   003C        WTASCI   013A        X        0002
   XBCKAR   01BC        XCR      0106        XCRLF    01BD        XEXNRM   007E
   XEXTNR   0171        XLF      0101        XLFLST   0139        XLIST    010B
   XMTCHR   00F1        XTAKCH   01BB        XXBKAR   005B        XXCRLF   0057
   XXLIST   0059

0000 ERRORS
```

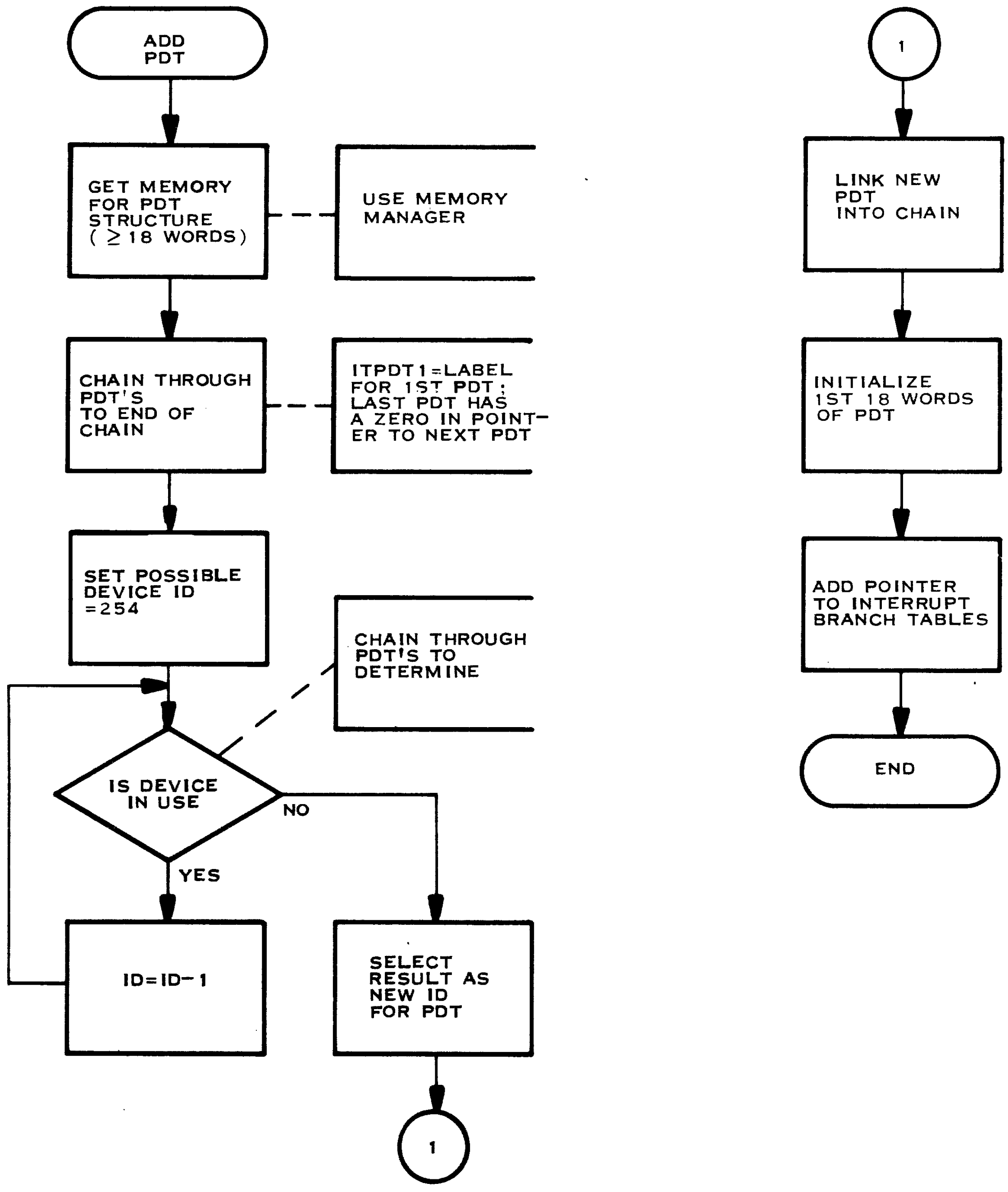
Table D-7. PDT Builder Required Definitions

Label	Definition
ITDBT0	Pointer to the internal IO expansion PDT branch table
ITDBT1	Pointer to the 1st IO expansion PDT branch table
ITDBT2	Pointer to the 2nd IO expansion PDT branch table
ITDBT3	Pointer to the 3rd IO expansion PDT branch table
ITDBT4	Pointer to the 4th IO expansion PDT branch table
ITDMT	Pointer to the DMAC expansion PDT branch table
ITVECT	Label indicating the start of the vector interrupt PDT branch table
ITPDT1	Label indicating the start of the 1st PDT in the system
SDMAEX	Label on word containing the DMAC expansion flag: 0 - no expander 1 - expander
SDBEXP	Label on word containing the number of I/O bus external expansions, value: 0-4
SINTEX	Label on word indicating that there is internal I/O expansion 0 - no internal expansion 1 - internal expansion
IDDMAC	Label indicating a word that returns control to the system when no processing is required on an interrupt. This label may be used in PDT words 12 and 17, or may be branched to when the processing required to clear an unsolicited interrupt is complete.
