SPERRY RAND

UNIVAC 1100 SERIES

OPERATING SYSTEM

PROGRAMMER
REFERENCE

UP-4144 Rev. 2

This document contains the latest information available at the time of publication. However, the Univac Division reserves the right to modify or revise its contents. To ensure that you have the most recent information, contact your local Univac Representative.

PRINTED IN U.S.A.

CONTENTS

APPENDIXES

FIGURES

TABLES

1. INTRODUCTION

1.1. SCOPE OF MANUAL

The UNIVAC 1100 Series Operating System comprises the Univac-supplied software for the UNIVAC 1100 Series Computer Systems. This manual discusses the base portion of the operating system, that is, the executive system (EXEC 8) and the associated software needed to construct, execute, and maintain user programs. Information on language processors such as COBOL, FORTRAN, and the assembler, and on applications software such as SORT/MERGE, APT, and PERT can be found in their respective manuals.

Information that is primarily of interest only to an operator, installation manager, or systems analyst is covered only briefly if at all (for example, operating procedures, system generation procedures, internal system logic, and so forth). Such material is covered in other Univac publications.

The purpose of this manual is to provide information for the user programmer so that he can make full use of the wide range of capabilities provided by the UNIVAC 1100 Series Hardware/Software Systems. Any differences between the operating system described in this manual and the latest released software are described in the *Univac 1100 Series System Memorandum.*

A knowledge of the 1100 series hardware architecture and machine (assembler) language programming is assumed. This knowledge is helpful, but not mandatory for the user of a higher level language or applications package.

1.2. THE OPERATING SYSTEM

The UNIVAC 1100 Series Operating System was designed to meet the total computing requirements of today's users, and to allow for the change and growth required for the future. The operating system is the outgrowth of Univac's many years of experience in multiprogramming, multiprocessing, time sharing, communications, and real time oriented systems, and provides a system that contains the facilities required in complex environments, yet it is easy to operate and use.

A complete set of software, ranging from high level language compilers to basic service functions, is included in the operating system. The six major categories are:

- Executive System
- System Processors
- Utility System Processors
- Language Processors
- Subroutine Library
- Applications Programs

The first three categories are discussed in detail in this manual. The execution environment is specified for the software falling into the last three categories. In addition to the standard operating system, this manual describes certain utility routines provided only for the convenience of the user and they are not supported as operational software. These unsupported utility routines are:

Document Processor (DOC) – see Section 21

Flow Charting Processor (FLUSH) – see Section 21

Log File Editor (LOGFED) – see Section 23

Billing Routine (BILLER) – see Section 23

Element Listing Routine (LIST) – see Section 18

Cross-reference Processor (CULL) – see Section 18

CUR-to-FUR Conversion (CON78) – see Section 18

Program Trace Routine – see Section 11

1.3. THE EXECUTIVE SYSTEM

To take full advantage of the speed and hardware capabilities of the 1100 series and to make effective use of a given hardware configuration, a comprehensive internal operating environment has been created.

This environment permits the concurrent operation of many programs; it allows the system to react immediately to the inquiries, requests, and demands of many different users at local and remote stations; it accords with the stringent demands of real time applications; it can store, file, retrieve, and protect large blocks of data; and it makes the best use of all available hardware facilities, while minimizing job turnaround time.

Only through central control of all activities of the system can this environment of the combined hardware and software systems be fully established and maintained to satisfy the requirements of all applications. The responsibility for efficient, flexible, centralized control is borne by the executive system, which controls and coordinates the functions of the complex internal environment. By presenting a relatively simple interface to the programmer it allows him to use the system easily, while relieving him of concern for the internal interaction between his program and other coexistent programs.

1.3.1. MULTIPLE MODES OF OPERATION

The technical capabilities of the executive system cover a great variety of data processing activities. Its design is such that no penalties are imposed upon any one of these activities by the support provided for the others, and an installation not interested in making use of the full range of capabilities may specify component features to be eliminated at system generation time.

1.3.1.1. BATCH PROCESSING

Foremost among the capabilities of the executive system is the support provided for batch processing. The system is designed to ease run preparation and submission, to shorten job turn-around time, and to reduce the need for operator intervention and decisions. Batch jobs may be processed from a variety of remote terminals, as well as from central site equipment.

1.3.1.2. DEMAND PROCESSING (TIME-SHARING)

Complementing the batch processing capabilities of the executive system are its time-sharing capabilities, the simultaneous accommodation by the executive system of requests and demands from users at numerous remote inquiry terminals, operating in a demand (or conversational) mode. All facilities available to the batch processing user are also available in the demand mode, the primary difference being that the executive system permits the user additional flexibility in the statement and control of individual runs; when an error is made, the user simply corrects it online and proceeds rather than suffering the turn-around cycle inherent in batch processing. The demand user may communicate directly with either the executive or a user program or he may communicate with a conversational processor, such as Conversational FORTRAN.

1.3.1.3. REAL TIME PROCESSING

The executive system is also designed to be applicable to programs which have real time requirements. The UNIVAC Communications Subsystem, together with efficient scheduling and interrupt processing features of the executive system, provide an environment satisfactory for any real time program.

1.3.1.4. MULTIPROGRAMMING AND MULTIPROCESSING

Runs may come from many sources, remote and central. These various runs, through the executive system's use and control of efficient multiprogramming and multiprocessing techniques may, at any given moment, be in different stages of activity; input, processing, and output may all be occurring simultaneously, thus ensuring efficient operation.

1.3.2. UTILIZATION OF MASS STORAGE

The executive system is designed to ensure effective and efficient utilization of the mass storage devices. The consequence is an unprecedented ability to relieve operators and programmers of the responsibility of maintaining and handling cards and magnetic tapes, thus eliminating many of the errors which heretofore have accompanied the use of large scale software systems. At the same time, the overall operating efficiency is considerably improved.

Permanent data files and program files are maintained on the mass storage devices, with full facilities for modification and manipulation of these files. Security measures are established by the executive system to ensure that files are not subject to unauthorized use. Provisions are also made within the executive system for automatic relocation of infrequently used files to magnetic tape, as unused mass storage space approaches exhaustion. When the use of files relocated in such a manner is requested, they are retrieved and restored under control of the executive system with no inconvenience to the user.

1.3.3. FUNCTIONAL AREAS OF THE EXECUTIVE SYSTEM

The executive system is composed of many different routines which perform many different functions. These functions and routines are summarized in the following paragraphs.

1.3.3.1. EXECUTIVE CONTROL LANGUAGE

In the executive system, the user has a simple means of directing the execution of the individual tasks of a run and of relaying operational information concerning the run to the executive. This is accomplished through a set of control statements capable of performing all of the functions desirable or necessary in a modern executive system. The control language is open ended and easily expanded, so that features and functions may be added as the need arises.

The basic format of a control statement is quite simple, and is adaptable to a large number of input devices. Statements are not restricted to punched cards and may be of variable lengths. Each control statement consists of a heading character (@), for recognition purposes, followed by a command and a variable number of parameters. The end of a control statement is indicated by the end of a card, a carriage return, or an equivalent signal, depending on the type of input device.

1.3.3.2. THE SUPERVISOR

The supervisor is the executive system component that controls the sequencing, setup, and execution of all runs. It is designed to control the execution of a large number of independent and interdependent programs.

The supervisor contains three levels of scheduling: coarse scheduling, dynamic allocation of storage space, and central processor unit (CPU) dispatching. Runs entering the system are sorted into information files and these files are used by the supervisor for run scheduling and processing. Control statements for each run are retrieved and scanned by a control statement interpreter in the supervisor to facilitate the selection of runs for setup by the coarse scheduler. The coarse scheduling of each run primarily depends on two factors: the priority of the run, and its facility requirements.

The dynamic allocator takes runs set up by the coarse scheduler and allots storage space according to the needs of the individual tasks (programs) of a run. Normally, tasks from many different runs are located in storage at the same time. Each run may be thought of as being made up of tasks, where a task is a single operation of a system processor or the execution of a user program. All tasks for a given run are processed serially but not necessarily consecutively; if there are several runs, the tasks of separate runs are interleaved.

When time-sharing of storage is appropriate, the dynamic allocator initiates storage swaps. This involves writing one program on mass storage and replacing it temporarily in main storage with another program. Such action is taken only to provide reasonable response time to remote demand-processing terminals, or to satisfy batch priority requirements.

The CPU dispatching routine is a third level of scheduling; it selects among the various tasks currently occupying main storage whenever it is appropriate to switch the commitment of the CPU from one task to another. Under normal circumstances, a batch program is allowed to use a CPU either until it becomes interlocked against some event or until some higher priority program is freed of all of its interlocks. On multiprocessor systems, two or more tasks will be in actual execution at the same time.

1.3.3.3. FACILITIES ASSIGNMENT

Available facilities and their disposition are indicated to the system at system generation time; thereafter, the executive system assigns these facilities, as needed and as available, to fulfill the facilities requirements of all runs entering the system. The executive system maintains current inventory tables that indicate what facilities are available for assignment, and which runs are using the currently unavailable facilities.

1.3.3.4. FILE CONTROL

The executive file control routines afford the highest degree of operational flexibility in storing and retrieving data, without concern for the physical characteristics of the recording devices. Thus, most files are made insensitive to input/output (I/O) media characteristics, as the system adjusts the interface between the file and the device. Security measures ensure that files are not subject to unauthorized use or destruction. File control routines are provided to roll out files from mass storage devices to magnetic tape, as well as reconstruct such files on the mass storage devices when the user calls for them.

Comprehensive utility routines are available for manipulation of files and for informing the user of current status and structure of his files. Provisions are made for random storage and retrieval of data, under the direction of the user. User program files and data files are maintained and processed in the same environment.

1.3.3.5. OPERATOR COMMUNICATIONS

Operator functions are required for a large variety of activities. The executive system groups them into four classes, thus equally dividing operator duties in a multioperator installation. These functions may be associated with as many as three system consoles or as few as one, depending on the complexity and layout of the installation.

The executive system displays information such as current system load and operator requests associated with I/O setup and I/O interlocks. The operator can request other information, such as backlog status. If the display area becomes filled up, the executive defers lower priority displays.

Since this manual is for the user programmer as opposed to the computer operator, it does not contain detailed information concerning the operator communication functions.

1.3.3.6. INPUT/OUTPUT DEVICE HANDLERS AND SYMBIONTS

The input/output device handlers and symbionts control the activities of all I/O channels and peripheral equipment attached to the system.

The following is a list of the onsite and remote peripheral hardware that is supported by the executive:

- Mass Storage Devices
  - FH-432, FH-880, and FH-1782 Magnetic Drum Subsystems
  - Unitized Channel Storage
  - 8414 Disc Subsystem
  - FASTRAND II and III Magnetic Drum Subsystems
- Magnetic Tape Devices
  - UNISERVO 12 and 16 Magnetic Tape Subsystems
  - UNISERVO IV–C, VI–C, and VIII–C Magnetic Tape Subsystems
  - UNISERVO II–A and III–A Magnetic Tape Subsystems
- Printer Subsystems
  - Type 0751, 0755, 0758, and 0768 High Speed Printers
  - UNIVAC 1004 Printer
- Card Subsystems
  - Type 0706 and 0711 Card Readers
  - Type 0600 and 0603 Card Punches
  - UNIVAC 1004 Card Reader/Punch
- Remote Devices
  - Onsite Interface Hardware

    (1) Communications Terminal Module Controller Subsystem (CTMC)
    (2) Communications Terminal Synchronous Subsystem (CTS)
    (3) Word Terminal Synchronous Subsystem (WTS)
  - Remote Terminal Hardware

    (1) UNISCOPE 100 Display Terminal
    (2) UNISCOPE 300 Visual Communications Terminal
    (3) DCT 2000, 1000, and 500 Data Communications Terminals
    (4) UNIVAC 9300/9300–II Remote System
    (5) UNIVAC 9200/9200–II Remote System
    (6) Teletypewriter Models 33 and 35
    (7) UNIVAC 1004 Card Processor
    (8) Friden 7100 Typewriter

1.4. SYSTEM PROCESSORS

The system processors of the operating system are programs which provide for the utilitarian functions required to construct and modify programs, maintain and modify files, and provide diagnostic information upon program termination.