JDSCAA	Pointer to a system memory managed area. This pointer should be used by the PDT Builder program when a block of memory is required in the system.

The utility requires some memory allocated permantly for the PDT and any other structures required by the new device. This memory block should be at least 18 words (more if temporary storage is required). To allocate this memory space, the utility has access to one of the system memory manager routines. The following is the calling sequence for the memory allocation (MRAL) routine:

```
        REF MRAL
        REF JDSCAA
         •
         •
         •
```

(listing continued on next text page)


ADD PDT
GET MEMORY FOR PDT STRUCTURE (≥ 18 WORDS)
USE MEMORY MANAGER
CHAIN THROUGH PDT'S TO END OF CHAIN
ITPDT1=LABEL FOR 1ST PDT: LAST PDT HAS A ZERO IN POINTER TO NEXT PDT
SET POSSIBLE DEVICE ID =254
CHAIN THROUGH PDT'S TO DETERMINE
IS DEVICE IN USE
NO
YES
ID=ID-1
SELECT RESULT AS NEW ID FOR PDT
1
1
LINK NEW PDT INTO CHAIN
INITIALIZE 1ST 18 WORDS OF PDT
ADD POINTER TO INTERRUPT BRANCH TABLES
END

(A)130254

Figure D-10. Addition of New PDT

```
              @LDA JDSCAA
                STA ARG1+1
              @LDM =ARG1
              @BRL MRAL
                 .
                 .
                 .
      ARG1    DATA 3      # of arguments
              DATA $-$
              DATA BLKADR
              DATA BLKSIZ

    BLKSIZ    DATA $ - $        Size of block needed
    BLKADR    DATA 0            Pointer to requested block
```

Upon return from the MRAL routine, determine if the requested block was given to the PDT Builder utility. If the pointer to the requested block (BLKADR) is zero, then the memory was not available.

The PDT Builder utility has access to a label indicating the start of the first PDT in the PDT chain. To determine a suitable device I.D. number, select the largest, non-assigned device I.D. (<255). Chain down through each PDT to determine if the selected I.D. is assigned. The last PDT on the chain is identified by a 0 in the Next PDT Pointer field (WORD 0).

After assigning a device I.D., chain the PDT to the last PDT on the list and initialize the PDT according the description provided in figure D-2. Then place the PDT pointer in the appropriate PDT interrupt branch table as described earlier in this appendix.

Once the PDT builder program is coded, assemble it using SAPG. Object output should be to the file (USER01, ASMOUT). Then link it with the DX980 operating system using DXOLE. The following job control is required to do this using the standard LINK procedure. Refer to Section VIII for a detailed description of the DXOLE utility. The following is a sample Link:

```
    //RUN DXOLEP  DOB1=DISC1  DIN=SC;
    .. FLM = (SYSTEM, UTLFIL) RLM LLM=(3, 0, 32, 3)
```

The input would then be as follows:

```
    ƀSUBSYSTEM OVLY
    ƀROOT MAIN.
    ƀSEGMENT 1
    ƀINCLUDE 1                  object for PDT utility
    /*
```

D.3 ADDING THE NEW DEVICE SERVICE ROUTINE

Every DX980 has a standard utility, LMUPDT, that updates a memory image phase (MIP) of a load module generated by the DXOLE utility. LMUPDT can replace any of four dummy memory image phases (MIP Nos. 181, 182, 183 and 184) in the DX980 system load module.

Perform the following procedure to update the DX980 load module:

1. Reference the description of the load module update (LMUPDT) utility in Section VIII of this manual.

2. Perform this update procedure using the <userid> of SYSTEM.

3. Run the standard LMUPDT. Set LUN 5 to the record input device. This record should be the MIP number of the system load module MIP to be updated. Set LUN 7 to the relative record file of the DX980 system load module, (SYSTEM, SYSLD). The input is as follows:

   ```
   //RUN LMUPDT DCON=SC FLM=(SYSTEM, UTLFIL);
     FUPD=(SYSTEM,SYSLD)
   ```

4. Once the phase has been replaced, reload the system via the IPL program.

This utility does not replace an existing module. Instead, it adds a module to the end of the File and changes pointers from the old MIP to indicate the new MIP. Therefore, after replacing the MIP several times in the process of testing and debugging, LMUPDT may terminate with an error indication such as "file full". To prevent this problem, build a development system load file using the load module copy program (DXCOPY). The new file should be large enough to allow for expansion. Once the MIP has been tested, this file may be deleted. Then run the update utility against the system file or disc.

ALPHABETICAL INDEX

ALPHABETICAL INDEX

INTRODUCTION

The following index lists key words and concepts from the subject material of the manual together with the area(s) in the manual that supply major coverage of the listed concept. The Reference column of the listing contains references to the following manual areas:

- Sections - References to Sections of the manual appear as "Section x" with the symbol x representing any numeric quantity.
- Appendixes - References to Appendixes of the manual appear as "Appendix y" with the symbol y representing any capital letter.
- Paragraphs - References to paragraphs of the manual appear as a series of alphanumeric or numeric characters punctuated with decimal points. Only the first character of the string may be a letter; all subsequent characters are numbers. The first character refers to the section or appendix of the manual in which the paragraph is found.
- Tables - References to tables in the manual are represented by the capital letter T followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the table). The second character is followed by a dash (-) and a number:

  Tx-yy
- Figures - References to figures in the manual are represented by the capital letter F followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the figure). The second character is followed by a dash (-) and a number:

  Fx-yy
- Other entries in the Index - References to other entries in the index are preceded by the word "See" followed by the referenced entry.

ALPHABETICAL INDEX

ALPHABETICAL INDEX (Continued)

ALPHABETICAL INDEX (Continued)

ALPHABETICAL INDEX (Continued)

ALPHABETICAL INDEX (Continued)

ALPHABETICAL INDEX (Continued)

ALPHABETICAL INDEX (Continued)

USER'S RESPONSE SHEET

Manual Title: DX980 General Purpose Operating System Programmer's Guide (943005-9701)

Manual Date: 1 August 1975 Date of This Letter: ______

User's Name: ______ Telephone: ______

Company: ______ Office/Department: ______

Street Address: ______

City/State/Zip Code: ______

Please list any discrepancy found in this manual by page, paragraph, figure, or table number in the following space. If there are any other suggestions that you wish to make, feel free to include them. Thank you.