1.4.1. COLLECTOR

The collector is designed to provide the user with the means of collecting and linking relocatable subprograms to produce an absolute program in the form ready for execution under control of the executive system.

1.4.2. FILE UTILITY PROCESSOR (FURPUR)

FURPUR consists of a set of file maintenance routines which provide the flexibility in management and manipulation of catalogued or temporary files containing data or programs.

1.4.3. POSTMORTEM DUMP PROCESSOR (PMD)

The postmortem dump processor (PMD) produces edited dumps of the contents of main storage at program termination; dumps produced dynamically during execution are automatically printed. Individual program parts are identified with the assistance of diagnostic tables produced with the absolute program by the collector.

1.4.4. DATA AND ELT PROCESSORS

The DATA and ELT processors are used to create and manipulate data streams and program elements.

1.4.5. FILE ADMINISTRATION PROCESSOR (SECURE)

The SECURE processor uses a source language structure which allows the user to define specific tasks with simple COBOL-like statements. The processor's primary functions are to produce backup tapes for catalogued files, and to provide a recovery mechanism for these files in case of system failure.

1.4.6. TEXT EDITOR (ED)

The ED processor is a text editor which enables a user to modify or move character strings in either program files or data files.

1.4.7. PROCEDURE DEFINITION PROCESSOR (PDP)

The procedure definition processor (PDP) accepts source language statements defining assembler, COBOL, or FORTRAN procedures and builds an element in the user-defined program file. These procedures may subsequently be referenced in an assembly or compilation without definition.

1.5. SYSTEM UTILITY PROCESSORS

The system utility processors provide features which are commonly required and used. Unlike the system processors, the features provided are not necessary for the effective utilization of the operating system.

1.5.1. CUR–TO–FUR CONVERSION (CON78)

This processor converts magnetic tapes created by the UNIVAC EXEC II complex utility routine (CUR) to magnetic tapes acceptable as input to UNIVAC 1100 series program files. The processor will accept UNIVAC EXEC II symbolic elements, COBOL library elements, and procedure elements, and converts them to the proper formats.

1.5.2. FLUSH

FLUSH (Flowcharting Language for User's Simplified Handling) is a processor which accepts assembler format input to produce a flowchart. FLUSH can:

- be instructed through the use of parameters, called options, which are contained in the comments field of the instructions to be flowcharted, or

- perform an analysis based on the assembler instruction statements.

1.5.3. SSG PROCESSOR

The SSG processor is a general-purpose symbolic stream generator. Any variety of symbolic streams, varying from a file of data to a run stream which configures an executive system, may be generated. Directions and models for building of the desired stream images are conveyed to SSG through a skeleton which is written in SYMSTREAM, an extensive manipulative language.

1.5.4. CULL PROCESSOR

CULL is a processor which produces an alphabetically-sorted, cross-referenced listing of all symbols in a specified set of symbolic elements. Provisions are included, via options, to selectively include or exclude defined symbols or symbol groups from the output.

1.5.5. DOCUMENT PROCESSOR (DOC)

DOC accepts the contents of a document and composes that document according to the user's specifications. Control statements provide listing and text control, including pagination, justification, indentation, and hyphenation. Document maintenance is provided on a line-image basis and by content addressing of text character strings.

1.5.6. LIST PROCESSOR

This special-purpose processor provides edited element listings which include associated element control information not normally of interest to the user. It is intended for debugging of software which deals with program files.

1.6. LANGUAGE PROCESSORS

The operating system provides several language processors, such as FORTRAN, COBOL, ALGOL, and the assembler. Certain of these processors are specifically designed for demand mode operation. Consult the appropriate manual for information on using a particular language.

1.7. RELOCATABLE SUBROUTINE LIBRARY

An extensive library of relocatable subroutines is provided. Subroutines referenced by user programs are automatically included when the absolute program is constructed by the collector. The library elements included in the operating system fall into the following general categories:

- Subroutines that support higher level languages (COBOL Library, FORTRAN Library, and so forth)

- Processor Interface Routines

- SORT/MERGE

- Diagnostic Subroutines

- Item Handler and Block Buffering Package, which provide intermediate-level record and block I/O control

- Service Routines, for editing, conversion, segment loading, and so forth

- MATH–PACK/STAT–PACK – Mathematical and Statistical Functions

- Assembler Procedure Library – Provides macro capability for generation of common machine-level coding and parameter sequences.

This manual describes only those subroutines that fall into the base portion of the operating system, such as the diagnostic subroutines, the item handler and block buffering package, the editing routines, and certain assembler procedures. Please consult the appropriate manual for information on a subject not covered in this document.

1.8. APPLICATIONS PROGRAMS

The operating system provides many applications programs such as APT, GPSS, PERT, and so forth. Please consult the appropriate manual for information on a particular applications program.

2. GENERAL CONCEPTS AND DEFINITIONS

2.1. INTRODUCTION

This section presents certain basic information that is essential for comprehension of the remaining sections of the manual which deal with specific areas of the operating system.

2.2. DEFINITIONS AND ABBREVIATIONS

The following paragraphs (2.2.1 through 2.2.8) define terms that aid in comprehending the remainder of the manual. The reader is encouraged to become familar with them before proceeding.

2.2.1. INTRODUCTORY DEFINITIONS

Sytem

The total UNIVAC 1100 series hardware/software complex comprising an integrated information processing installation.

Operating System

The UNIVAC 1100 Series Operating System. The entire set of system software available for the UNIVAC 1100 series which is either a part of or operates under the executive system. This includes the executive system proper, compilers, utility programs, subroutine libraries, and so forth.

Executive

1100 Series Executive System, (EXEC 8). An executive is a routine that controls the execution of other routines. The executive is the principal interface between the user and the system as a whole. It is responsible for such functions as time and space allocation of system resources; first-level I/O control and interrupt answering; logging of system accounting data; first-level debugging assistance; and protection against undesired interaction of users with other users or the system.

User

An individual or organization that consumes services provided by the system.

2.2.2. HARDWARE DEFINITIONS

Application

The total hardware configuration, or a subset, resulting from partitioning that configuration by using either the availability control unit (ACU) or software downing of components.

Facilities

The peripheral units associated with an application; for example, tape units, mass storage, printers, and so forth.

CPU

Central Processing Unit. A unit of the system containing circuitry and operating registers which control the interpretation and execution of instructions. A CPU does not contain any main or auxiliary storage. Under the executive, multiple CPUs may access common main and auxiliary storage.

Multiprocessor
An application having two or more CPUs. Commonly abbreviated as MP.

Unit Processor
An application having a single CPU. Commonly abbreviated as UP.

Control Registers
Those operating registers of a CPU which can be utilized directly by a program; that is, the X, A, and R registers (except X0 and R0).

P Register
CPU operating register whose contents reflect the instruction currently being executed.

PSR
Processor State Register. A privileged register which controls the absolute main storage location of a program's I and D banks and specifies modes of operation of the CPU for the program. The PSR contains two basing fields which in conjunction with a program relative address determine an absolute main storage location within a 262K range.

SLR
Storage Limits Register. A priveleged register which provides program isolation in a multiprogramming environment. The executive loads the SLR with the programs I and D bank limits such that if a program attempts to access an address outside the program area, a guard mode fault interrupt is generated.

Main Storage
The general puspose, high speed core storage of the system, directly accessible by CPU operating registers, and serving principally to contain executing programs. As opposed to auxiliary storage.

Mass Storage
Supplemental storage which has random access capability; as opposed to magnetic tape, for example. Any type of flying-head magnetic drum, FASTRAND drum, disc, or unitized channel storage.

Core Storage
Synonymous with main storage.

Unitized Channel Storage
Core storage which is treated as and accessed by peripheral I/O hardware.

Word
A sequence of bits or characters treated as a unit and capable of being stored in a single main storage location (A word is represented by 36 bits for the 1100 series).

Granule
The incremental unit of size in which a storage medium can be allocated.

Word Addressable Drum Mass Storage
Mass storage which is capable of being accessed in units of single words. This is generally restricted to hardware having this capability (that is, flying head magnetic drum or unitized channel storage) but in some cases may be simulated by the executive. As opposed to FASTRAND mass storage.

FASTRAND-formatted Mass Storage
Mass storage which is accessible in units of 28 words (one sector). This may be on actual FASTRAND hardware, or may be simulated (by the executive) on any other mass storage device. The term FASTRAND in this manual refers to the format, not the hardware device, unless otherwise stated. This is the most common mass storage format. As opposed to word addressable drum mass storage.

Track
In the context of FASTRAND-formatted mass storage, a granule consisting of 64 sectors, each sector consisting of 28 words giving a total of 1792 words per track.

Position
A granule of 64 contiguous tracks, in the context of FASTRANT-formatted mass storage.

Communication Device
An input or output device which operates in a real time mode. CPUs must be prepared to receive input at any time or information may be lost.

Central Site
The CPU(s), main storage, and attached on-site peripheral equipment in a particular application.

Remote Site

Data terminal equipment that is time, space, or electrically distant from a central site, and capable of information exchange with the central site through some common carrier or transmission scheme, typically as a communication device.

CRT

Cathode ray tube (CRT). Used to denote any of several supported remote terminals which incorporate a CRT as the output display device as opposed to a typewriter.

2.2.3. PROGRAM ORGANIZATION DEFINITIONS

Program

Generally, a series of instructions, in a form acceptable to a computer, prepared in order to achieve a certain result. In the context of run processing (see below), a program is an absolute element to be executed as a task, and may be a processor or a user program.

User Program

Any program other than a processor. Usually developed by a user; however, certain UNIVAC system software packages operate as free-standing user programs (for example, PERT).

Element

A named grouping of information, typically manipulated as a unit, and typically defining a logical program part such as a subroutine. There are three basic types of elements: symbolic, relocatable, and absolute.

Symbolic Element

An element containing information generally in human-intelligible format (typically card images). The most common usage of symbolic elements is as source language to be input to a language processor.

Relocatable Element

An element containing a program part in relocatable binary format, suitable for combination with other relocatable elements to produce an executable program (absolute element). Such elements occur must commonly as the output of a language processor to be input to a collection.

Absolute Element

An element containing a complete program in binary form suitable for execution by the executive. Such elements normally occur as output from a collection of relocatable elements, with all necessary linkages and relocation performed.

Processor

A program incorporated as an integral component of the operating system. Such programs typically reside in the system library (LIB$) as absolute elements, and are invoked in a special standardized manner, but are otherwise treated as ordinary user programs. Processors fall in two broad classes: language processors and system processors.

Language Processor

A processor whose principal functions include compiling, assembling, translating, or related operations for a specific programming language (for example, COBOL, FORTRAN, ASSEMBLER, and so forth). As opposed to system processor.

System Processor

A processor whose principal functions are of a specialized systemic service or utility nature (for example, the collector, postmortem dump, and so forth). As opposed to language processor.

Collection

The process by which individual (relocatable) elements are combined to form a complete program (absolute element). This process begins with expicit specification of elements to be included, and typically involves inclusion of additional unspecified elements required to satisfy undefined references (these are most commonly obtained from the system relocatable subroutine library RLIB$).

Collector

A system processor that provides the collection function.

2.2.4. DEFINITIONS CONCERNING FILES

File

An organized collection of data, treated as a unit, and stored in such a manner as to facilitate the retrieval of each individual datum.

Catalogued File

A file known to and retained by the executive, for an indefinite period not necessarily related to the life of a particular run, and generally retrievable by runs other than the run which originally created the file. In some cases, a catalogued file may be accessed simultaneously by two or more runs. As opposed to temporary file.

Temporary File

A transient file created by, accessible to, and existing within the life of, a single run only. As opposed to catalogued file.

Master File Directory

A directory maintained by the executive to control the retrieval and retention of catalogued files.

Public File

A catalogued file that can be assigned and accessed by a run of any project. As opposed to private file.

Private File

A catalogued file that can be assigned and accessed only by runs of a particular project. As opposed to public file.

External File name

The full name by which a file is identified to the system. In addition to the basic name, full identification may require qualifier, cycle, and key information. As opposed to internal file name.

Internal File name

An abbreviated file name used on individual I/O and related operations concerning a particular file. The internal file name may have an implicit association with an external filename, or may be associated to a particular external filename by explicit programmer directive. As opposed to external file name.

Qualifier

An extension to the basic name of a file, employed to resolve a variety of ambiguous situations. Every file has a qualifier, but is normally implied according to system conventions, rather than being explicitly stated in references to the file.

Program File

A specially structured file containing a group of elements, residing on FASTRAND-formatted mass storage. As opposed to element file.

Element File

A specially structured file containing a group of elements, residing on magnetic tape. As opposed to program file. The arbitrary distinction between the two file types is made to avoid confusion between operations that may be done on one medium but not the other.

Temporary Program File (TPF$)

A mass storage file assigned automatically by the executive to each run. As a convenience to the user, in a wide variety of program file and element manipulation operations TPF$ is assumed as the program file in the absence of an explicit filename reference.

SDF Format

System Data File Format. The standard data format employed by the operating system. Briefly, SDF format is a sequential, fixed-block, variable-record format in which records may span blocks.

DATA File

A file in SDF format created or updated by one of several operating system mechanisms, usually a system processor called DATA. Not to be confused with the generic term "data file".

2.2.5. RUN PROCESSING DEFINITIONS

Control Statement

A data image, used to direct the executive in processing a run. A control statement is identified by a master space (@) in column 1.

Executive Control Language

The language in which control statements are written.

Task

A discrete processing step in a run, involving the execution of an absolute element (that is, a processor or user program). Synonymous with program in run processing contexts.

Run

Job. A specified group of tasks prescribed as a unit of work for the system. The run is the largest work grouping treated and manipulated as a unit by the executive. The tasks of a particular run are executed serially in the order specified by the run stream.

Run Stream

A sequence of data images which, taken as a whole, constitute the specification of a run. A run stream consists of a @RUN control statement, followed by other control statements and data, which direct the performance of individual tasks.

Processor Control Statement

A control statement used to direct the execution of a processor. Such statements have a standardized format which facilitates specification of parameters typically required by processors, such as element names.

Batch Processing

A mode in which runs are processed without any basic requirement for interactive manual data or control input (such as from a keyboard) during processing. As opposed to demand processing.

Demand Processing

A mode in which run processing is basically dependent on manual interaction with the system during processing, typically from a remote site. Also commonly known as 'time-sharing'. As opposed to Batch Processing.

Real Time Processing

A mode of operation in which the system's response to external stimuli is sufficiently fast to influence the process or operation being monitored or controlled so as to obtain a desired result. Generally, real time processing is under the influence of asynchronous inputs from one or more communications devices. Real time processing may occur in batch or demand mode (typically batch).

Deadline Run

A deadline run is a batch run which is afforded certain scheduling priorities to assure run completion by a prespecified time. Except for these scheduling exceptions, deadline runs are treated as batch runs by the executive.

PCT

Program Control Table. A special table maintained by the executive containing the bulk of the control information for a particular run and the program (if any) currently in execution for that run.

2.2.6. MULTIPROGRAMMING DEFINITIONS

Activity

Formally, a logical CPU. That is, a software mechanism wherein the executive maintains a CPU environment (current P register value, control register contents, and so forth) for an execution sequence or thread, called an activity, such that the activity appears to have continuous use of a single CPU as long as it desires, even though the executive may in fact, interleave CPU usage among many activities and execute them on different CPUs. All program execution is by activity. A program is initially assigned and typically needs just one activity; complex programs may register additional activities to be executed asynchronously.

Activity Registration

Forking. The act of creating and registering a new activity with the executive.

Activity Name

A general-purpose identifier acquired by an existing activity to allow other activities of the same program to communicate or synchronize with it.

Activity-id

A special-purpose numeric identifier which may be acquired upon registeration of a new activity. An activity having such an identifier may wait for the termination of one or more other activities having an id. Activity-id is not to be confused with activity name; their functions are separate and independent.

Activity Termination

The permanent cessation of execution by an activity. This is normally done voluntarily by the activity itself, but may also result from an externally initiated action such as an abort sequence. A program terminates when all of its activities have terminated.

Switching

The process by which the executive controls CPU usage. This principally involves determination of which activity(s) of which program(s) are to be executed on which CPU(s) for how long, and the control functions needed to fulfill that determination. Also called dispatching.

Multiprocessing

The simultaneous execution of multiple activities of one or more programs, by employing two or more CPUs which access a common main storage.

Multiprogramming

The concurrent (interleaved) execution of two or more programs or activities which reside in main storage. This is accomplished by sharing CPU usage through switching.

Reentrant Routine

A routine coded such that more than one activity at a time may execute the routine and still obtain desired results. Most commonly this is achieved by executing the same instructions on different data sets, frequently with some sort of locking procedure invoked at critical moments to prevent simultaneous operation on the same data set. The use of multiple activities by a program generally implies that part of the program is reentrant.

REP

Reentrant Processor. A common reentrant routine that may be referenced by more than one program simultaneously. Typically, one or more REPs contain the bulk of the instructions necessary to a particular task, with the data area provided by the referencing program(s). In this manner, a substantial saving in main storage space can be achieved when several runs require performance of the same or very similar tasks. (For example, Conversational FORTRAN).

2.2.7. MISCELLANEOUS DEFINITIONS

ER

Executive Request. An instruction which causes a special interrupt used to request executive service (for example, I/O, time of day, and so forth). Also, the service resulting from the request. This is the standard interface between programs and the executive.

PMD

Postmortem Dump. A printout of a program's main storage contents following execution. Also, the system processor which produces the printout.

Swapping

The process of storing low priority or suspended programs on mass storage to allow main storage space to load other higher priority programs.

Symbionts

A complex of executive routines providing the user interface with unit record peripherals and nonreal time remote devices.

Breakpoint

Division of symbiont - defined files into parts such that the output of completed parts may be initiated prior to run completion. This procedure allows more efficient utilization of printers and punches when large symbiont output files are involved.

Project

An identifier used to classify a run for accounting purposes. May also be used to provide implied filename qualification to avoid confusion of similarly named files of different projects.

Cycle

A number used to differentiate successive updates of files or symbolic elements.

Contingency

An abnormal or unanticipated event requiring special action, and usually causing diversion of an activity's execution path to a specially prepared routine or to a standard action sequence.

Standard Action
Action performed by the operating system in a particular circumstance, in the absence of explicit user directive.

ACW
Access Control Word. A word defining the length and location of a data area in main storage, most commonly an I/O buffer.

Scatter/Gather
Scatter Read/Gather Write. An I/O technique wherein multiple discontiguous buffers in main storage (described by a string of access control words) are read into (scatter read) or written out (gather write) in a single continuous operation involving a contiguous area or block on the peripheral device being accessed.

Packet
A contiguous set of words that contains information to enable the execution of an operation or function to be performed, typically an ER.

Interlock
A condition by which a peripheral unit is unable to perform an executable command until the condition is removed by the operator.

Noise Constant
The size of a record, in characters, to be skipped as a noise record on parity error. Applies only to magnetic tape files.

Privileged Instruction
One of a set of machine instructions reserved for use by the executive. If the execution of a privileged instruction is attempted by a user program, a guard mode fault interrupt occurs.

Fieldata
A six-bit character code which is the native character set of the operating system. The character set and associated codes are listed in Appendix D.

Data Image or Image
A data record in human intelligible (character) format; most commonly refers to punched card or printer data.

ISI
Internally Specified Index. A mode of I/O operation for an 1100 Series computer I/O channel, wherein the I/O access to main storage is determined solely by the CPU. This is the normal mode for noncommunications I/O. As opposed to ESI.

ESI
Externally Specified Index. A mode of I/O operation for an 1100 series computer I/O channel through which many communication peripheral devices are multiplexed into one I/O channel. Each communication line has its own area in main storage for access control words and specifies this area by an identification word. An I/O operation wherein this identification word is given to the computer by the peripheral device is known as externally specified index (ESI).

ESI Completion Activity
An ESI completion activity is created when a real time program initializes a line terminal. The ESI completion activity is activated upon the detection of an ESI interrupt for the line associated with its line terminal group and operates as the highest level activity in the system while processing that interrupt.

AXR$
A system procedure that contains the numeric definitions of the standard mnemonic designators for control registers, partial word designators, and so forth, which are used in assembly language coding.

ERU$
A system procedure that defines the numeric index associated with the mnemonic designation of each executive service request (ER).

2.2.8. ABBREVIATIONS USED IN THIS MANUAL

abs	absolute
acct	account
ack	acknowledge
act	activity
ACU	availability control unit
ACW	access control word
addr	address
ADH	arbitrary device handler
AFC	abnormal frame count
ANSI	American National Standard Institute
ASCII	American Standard Code for Information Interchange
BBP	block buffering package
BCD	binary coded decimal
BPI	bits per inch
BPS	bits per second
BSP	basic service package
CA	character availability
CB	column binary
CCC	core contents control
char	character
ckpt	checkpoint
CLT	communications line terminal
col	column
cpm	cards per minute
cps	characters per second
CPU	central processor unit
CR	carriage return
CRT	cathode ray tube

CSI	control statement interpreter
CTMC	communications terminal module controller
ctr	counter
CTS	communications terminal synchronous
DA	dynamic allocator
DAPA	dynamic allocator periodic adjustment
DAS	directory allocation section
D bank	data bank
EF	external function
EI	external interrupt
eltname	element name
EOB	end of buffer
EOF	end of file
EOFMRK	end of file mark
EOI	end of input
EOM	end of message
EOT	end of transmission
ER	executive request
ESI	externally specified index
ETX	end of transmission
FAC REJ	facility reject
FAC WARN	facility warning
FCT	file control table
FGC	final granule count
FH	flying head
FPI	frames per inch
FURPUR	file utility routine/program utility routine