Location in Manual	Comment/Suggestion

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
FOLD ON TWO LINES (LOCATED ON REVERSE SIDE), STAPLE AND MAIL

CUT ALONG LINE

First Class
PERMIT NO. 3135
Austin, Texas

BUSINESS REPLY MAIL
No Postage Necessary if Mailed in the United States

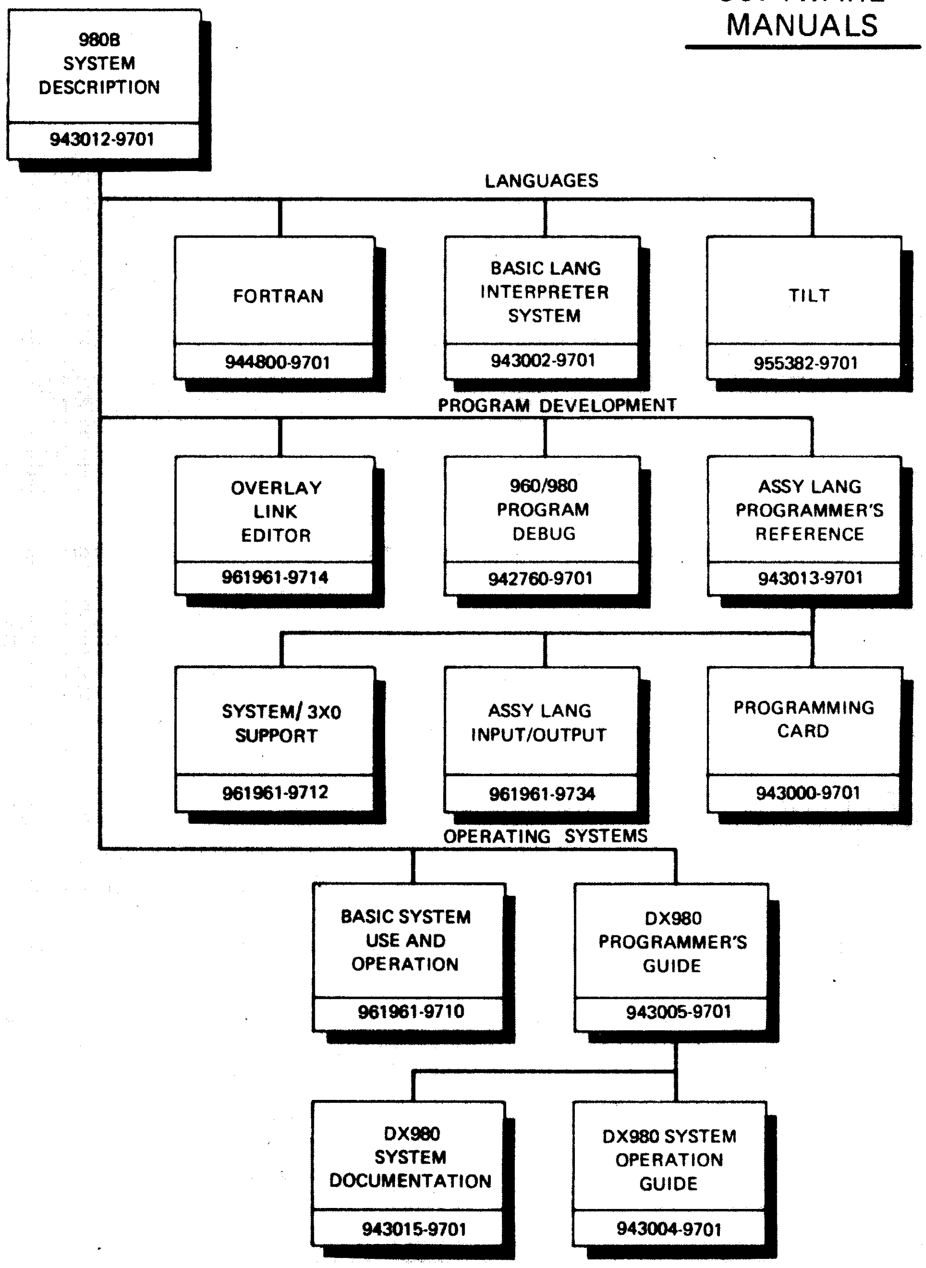
Postage Will Be Paid by

TEXAS INSTRUMENTS INCORPORATED
DIGITAL SYSTEMS DIVISION

P.O. BOX 2909 · AUSTIN, TEXAS 78767

Attn: TECHNICAL PUBLICATIONS, MS 2146

980 COMPUTER SYSTEM SOFTWARE MANUALS



TEXAS INSTRUMENTS
INCORPORATED
DIGITAL SYSTEMS DIVISION
POST OFFICE BOX 2909 • AUSTIN, TEXAS 78767