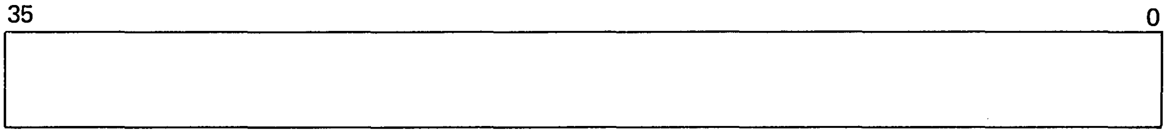
IACW	input access control word
I bank	instruction bank
int	interrupt
INFOR	internal control statement format
I/O	input/output
IOC	input/output controller
ISI	internally specified index
j-desig	partial word designator
LAF	look ahead factor
LF	line feed
LIB$	system library
loc	location
LT	line terminal
LTG	line terminal group
LTR	line terminal routine
LTT	line terminal table
MFD	master file directory
MP	multiprocessor
MSA	multi subsystem adapter
msg	message
nbr	number
NOL	number of open lines
NRTF	nonreal time flag
OACW	output access control word
PCT	program control table
PDP	procedure definition processor
PET	program error table
PFP	program file package

pkt	packet
pktaddr	packet address
PMD	postmortem dump
pos	position
prev	previous
proc	procedure
PSR	program state register
refs	references
rel	relation
req	requirement
REP	reentrant processor
ret	return
RLIB$	system relocatable library
RSEG	relocatable segment
RTS	real time subroutine
SDF format	system data file format
secs	seconds
SGS	stream generation statement
SLR	storage limits register
TPF$	temporary program file
trans	translator
TRK	track
TS-flag	test and set flag
UP	unit processor
wd	word
wds	words
WSA	working storage area
WTS	word terminal synchronous

2.3. CONVENTIONS

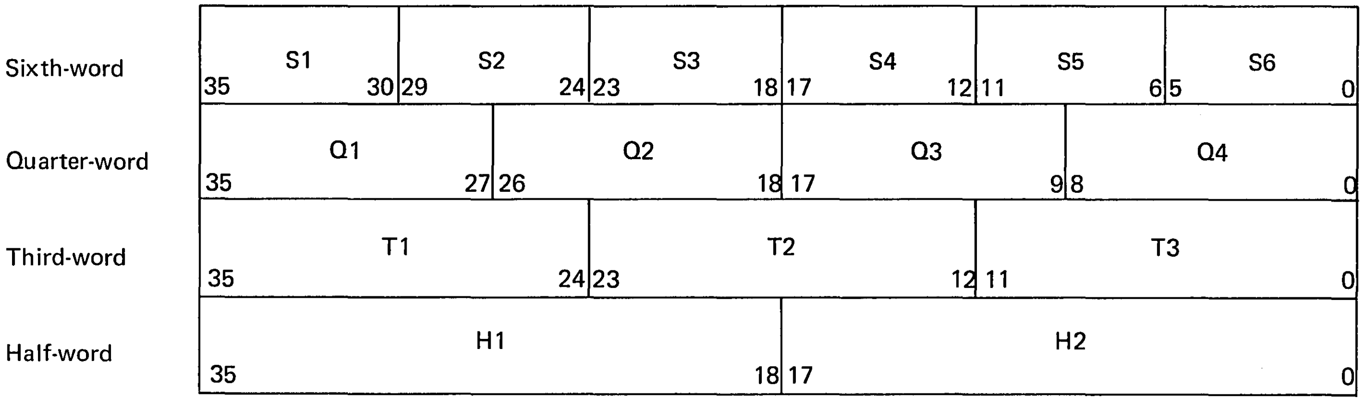
2.3.1. NOTATIONAL CONVENTIONS

The 36 bits in the 1100 series computer word are numbered from right to left. For example:



(The mnemonic W is used in ambiguous situations to indicate whole word references.)

To reference partial words, the following mnemonics are used:



The 1100 series assembler mnemonics are used whenever machine instructions are discussed (see *UNIVAC 1100 Series EXEC II & 8 Assembler Programmers Reference Manual, UP-4040,* current version). The assembler mnemonics for partial word transfers and registers are defined in system procedure AXR$ for use in assembly language routines.

The mnemonic U is used to indicate immediate data references (operand taken directly from address portion of instruction rather than from the main storage location referenced by that address).

Control registers are referenced by the following mnemonics:

■	A0, A1,..., A15	Accumulators (A0 – A3 also usable as index registers).
■	A15+1, A15+2	Additional accumulators involved in double or triple precision instructions.
■	X1, X2,...,X11	Index registers.
■	R1, R2,..., R15	R registers.

Activities may use one of two sets of control registers:

■	Major Set	All control registers as given above.
■	Minor Set	Registers X11, A0 through A5, R1, R2, R3.

The dollar sign ($) is generally used in system-defined external symbols, procedure names, and file names; to avoid duplication, the user should not use this character. The $ generally occurs as the last character of a symbol excepting procedure names in which it is the second character.

In packet and table formats, parameters in regular type indicate information that must be supplied by the programmer; parameters in italics indicate information that the system returns. Brackets ([]) are used to indicate optional parameters.

The symbol ƀ used to indicate a blank character.

When specifying the formats of procedure calls and tables, brackets are used to indicate optional parameters.

In control statement, executive request, and procedure call formats, capital letters represent themselves and must be coded as shown; lower case letters represent variables which must be coded as directed in the text.

Numbers are represented in examples as in assembler syntax, that is, a leading zero specifies octal.

2.3.2. CONTROL STATEMENT NOTATION

Control statements have the general format:

```
@label:command,options    parameters · comment
```

Parameters are given in one or more fields separated by commas. A field may specify a single parameter, or may contain several related parameters given in subfields, which are delimited by slahses. An ellipsis (...) indicates that any number of additional parameters, of the same format as the last shown, may be given (for example, reel numbers of a tape file, elements to be listed, and so forth).

See 3.2, for a complete discussion of control statement syntax.

2.4. BASIC CONCEPTS OF RUN CONTROL

2.4.1. RUN INITIATION

The executive symbiont complex provides the primary input interface between the user and the system. The symbionts control run input from onsite card readers and remote sites, as follows:

(a) In batch mode, the entire run stream is normally buffered to mass storage by the symbionts before run processing is initiated. At this point, an executive component called the coarse scheduler takes over. It examines that portion of the run stream prior to the first task for initial facilities requirements. Based on those requirements, and certain other operating parameters such as run priority and deadline time (if any), the coarse scheduler determines the proper time to open the run.

(b) In demand mode, the run is normally initiated immediately upon acceptance of the @RUN control statement. Additional run stream input generally occurs dynamically on an interactive or conversational basis. See Section 12 for a complete discussion of demand processing.

When a run is opened, two temporary files are automatically assigned to it: the temporary program file (TPF$), and the run diagnostic file (DIAG$) which is not normally referenced directly by the user.

2.4.2. RUN EXECUTION

Once a run is opened, the coarse scheduler, in cooperation with the symbiont interface routines, processes the run stream sequentially. When a control statement is encountered, the appropriate executive routines are invoked to accomplish the specified action. When a control statement that causes execution of a task is encountered, the coarse scheduler sets up the task and passes control to the dynamic allocator (see 2.5) for execution. Run stream images are then passed by the symbiont interface routines directly to the task as data, one at a time as requested by the task, until the next control statement is encountered or the task terminates (certain control statements are transparent and do not signify the end of run stream input to the task). Run stream data images (that is, images that are not control statements) are ignored with a warning diagnostic if encountered when a task is not being executed.

2.4.3. SYMBIONT OUTPUT

Every run has associated with it an output print file. In general, all control statements, executive diagnostic messages, and summary accounting information are printed in this file, as well as primary output print generated by the tasks of the run. An output punch file also is created if any user tasks generate card output.

Normally, the executive controls the disposition of these files without the need for user directives, as follows:

(a) In batch mode, the files are buffered on mass storage. At run termination, they are printed (or punched) at the site from which the run was initiated.

(b) In demand mode, print output occurs at the terminal as it is generated. Punch output is unusual, and occurs at the central site in the normal case.

2.4.3.1. SYMBIONT FILE CONCEPTS

From the standpoint of run processing, there are two basic classes of symbiont files: primary files and alternate files. They differ in usage rather than structure.

Primary symbiont files comprise the standard files through which the user communicates with unit-record equipment. There are three types of primary symbiont file:

(1) Primary Input File (READ$). This file is automatically established for each run and contains the run stream (see 2.3.1 and 2.3.2). This file cannot be manipulated, as a unit, by user directive.

(2) Primary Print File (PRINT$). Each run has associated with it a standard output print file. In general, all control statements, executive diagnostic messages, and summary accounting information are printed in this file, as well as primary print output generated by the tasks of the run.

In the normal case, the executive establishes and controls the disposition of the primary print file without the need for user directives, as follows:

(a) In batch mode, the file is buffered on mass storage. At run termination, it is printed at the site from which the run was initiated.

(b) In demand mode, print output occurs at the terminal as it is generated.

Primary print file may be thought of as an output stream. By a procedure called breakpointing, the user may direct this stream to his own files and/or partition it into several files called parts. The use of breakpointing allows printing of large volume of output to begin prior to run termination. To simplify file referencing, the current primary print file is always referenced by the generic name PRINT$, regardless of whether the file is an executive or user file.

(3) Primary Punch File (PUNCH$). Primary punch output is handled in the same fashion as primary print output, with two exceptions:

(a) No file is established unless user tasks generate primary punch output.

(b) Punch output generated at demand terminals is punched at the central site in the absence of user directives.

The current punch output file is always referenced by the generic name PUNCH$.

The user may define alternate symbiont files in addition to the primary files. The principal purpose of this feature is to allow the user multiple concurrent symbiont operations of a particular type (read, print, or punch).

Symbiont concepts and interfaces are discussed in detail in 3.6 and Section 5.

2.4.4. RUN TERMINATION

A run terminates upon reaching the end of the run stream (@FIN control statement) or as the result of an abnormal task termination. A number of actions are triggered at run termination, normally including:

- Summary accounting information is entered in the print file, including such items as run start and termination time, CPU time used, pages printed, pertinent console message, and so forth.
- Symbiont output files are closed, and queued for printing/punching in batch mode (see 2.4.3).
- Any main storage space allocated to the run is released.
- All facilities allocated to the run are released back to the system facilities pool, with the exception of mass storage space being retained in a catalogued file.
- Temporary files are freed and cease to exist.
- Catalogued files are freed, and, in the absence of assignment options to the contrary, retained. Any exclusive use interlocks are released.

2.5. BASIC CONCEPTS OF TASK CONTROL

The primary responsibility for the execution of individual tasks (programs) belongs to two executive components: the dynamic allocator, which manages main storage, and the dispatcher, which allocates CPU usage. Of these, the dynamic allocator is dominant in the sense that a task cannot use CPU time unless it is loaded in main storage.

Normally, the executive executes a number of tasks concurrently (unit processor) and/or simultaneously (multiprocessor) by time-sharing the usage of main storage and CPU time. However, in most cases the user is unaware of, and need not be concerned with, the presence of other tasks in the system.

2.5.1. REAL TIME

Runs and tasks are always initiated in demand or batch mode. Real time mode is entered only when a task requests it.

Most operational details of executive task control are of interest primarily in real time applications. These are covered in Section 16. The following sections apply only to demand and batch tasks and activities.

2.5.2. TASK INITIATION

The dynamic allocator initiates a task by allocating sufficient main storage space to accommodate the program, loading the program into main storage, assigning it a single activity with the major set of control registers and setting that activity's current instruction address (P register) to the program starting address specified at collection.

2.5.3. TASK EXECUTION AND SWITCHING

Once the task is ready for execution, its initial activity is passed to the dispatcher. Eventually, that activity is given control and allowed to execute instructions for a period of time until it either voluntarily relinquishes control (for example, to do I/O), has used up the time allotted it by the dispatcher, or is preempted by a higher priority activity, at which time the dispatcher switches to another activity (if there is one requiring CPU service). When the original activity's turn comes again, its CPU environmemt is restored and it resumes execution at point of interrupt. This switching process continues until the activity terminates; the same process also applies to any additional activities registered by the program.

A task remains loaded in main storage until it terminates, or until it must be removed (swapped) in favor of a higher priority task or because its space requirements have increased. In the case of swapping, the dynamic allocator first suspends the program by deactivating all of its activities such that the dispatcher will not attempt to given them control. The program's main storage contents are then written to mass storage. When the dynamic allocator determines that the program should be reloaded, it is then read back into main storage and its activities are reactivated (made candidates for switching).

Although batch programs may be swapped, swapping occurs must commonly for demand programs, which need not be in main storage while awaiting input from the demand terminal user.

Note that a program is not necessarily always executed in the same place in main storage, and is generally 'unaware' of where it is loaded. This is accomplished by means of hardware relocation using basing registers. Whenever a program is loaded (or reloaded) into main storage, the dynamic allocator sets the basing register values associated with each activity to reflect the program's absolute position in main storage. The program itself uses program-relative addresses which the hardware adds to the appropriate basing register to determine the true absolute main storage addresses of the program's operands and instructions.

2.5.4. EXECUTIVE REQUESTS

A program activity accomplishes its functions in two basic ways. The most common, of course, is by executing instructions; the other is by having the executive do the actual execution. This is done via the Executive Request (ER) instruction.

The ER is the principal interface between an executing program and the executive; it has the same fundamental relationship to task control that control statements have to run control. ERs are provided for a wide variety of functions, including activity control, input/output, facilities control, clocking, storage control, and so forth.

ERs related to specific areas of the executive (for example, I/O, symbionts, and so forth) are covered in associated sections. Section 4 contains a more detailed discussion of the ER mechanism, certain ERs not covered elsewhere, and a master cross-reference for all ERs.

2.5.5. MULTIPROGRAMMING CONSIDERATIONS

Programs which do not register multiple activities in general need not be concerned with the impact of switching. In essence, such programs can be written as if they have a single CPU to themselves.

Multiactivity programs, however, must take care that their activities do not interact in an undesirable fashion. For example, two activities calling a common subroutine via the Store Location And Jump (SLJ) instruction could return to the wrong address if the second activity made its call before the first had exited (this can be solved by using the Load Modifier And Jump (LMJ) instruction). Or two activities trying to update a counter in main storage concurrently could produce incorrect results. The possibilities for confusion are infinite.

For demand and batch programs, the dispatcher treats all activities equally, insofar as no activity can be certain of executing ahead of any other activity unless the programmer employs some method of synchronization; interrupt activities are given priority but only for a short period that is not programmer controllable.

Two basic approaches to activity synchronization are provided:

(1) A set of ERs which allow activities to wait for events triggered by other activities (see Section 4).

(2) By use of the Test And Set (TS) hardware instruction, which is designed expressly for synchronization of asynchronous processes. This instruction functions, in conjunction with the dispatcher, as follows:

   (a) If bit 30 of the operand is zero, the next instruction is executed.

   (b) If bit 30 is a 1, an interrupt occurs and the activity's switching priority is reduced. The activity receives control back at the TS instruction on its next turn to execute; if bit 30 is still set, the process is repeated. In this fashion, the activity spirals downward in priority until bit 30 is reset to 0 and execution then proceeds to the next instruction.

(c) Regardless of the setting of bit 30, bit 30 is set to 1 and bits 35–31 are cleared to zero; bits 29–0 are unaltered.

(d) The testing and setting of bit 30 occurs in a single main storage reference, thus assuring that two CPUs cannot both find bit 30 set to zero.

When the protected sequence has been completed a simple store zero (SZ) instruction will clear the test and set condition. The instruction sequence for protecting against concurrent execution of a critical instruction sequence is:

```
TS       IND
CRITICAL SEQUENCE
SZ,SI    IND
```

where:

IND is any main storage location; it may be associated with a particular data set to be protected or may be a global lock for the sequence.

The executive performs software simulation of the Test And Set (TS) instruction for UNIVAC 1108 unit processors which do not have test and set hardware. The a field of the TS instruction must be zero for it to be recognized as a TS. Also, the h bit (automatic incrementation) must be zero. The simulation code does not check for this, and errors result if the h bit is set and the increment portion of the index register in use is nonzero. The test and set should not be executed by way of the Execute Remote instruction.

2.5.6. TASK TERMINATION

A task terminates when all of its activities have terminated. On termination, main storage space for the program is released. On a normal termination, the coarse scheduler continues processing the run stream.

An error termination occurs when one or more activities terminate in error. In this case, further run stream processing is usually limited to at most the processing of a postmortem dump (PMD).

All ERs are subject to extensive validation to ensure against interference with other runs. In most cases, when an error is found, the activity is placed in error mode. This does not necessarily mean that the activity acutually terminates, but rather that a contingency occurs; standard action is error termination in the absence of a user specified contingency routine.

See 4.3.2 and 4.9 for details on activity termination and contingencies.

2.5.7. PROGRAM PROTECTION

The multiprogramming capabilities of the executive system imply that many unrelated programs may be residing in main storage at the same time. Such jobs may be real time, production, classified, or simple debugging. Infringement of privacy in such a mixture is highly probable especially in cases where debugging tasks are executing. The knowledge or ignorance of an invasion may range from little or no concern for some runs to great concern for classified or real time applications.

To combat this invasion, intentional or unintentional, the executive system has unique features that automatically guarantee absolute protection for each program. The protection guards against two forms of invasion, direct and indirect.

Direct protection safeguards all programs in main storage from an active program that may attempt to read, write, or jump into another program area. This safeguard is effected by locking out any area of main storage that is not assigned to the presently active program or, in effect, locking in the active program. Any attempt to perform any of the above functions is immediately reported to the executive system in the form of a guard mode fault interrupt. The same applies to the attempted use of privileged instructions.

Indirect protection is realized by reserving certain control functions for the exclusive use of the executive system. These functions are of the type that could cause a system malfunction and, in turn, a program malfunction if erroneously used. The executive system will prohibit the direct use of these functions; ERs are used to achieve the desired result.

In both forms of protection, the executive system is, in reality, guaranteeing its own safety from abuses that may prove catastrophic to the system.

2.6. FILE NAMES AND ELEMENT NAMES

2.6.1. FILE NAMES

Each file in the operation system is assigned a unique name to distinguish it from all other files. The file name is required in the many control statements and directives that are used to reference files.

The following notation is used to specify file names:

```
[[qualifier] *] file-name[(F-cycle)] [/[read-key] [/write-key] ]
```

File-name is used as the basic name of the file and qualifier is used to establish uniqueness between files that have the same basic name. F-cycle is used to identify a particular file from a set of related files that have the same qualifier, file-name, read-key, and write-key (see 2.6.3). Read-key and write-key are used to obtain read and write access, respectively, to the file; these keys are not a part of the file name for purposes of assigning a file (see 3.7).

Qualifier and file-name each consist of from one to twelve characters selected from the set: A–Z, 0–9, –, and $ (see 2.3.2 for caution on the use of $). F-cycle is an integer number (see 2.6.3 for range of values). Read-key and write-key each consist of one to six characters; any Fieldata character may be used except blank, comma, slash, period, and semicolon.

All parameters, except file-name, are optional when naming a file. For those parameters that are omitted, the executive provides standard values. Whenever qualifier is omitted but the * is specified, the qualifier specified on the last @QUAL control statement (see 3.7.6) is used; however, when no @QUAL control statement is given or if both qualifier and the * are omitted, the project-id from the @RUN control statement (see 3.4.1) is used as the qualifier. If the read-key or write-key is omitted, blanks are supplied as the key. If F-cycle is omitted when naming a set of files, the relative cycle number of –0 is supplied (see 2.6.3).

2.6.2. EXTERNAL AND INTERNAL FILE NAMES

The term 'external file name' refers to the name of a catalogued file or to a temporary file assigned to a particular run. As previously stated (see 2.6.1), it may be necessary to specify the qualifier, file-name, and F-cycle to ensure that the file is uniquely identified. In the case where the file is uniquely identified, but file-name is not unique (either qualification or cycling has caused unique identification), it is necessary to attach a unique file-name (internal filename) to accomplish I/O or related operations. It may also be convenient to have a short name by which to refer to a file that has a long external file name; or a standard name which is attached to the particular file assigned or to be assigned to the run. By means of the @USE control statement (see 3.7.5), the executive provides the capability of attaching such an alternate or 'internal filename' to a file for referencing within a run. For example, the file name EZ can be made the internal filename for file ABCDEFGHIJLM*MLJIHGFEDCBA(–23) by the control statement:

```
@USE   EZ,ABCDEFGHIJLM*MLJIHGFEDCBA(–23)
```

For the remainder of the run, whenever a reference to the file is to be made, EZ can be used as the filename. Thus the file can be referenced by either the external or internal filename. Several internal filenames can be attached to a particular external file name by multiple @USE control statements. For example:

```
@USE   EZ,ABCDEFGHIJLM*MLJIHGFEDCBA(–23)

@USE   BACKUP,EZ

@USE   Q,BACKUP
```

The file can now be referenced by the filenames:

```
ABCDEFGHIJLM*MLJIHGEFDCBA(–23)

EZ

BACKUP

Q
```

The internal filename established is valid only for the particular F-cycle of the external filename. Thus, the internal filenames established in the foregoing example are valid only for cycle –23 of the set of files.

The I/O device handlers (see Section 6), the symbiont interface routines (see Section 5) and others use a 12-character name to reference files; therefore, each file so referenced in a run must have a unique 12-character name by which it may be referenced. The following is an example of the use of internal file names for referencing files with nonunique file-name parameters:

```
@USE   NEWCYC, ID*FILESET(–0)

@USE   OLDCYC, ID*FILESET(–1)

@USE   OTHER, QUAL*FILESET

@USE   SAME, UNIQUE*PF
```

The internal filenames NEWCYC and OLDCYC may be used to accomplish I/O on different cycles in a set of files. The name OTHER is used distinguish the file to which it is attached from all other files with file-name FILESET. Assuming that no other file with file-name PF is used in the run, file-name PF or SAME may be used in I/O and related references to uniquely identify the file to be accessed.

Internal filenames may contain one to twelve characters from the set: A-Z, 0–9, –, and $.

2.6.3. FILE CYCLES (F-cycles)

In order to produce and maintain a set of catalogued files with the same qualifier, filename, read-key, and write-key, an integer parameter called F-cycle is associated with each file. The use of F-cycles enables the user to manipulate any of the set or the entire set of files without modifying his run stream.

A system-standard maximum of 32 consecutively numbered F-cycles may be retained in the set of files. This value may be set to any value needed (from 1 to 32) for a particular set of files by the use of standard may be changed at system generation. Files within the set of files maybe referenced by using either an absolute or relative F-cycle number. Relative F-cycle numbers are integers in the range -0 to -31. As a new cycle of a file is being constructed, its F-cycle must be specified as +1 if the relative F-cycle scheme is being used. When the new cycle of the file is catalogued by freeing the file or by run termination, its relative F-cycle number is set to -0 and existing files of the set have their relative F-cycle number decreased by 1. In addition, the name of the file is treated by the executive as a new filename (even though it has the same name as a previously existing file) until the file is actually catalogued. At this point, the executive recognizes the fact that the updated file is actually a new cycle of an already existing set of files, and is catalogued and handled accordingly. When the maximum number of F-cycles that may be retained is exceeded, record of the file with the most negative relative F-cycle number is deleted and, if the file is stored on mass storage, the file itself is deleted.

When a file is deleted from a set, it may be necessary to change the relative F-cycle number of other files in order to maintain consecutive numbering from -0 to -n. If the file at the beginning of a set is deleted, the relative F-cycle of all other files must be increased by one; whereas, if a file is deleted from the middle of a set, only those files with more negative relative F-cycle numbers need be adjusted. The deletion of the last remaining file of a set causes all recognition of the set to be removed from the system.

As a file of a given set is being created, a number is assigned to it. This number is called the absolute F-cycle. Absolute F-cycle numbers are unsigned integers that begin with 1 and continue through 999, at which point, the numbering recycles to 1. The circular assigment of cycle numbers does not cause conflicts since a maximum of 32 consecutively numbered F-cycles may be retained.

Absolute F-cycles and their usage differ from relative F-cycles in two respects:

- An absolute F-cycle number is permanently assigned to a file – a file's relative F-cycle may change as files are added to and deleted from the set.

- The absolute F-cycle numbers of files in a set need not be consecutive – consecutive numbering of relative F-cycles in a set is always maintained.

As an example of the correlation between the absolute and relative F-cycle numbering schemes, assume that the most recently catalogued cycle of a file had an absolute F-cycle number of 28. When referring to this file, the F-cycle could be given as 28 or -0. Now assume a new cycle of the file is to be catalogued. Since the file has not yet been catalogued, its relative F-cycle is +1 and its absolute F-cycle is 29. When the new file is catalogued, the absolute F-cycle remains at 29 but the relative F-cycle becomes -0. The file whose absolute F-cycle is 28 is now given a relative F-cycle of -1; the file whose absolute F-cycle is 27 is now given a relative F-cycle of -2; and so on down the line. The use of relative F-cycle numbers enables the user to refer to a particular relative backup file.

For example, assume that the nature of the program requires that a particular file always be created from a file four cycles earlier than the file being created and later catalogued. The relative F-cycle of -3 could be used when referencing the earlier file and this reference would be valid each time the program is run. If an absolute F-cycle notation had been used, the F-cycle designation must be changed each time the program is run.

2.6.4. ELEMENT NAMES

Three different types of elements are recognized by the operating system:

(1) symbolic elements,

(2) relocatable elements, and

(3) absolute elements.

Typically, symbolic elements contain source language images for language processors or executive control statements and data to be processed by other control statements. Relocatable elements are the binary output of certain processors such as COBOL and FORTRAN. Absolute elements are the output of the collector.

Elements of any or all of the three types are contained in program files (see 24.2.1) on mass storage or in element files (see 24.2.2) on tape. Each element is uniquely identified within a file by its element type and element name. Of course, each file in the operating system has a unique name and the filename is used as part of the element name to uniquely identify each element. The following notation is used to specify element names:

```
[[filename].]element-name[/version][(cycle)]
```

Filename is the name of the file in which the element is contained; filename conforms to all the rules specified for file names in 2.6.1. Element-name is used as the basic name of the element and version is used to establish uniqueness between elements in the same file that have the same type and element-name. Cycle is used to specify a particular update of a symbolic element; cycles are not used for relocatable and absolute elements.

Element-name and version may consist of one to twelve characters selected from the set: A–Z, 0–9, –, and $ (see 2.3.2 for caution on use of $). Cycle is an integer number (see 2.6.2 for range of values). All parameters, except element-name, are optional when naming an element. For those parameters omitted, the executive supplies values according to standard rules. If cycle is omitted for a symbolic element, relative cycle –0 is supplied in order to select the most recent cycle. If version is omitted, blanks are supplied. Omission of filename and the period implies reference to the run's temporary program file (TPF$). Filename must refer to a program file, unless otherwise stated. In a series of element names such as on a processor call statement (see 9.4), the period may be given without filename to specify the same file as for the preceding element in the series.

2.6.5. SYMBOLIC ELEMENT CYCLE

To save altered or deleted images (updates) within symbolic elements, an integer parameter called cycle is associated with each symbolic element. New cycles are created by specifying the U option on the processor call statement (see 9.4).

Each item (image) in a symbolic element has a cycle number that indicates to which element cycle it belongs, and, if deleted, a delete cycle number to indicate in which cycle the item was deleted. When an element is updated, the updated items are inserted in their proper position and they are given a cycle number one greater than the last cycle of the element. Any items deleted by the update are so marked.

When specifying a symbolic element for compilation or assembly, the user may select a specifc update from a sequence of retained updates by referencing the proper cycle number as part of the element name. In compilation, the update entry is combined with the element in its complete state thereby creating a complete element as of that cycle.

A system-standard maximum of five, consecutively numbered cycles may be retained in a symbolic element. This maximum may be set to any value needed (up to 63) for a particular element by the use of the @CYCLE control statement (see 8.2.16). In addition the system standard may be changed at system's generation time. As soon as the number of retained updates for an element exceeds the specified maximum, the update with the lowest numbered cycle is combined with the update having the next higher cycle number to create a new element which in effect becomes the oldest cycle of the element.

A particular cycle may be referenced by either an absolute or a relative cycle number. Absolute cycle numbers are unsigned integers in the range 0 to 62; however, since only a limited number of cycles are retained, the absolute cycle numbers used when referencing an element must be in the range of those absolute cycles retained. Relative cycle numbers are signed integers. If the relative cycle is given as —n, then absolute cycle r—n is referenced, where r is the most recent absolute cycle retained. If +n is used, then absolute cycle x+n is referenced, where x is the oldest absolute cycle retained. The use of relative cycle numbers makes it unnecessary to know the absolute cycle number of either the oldest or most recent cycle retained.

Since absolute cycle numbers may not be greater than 62, when absolute cycle 62 of an element is updated all retained cycles are renumbered. The renumbering assigns cycle 0 to the oldest cycle retained, 1 to the next oldest, and so forth. For example, assume that a maximum of three cycles may be retained, cycles 60, 61, and 62 are currently retained, and cycle 62 is to be updated; as a result of the update, the element would contain cycles 0, 1, and 2. Cycle 0 is the equivalent of the previous cycle 61, 1 is the equivalent of 62, and 2 is the update of cycle 62; cycle 60 was dropped since a maximum of three cycles may be retained.

The technique of using cycled symbolic elements offers two distinct advantages over other methods:

(1) The equivalent of many different copies of the same element can be kept while requiring very little additional storage space over that needed for a single copy.

(2) Earlier copies of the element can be referenced without having to prepare correction cards to delete later corrections. If, however, any cycle other than the latest cycle is corrected and the corrected cycle is to be retained, all cycles following the cycle to be updated are deleted. The new cycle number is the updated cycle number plus one.

2.6.6. REFERENCING FILES AND ELEMENTS

Many of the control statements and directives discussed in this manual require that the particular file or element desired be specified. If the control statement or directive specifies *filename*, the following form is used:

```
qualifier*file-name(F-cycle)/read-key/write-key
```

If *eltname* is specified, the following form is used.

```
filename.element-name/version(cycle)
```

If *name* is specified, either a filename or element name may be used.

On certain control statements (such as those of the FURPUR processor, see Section 8), if a filename is expected but the field is left empty, reference to the run's TPF$ is assumed. On other control statements (such as the processor call statement, see Section 9) if an element name is expected but the field is left empty, the name TPF$.NAME$ is supplied.

Note that if all optional parameters are dropped from a filename or eltname, it would seem that the executive could not distinguish between a file and element name where either may be used. In such cases, the absence of a period in the specification implies that it is an element name. Thus:

name references an element (in TPF$);

name references an element (in some implied file);

name-1.name-2 references an element (name-2) in a specific file (name-1);

name. references a file.

The use of the period to distinguish a file from an element name is required only when either of them could be specified. In some control statements and directives, only a file name is required and the executive assumes that any name specified is a filename. Confusion may be avoided by always giving a period following a filename.

2.6.7. EXAMPLES OF FILE AND ELEMENT REFERENCE

Name	Interpretation
PAYROLL*BACKUP(–2).	References relative F-cycle –2 of file BACKUP with a qualifier of PAYROLL
COST*PROG.EDIT	References element EDIT in file COST*PROG
*BACKUP.TLU/TWO	References version TWO of element TLU in file BACKUP. Qualifier will be taken from the last @QUAL control statement encountered or the project-id parameter if no @QUAL control statement was used.
PCF6.INTL(14)	References absolute cycle 14 of element INTL in file PCF6. Qualifier will be taken from project-id parameter in @RUN control statement.
SORT	References element SORT in TPF$ file.
SORT	References element SORT in TPF$ file.
SORT.	References file SORT.
SORT(–4).	Reference the fifth most recent F-cycle of file SORT.
SORT(7)/YES/NO.	References F-cycle 7 of file SORT.' The read and write keys are YES and NO, respectively.
MERGE//IN.	References file MERGE. Write key is IN and read key is blanks.
MERGE/IN.	Reference file MERGE. IN is the read key.
A*B(–1)/CQ/QT.C/D(–3)	References the fourth most recent cycle of version D of element C in the second most recent F-cycle of file A*B. Read and write keys are CQ and QT, respectively.

Note that when an F-cycle or element cycle is not specified, it is assumed that the most recent (that is, newest) cycle is desired.

3. EXECUTIVE CONTROL STATEMENTS

3.1. INTRODUCTION

Control of the operating environment for the 1100 series computers is accomplished through a set of executive control statements. These control statements direct the executive system in the processing of a run. Control statements may envoke executive functions such as scheduling, assignment of facilities, and so forth; or may cause the execution of a user program or a processor (that is, a task). The executive control statements are designed in a compact and descriptive manner to facilitate use and yet provide full access to all of the features and functions of the executive system.

3.2. CONTROL STATEMENT FORMAT

Control statements consist of a recognition character and three major sections:

- the label field
- the operation fields
- the operand fields

Each of these sections may contain one or more parameter fields and each of the parameter fields may be further subdivided into parameter subfields. The recognition character is the master space (@) which is a 7-8 multipunch for punched cards or its equivalent for other devices. The recognition character must always appear in column 1. The format of the control statements, with the exception of the @END, @ENDCL, @EOF, and @FIN control statements, is free-form; that is, the order of the parameter fields within the control statement is fixed, but the parameter fields are not restricted to a particular column. The aforementioned control statements must be coded exactly as shown in their respective descriptions.

The basic format of a control statement is:

	Label Field	Operation Fields	Operand Fields
@	label:	command,options	parameter-field-1,parameter-field-2,...,parameter-field-n . comment

3.2.1. LABEL FIELD

The label field need not appear but may be used to name a control statement. The label is limited to six characters from the set: A–Z, 0–9. The first character of a label must be alphabetic. If a label is specified, it must be followed immediately by the colon (:). A label is used only when dynamic adjustment of the run stream is required. The discussion of its use is given in 3.9.3.

3.2.2. OPERATION FIELDS

Unless the control statement is to be used only as a label statement (see 3.9.3), the command field must always be specified as it determines the control statement's basic operation. The command on all control statements except the processor control statements (see 9.4) is limited to six characters from the set A–Z and 0–9, the first of which must be an alphabetic. The command field is terminated by a blank, or if options are specified, by a comma.

The options field provides the user with ability to specify certain options in the form of unsequenced alphabetic characters related to the particular command. On some control statements, the options can be broken into subfields, each of which is separated by a slash (/). A blank character or a series of blank characters separates the command or options field from the operand fields.

3.2.3. OPERAND FIELDS

The operand fields specify parameters associated with the command fields. These are separated by commas and are specified by the user as dictated by his requirements. The content of each operand field, the number of operand fields, and whether each is required or optional varies with command selected. Operand fields, in turn, may contain parameter subfields that are separated by a slash (/). For the most part, these subfields are optional within a field. Thus, it is possible to specify parts of a field without specifying the entire field.

3.2.4. CONTROL STATEMENT ANNOTATION

Control statements may be annotated with comments following the operand field. At least one blank character must precede the comment. The comment itself may contain any character except the semicolon (;), which is the continuation character. The comment is terminated by the end of the card or its equivalent for other input devices. The comment is never required. If the operand parameter fields are omitted, the comment must begin with a period (.) followed by a blank. This is also true when the content of an operand parameter field is unrestricted and variable in length (as with the @LOG and @MSG control statements). The @XQT control statement is an example of a statement where operand parameter fields are possible but may be omitted.

3.2.5. CONTROL STATEMENT CONTINUATION

In certain situations, a control statement may require more than one line or card. In such cases, coding a semicolon (;) indicates continuation on the next card or line. A control statement may be split at any point, after the options field, where a leading blank is allowable or within the comment field. It is treated logically as a space. Continuation on the next line can begin in any column, with one exception: a master space character (@) must not be placed in column one of the continuation line.

3.2.6. LEADING BLANKS IN FIELDS

Leading blanks within a statement are permissible in the following cases:

- Following the master space (@) character
- Following a colon (:) when a label is specified
- Following a parameter field separator (,)
- Following a parameter subfield separator (/)

A blank, placed at any position in the coding other than those listed, is interpreted as the termination of the parameter field.

3.2.7. GENERAL DROPOUT RULES

When parameter fields and subfields are optional, the following rules apply, where an empty field is defined as one that contains no nonblank characters:

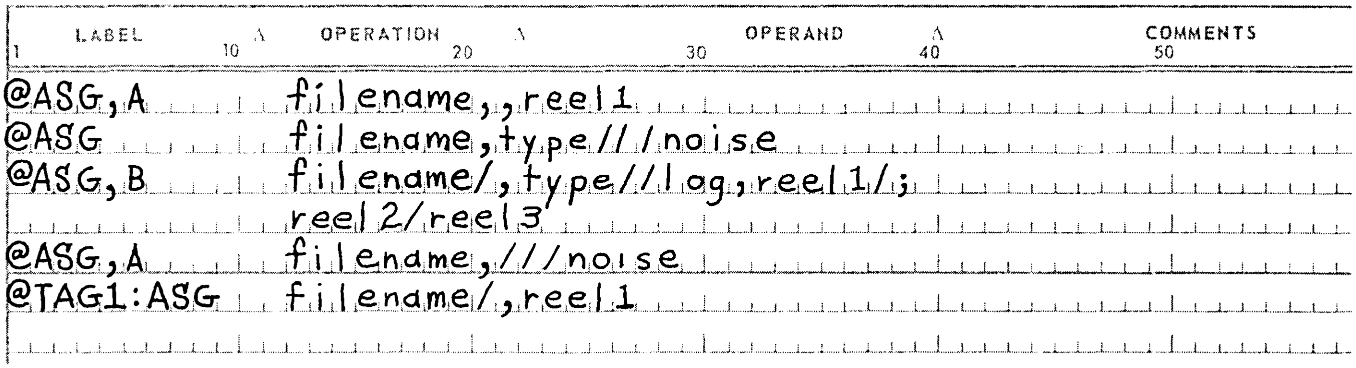
(1) Parameter field separators must be specified, left to right, through the last parameter given; fields preceding the last parameter may be empty; trailing field separators need not be specified.

(2) The same holds true of parameter subfield specifications within a field.

For example, in the magnetic tape @ASG control statement, the only required parameters are ASG and filename (see 2.6.1). The format of this control statement is:

@label: ASG, options   filename,type/units/log/noise/MSA-trans/unit-trans/format,reel1/.../reeln,expiration-period

The following symbolically illustrates the possicle coding combinations that result from applying the general dropout rules:



```
LABEL      ∧  OPERATION    ∧              OPERAND      ∧              COMMENTS
1          10            20            30             40              50
@ASG,A        filename,,reel1
@ASG          filename,type///noise
@ASG,B        filename/,type//log,reel1/;
              reel2/reel3
@ASG,A        filename,///noise
@TAG1:ASG     filename/,reel1
```

Although control statements are free-form, most programmers align the label, operation, and operand fields when coding to make the run listing easier to read.

3.3. SUMMARY OF CONTROL STATEMENTS

The executive control statements can be divided into eight groups. These groups are:

(1) Scheduling statements

(2) Message statements

(3) Symbiont directive statements

(4) Facility statements

(5) Data preparation statements

(6) Program execution statements

(7) Dynamic run stream modification statements

(8) Checkpoint and Restart statements

Table 3–1 lists all the executive control statements in their respective groups and presents a brief description of each statement's function.

Certain control statements, because of their complexity or their close association with concepts discussed elsewhere in this manual, are not discussed in this section. Table 3–1 contains references to the sections in which these statements are discussed.

Statement Group	Control Statement	Description
Scheduling Statements	@RUN	Appears at the beginning of each run. Provides accounting, scheduling, and run identification information (see 3.4.1).
	@FIN	Appears at the end of each run (see 3.4.2).
	@START	Used to initiate the execution of an independent run (see 3.4.3).
Message Statements	@LOG	Places user specified information in the system log (see 3.5.2).
	@MSG	Places a message on the operator's (cental site) console (see 3.5.1).
Symbiont Directive Statements	@HDG	Used to place a heading line on print output (see 3.6.1).
	@SYM	Used to direct nonstandard symbiont output action (see 3.6.3).
	@BRKPT	Used to segment primary symbiont output files and to close alternate symbiont files(see 3.6.2).
	@COL	Used to specify various forms of image input (see 3.6.5).
Facility Statements	@ASG	Used to assign files (peripheral devices) and catalogued files to a run. There are five types of @ASG control statements: FASTRAND format (see 3.7.1.1), tape (see 3.7.1.2), word addressable drum (see 3.7.1.3.1), word addressable unit (see 3.7.1.3.2), and arbitary device (see 3.7.1.4).
	@MODE	Used to change the mode settings (density, parity, etc) of tape files (see 3.7.2).
	@CAT	Catalogues mass storage or existing tape files (see 3.7.3).
	@FREE	Used to deassign a file and its input/output device or mass storage area (see 3.7.4).
	@USE	Used to set up a correspondence between internal and external filenames (see 3.7.5).
	@QUAL	Used to define a filename qualifier (see 3.7.6).
Dynamic Run Stream Modification Statements	@ADD	Used to dynamically expand the run stream (see 3.9.1).
	@SETC	Places a value in the condition word (see 3.9.4.1).
	@JUMP	Used to bypass a portion of a run stream (see 3.9.4.3).
	@TEST	Used to test the condition word when determining portions of the run stream to be processed or bypassed (see 3.9.4.2).
Checkpoint and Restart Statements	@CKPT	Used to establish a checkpoint dump that may be used for restart at some future time (see 17.2.1.1).
	@CKPAR	Used to establish a program checkpoint dump that may be used for restart at some future time (see 17.3.1).
	@RSTRT	Used to restart a run at some previously-taken checkpoint (see 17.2.4).
	@RSPAR	Used to restart a program at some previously-taken checkpoint (see 17.3.2).

Table 3–1. Summary of Executive Control Statements (Part 1 of 2)

Statement Group	Control Statement	Description
Program Execution Statements	@processor	Used to execute a processor (that is, @COB for COBOL compiler, and so forth), see 9.4.
	@MAP	Used to call the collector and prepare an absolute element (see 10.2.1).
	@XQT	Used to initiate the execution of a program (see 10.3.1).
	@EOF	Used to separate data within the run stream (see 10.3.2).
	@PMD	Used to take edited postmortem and dynamic dumps of the program just executed (see 11.2.1).
Data Preparation Statements	@ELT	Inserts or updates a program file element from the run stream (see 18.2).
	@DATA	Used to introduce or update a data file from the run stream (see 18.3).
	@END	Used to terminate a data file (see 18.2.1).
	@FILE	Used to cause the direct creation of a file containing data taken from the run stream (see 3.8.1).
	@ENDF	Used to terminate the data that follows the @FILE statement (see 3.8.2).

NOTE:

The total system control statement capability is not given in the above table in that the control statements for the system processors are not shown.

Table 3–1. Summary of Executive Control Statements (Part 2 of 2)

3.4. SCHEDULING CONTROL STATEMENTS

3.4.1. RUN INITIATION (@RUN)

Purpose:

Identifies the run to the executive and provides the information needed for accounting and scheduling purposes. The @RUN control statement must be the first statement of each run.

All parameters in the @RUN control statement are optional.

Format:

```
    @RUN,priority/options   run-id,acct-id,project-id,run-time/deadline,pages/cards,start-time
```

Parameters:

priority

Indicates the preference this run should be given in relation to all other runs which are available for execution. This parameter consists of a single alphabetic character selected from the set A–Z. The nearer the selected letter is to be beginning of the alphabet, the higher the priority assigned to the run. If a priority is assigned higher than that permitted for the specified account number, the executive alters the priority to the highest permissible level for that account. The executive system also supplies a standard run priority whenever the priority parameter is omitted.

options

The options are alphabetical characters that may be given in any order. The options and their meanings are given in Table 3–2.

Option Character	Description
C	Terminate the run if punched card estimate is exceeded.
E thru L	Designates the initial size of the PCT and is necessary only for real time programs. The letters have the following meaning: E = two main storage blocks, F = three main storage blocks,..., L = nine main storage blocks.
N	Inhibit all postmortem and dynamic diagnostic dumping.
P	Terminate the run if the page estimate is exceeded.
R	Restart the run in the event of a recoverable system failure.
S	Process this run in sequence with the previous run submitted from this terminal. This run is not considered for execution until the previous run has terminated.
T	Terminate the run if the run time estimate is exceeded.
Y	Allow postmortem and dynamic diagnostic dumping of processors and programs in the systems absolute library file SYS$*LIB$.

Table 3–2. @RUN Control Statement, Options

run-id

Identifies the run to the executive. Run-id may consist of one to six characters selected from the set A–Z, 0–9. If the specified run-id duplicates a run-id already in the system, the executive modifies the newly submitted run-id to make it unique. When the run-id is omitted, the executive assigns a run-id of RUN000. The numbers 000 may vary in order to establish a unique run-id. When the run-id is modified, both the original and the modified run-ids are output on the operator's console, in the master log, and in the printer listing.

acct-id

Specifies an account number. Consists of 1 to 12 characters selected from the set A–Z, 0–9, period, and dash. When omitted, the executive assigns an acct-id of 000000. If the submitted acct-id is not registered with the executive, the operator is notified and he can either:

(1) abort the run,

(2) accept the submitted acct-id which is then registered as a valid account.

(3) accept the submitted acct-id without adding to the list of allowable acct-ids.

project-id

Classifies the run for accounting purposes, and provides for the insertion of an implied qualifier for filenames. This parameter may consist of 1 to 12 characters from the set A–Z, 0–9, –, and $. If omitted, the executive provides QQQ$ as the project-id.

run-time

Estimate of run time in minutes. When preceded by the letter S, the entry is interpreted as time in seconds. If omitted, the executive assumes standard system value. When the estimated run time is exceeded, the executive:

(1) notifies the operator allowing him to manually terminate the run, or

(2) terminates the run, providing that the T option is specified in @RUN control statement or if automatic run termination was specified at systems generation.

deadline

Specifies a time (based on a 24-hour clock) by which a run must be completed. The parameter format is:

[D] hhmm

where:

hh	specifies hours
mm	specifies minutes

Leading zeros may be omitted from hhmm. The deadline time can be specified as either time of day or elapsed time from run submission. The optional D prefix indicates a time-of-day deadline; when the D is omitted, elapsed time deadline is indicated. Examples:

D910 – Run must be done by 9:10 A.M.
D2110 – Run must be done by 9:10 P.M.
100 – Run must be done one hour after submission
230 – Run must be done 2.5 hours after submission

The deadline parameter is ignored if the run-time parameter is not specified.

If a deadline becomes critical or if a deadline cannot be met by normal scheduling, the executive reschedules the run (raises run priority) so that it is completed, if possible, at the specified completion time. This action, however, degrades system operation since it most likely involves suspending other runs.

pages

Estimate of the number of printed pages expected as output from the run. When omitted, the executive assumes a standard system value. If the estimate is exceeded, the executive:

(1) notifies the operator and gives him the option of terminating the run, or

(2) terminates the run if the P option is specified or if automatic run termination was specified at system generation.

cards

Estimate of the number of punched cards expected as output from the run. When omitted, the executive provides a standard system value. If estimate is exceeded, the executive:

(1) notifies the operator and gives him the option of terminating the run, or

(2) terminates the run if the C option is specified or if automatic run termination was specified at system generation.

start-time

The earliest time at which a run is considered for processing. Before that time is reached, the run is placed in a hold state. The format of the parameter is the same as for deadline

[D] hhmm

Once the start time has been reached, the run is released from the hold state and considered for scheduling under normal priority rules.

Other @RUN control statement parameters such as deadline and priority are not interpreted until the start time has been reached; for example, the start-time parameter is considered to be the time of run submission when considering the deadline.

Description:

Postmortem and dynamic dumping is governed by three modes of operation which are established at the beginning of a run. To establish each of these modes, the user must specify the proper options parameter (N or Y) or omit the options parameter entirely. The effects of each condition are described in the following paragraphs.

- Normal Mode – No Option Specified

  When the options parameter is omitted, all programs, except processors loaded from the system library (LIB$), are dumped to the diagnostic file (DIAG$) and normal PMD action occurs. (See 11.1 and 11.2 for information concerning normal PMD action.) @PMD control statements encountered after a processor control statement are not honored and are only printed out along with the message: PMD DUMPS AND DIAGNOSTIC PRINT-OUT NOT ALLOWED. The rules governing postmortem dumping also apply to the printout for diagnostic dumping. The no-option mode should be used whenever one or more programs, other than those in the library LIB$, are being debugged. This is the normal mode for user debugging of runs, but still provides reduced overhead as opposed to the complete capability (Y option).

- N Option Mode

  When the N option is specified on the @RUN control statement, no program or processor is dumped to the diagnostic file upon run completion. @PMD control statements are not honored, but are printed along with the message described in the preceding paragraph. Diagnostic dumps are not printed. Specify the N option whenever the run is being executed for production purposes because the need for the overhead of saving the program for a possible @PMD control statement does not exist. The use of the N option provides minimum overhead in the run.

- Y Option Mode

  The specification of the Y option in a @RUN control statement establishes a mode in which all programs and processors in the run are dumped to the diagnostic file. All @PMD control statements are honored, and diagnostic dumps are printed. This option should be used only when a program in the LIB$ library is being debugged.

The R option on the @RUN control statement ensures that the run stream of an open run is recovered in the event of a system failure. If the R option is specified and the associated run is open at the time of a system failure, the run is reinstated in the schedule queue during a recovery bootstrap. The following actions and restrictions govern the use of the R option:

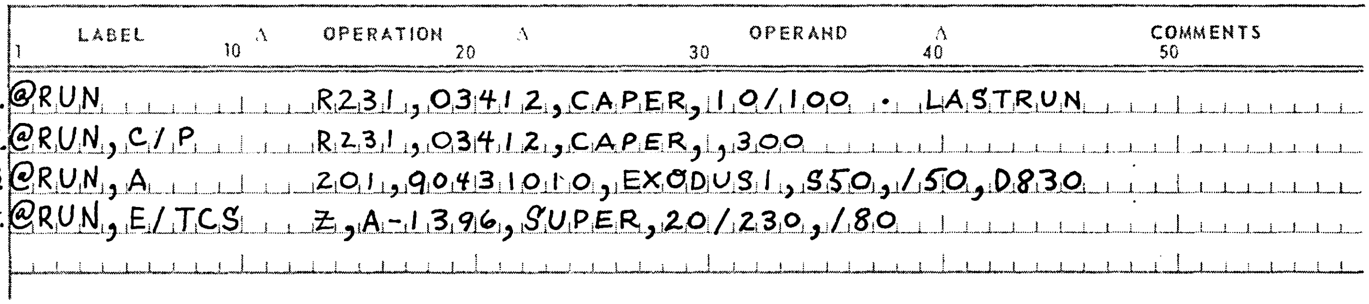
(1) All queued print and punch files generated by the open run being rescheduled are released during the recovery bootstrap unless these files are user defined files.

(2) The R option is ignored on all demand runs and these runs are never rescheduled.

(3) A run executed with an R option is rescheduled only once. This prevents a nonrecoverable situation where a particular program causes system failure each time it is executed.

(4) The programmer is responsible for handling facility assignments in his run such that conflicts during the restart of the run, resulting in the abort of this run, do not occur.

Examples:



```
   LABEL        10     OPERATION      20          30   OPERAND      40              COMMENTS
 1                                                                                  50
1.@RUN                 R231,03412,CAPER,10/100 . LASTRUN
2.@RUN,C/P             R231,03412,CAPER,,300
3.@RUN,A               201,90431010,EXODUS1,550,/50,D830
4.@RUN,E/TCS           Z,A-1396,SUPER,20/230,/80
```

1. Run R231 of project CAPER is assigned standard system priority by the executive. Expenses incurred by the run are charged to account 03412. Run-time is estimated at 10 minutes with results required within one hour of run submission. Automatic run termination does not occur if run-time, page, or card estimates are exceeded unless provided for at system generation time. The executive uses standard system card and page output estimates since these parameters are omitted. The run is scheduled by the executive for execution according to its priority (start-time parameter omitted).

2. Run R231 of project CAPER has a C priority. Automatic run termination occurs if the page output estimate of 300 pages is exceeded (P option). Expenses incurred by the run are charged to account 03412. The executive provides system standard entry for estimated run time (run-time parameter omitted) and assumes that there is no deadline since this parameter is also omitted. The run is considered immediately (start-time parameter omitted) and scheduled for execution according to its priority.

3. Run 201 of project EXODUS1 has a priority of A. Run expenses are charged to account 90431010. Running time is estimated at 50 seconds with an expected punched card output of approximately 50 cards. The start-time specifies that the run not be considered for execution before 8:30 AM. Unless provided for at system generation, automatic run termination does not occur if running time or output card estimates are exceeded, since these options were not specified.

4. Run Z of project SUPER has a priority of E and is to be considered for execution only after termination of the previous run inputted from the same device (S option). The T and C options ensure that automatic run termination will occur if the specified running time or card output estimates are exceeded. Charges incurred by the run are charged to account A–1396. Running time is estimated at 20 minutes with the results expected within 2.5 hours of run submission. Card output is estimated at 80 cards.

3.4.2. RUN TERMINATION (@FIN CONTROL STATEMENT)

Purpose:

Identifies the end of a run. The @FIN control statement must appear as the last statement in all runs.

Format:

```
@FIN
```

Description:

@FIN control statements cannot be continued and must be coded exactly as shown (punched into first four columns of the card).

When the @FIN control statement is encountered, the accounting routines are called and all facilities, temporary files, and main storage areas assigned to the run are released.

The @FIN control statement is never treated as data by the ELT or DATA processors.

When the @FIN control statement is encountered, the operating system prepares a printout summary that includes initiation time, termination time, page count of printed output, and all @LOG and operators console messages.

At a demand terminal, the @FIN control statement is normally followed either by another @RUN control statement or an end-of-transmission (EOT) keyin.

3.4.3. DYNAMIC INITIATION OF AN INDEPENDENT RUN (@START)

Purpose:

Permits the user to schedule independent batch runs where the run streams for these runs have been previously created and entered into the system. This feature allows the user to automatically schedule run execution at a time of his choosing (for example, on a daily basis). Runs scheduled by this control statement must be SDF format and must be catalogued as either data files or data elements. The run stream may be created by:

- the DATA processor
- the ELT processor
- an @ELT,D control statement
- a user program

Two formats are provided for the @START control statement. Format 1 is used when all parameters from a prestored @RUN control statement are to be used. Format 2 is used when changing all or part of a @RUN control statement.

All parameters in the @START control statement are optional except name.

Format 1:

```
@label:START    name,set
```

Format 2:

```
@label:START,priority/options    name,set,run-id,acct-id,project-id,run-time/deadline,pages/cards,start-time
```

Format 1
Parameters:

name — Specifies the file or element. The file or element named must contain all the control statements required for the run. The @RUN control statement must be the first statement and the end of the file or element denotes an implied @FIN control statement.

set — Specifies an octal number to be placed in T2 of the condition word (3.9.2) of the run being scheduled.

Format 2
Parameters:

Format 2 of the @START control statement results from the integration of the basic @START and @RUN control statements. When this format is used, all parameters in the operand fields and subfields replace the corresponding parameters in the prestored @RUN control statement. Parameters existing in @RUN control statement therefore can be modified or replaced, but not deleted, by using this format of the @START control statement.

The acct-id specified in the @START control statement replaces the acct-id on the prestored @RUN control statement. If the @START control statement does not contain an acct-id parameter, the acct-id of the run containing the @START control statement is used.

Description:

The prestored set of images must have a @RUN control statement as the first image. The @RUN control statement for the prestored run must comply with the rules detailed in 3.4.1.

The @START control statement can be used when one run is used to generate data for input by another run. In fact, the generating run could build a catalogued file containing an entire run control stream and then call for it to be scheduled. As a result of this facility, the @START control statement can be used to initiate parallel processing since tasks from different runs can be executed concurrently.

Demand terminals, through the @START control statement, can initiate a batch run whose control stream has been previously entered into a data file, thus eliminating the necessity of retyping the required control statements. Any run initiated from a demand terminal using the @START control statement is scheduled as a batch run with its output going to the onsite peripherals.

The @START control statement is of particular benefit at the central site for initiating prestored utility routines and standard production runs.

The @START control statement can be issued by a user program by means of the CSF$ request (see 4.8.1).

Examples:

```
  LABEL     ∧   OPERATION    ∧          OPERAND        ∧             COMMENTS
1          10               20              30              40             50
1.@START    FILEA.
2.@START    FILEA.,O12
3.@START    FILEC.ELTB,,R231,03414,,,0/50
```

1. The run found in file FILEA is scheduled for execution by this statement. FILEA must contain all the control statements needed to initiate the run.

2. FILEA is scheduled for execution; however, the normal execution sequence of the run for this file is altered by changing T2 of the condition word for the run to 12_8.

3. The run-id (R231), acct-id (03414), and the page/card (0/50) parameters are used instead of the values found in the @RUN control statement prestored in element ELTB of the file FILEC.

3.5. MESSAGE CONTROL STATEMENTS

3.5.1. DISPLAYING A MESSAGE (@MSG)

Purpose:

Used to display messages.

All parameters in the @MSG control statement are *required* except label and options.

Format:

@label:MSG,options message

Parameters:

options — See Table 3–3. If the C, H, I, or S options are omitted, messages are displayed on the operator's console.

message — Contains the message to be displayed. The length of this parameter is variable with a limit of 50 characters (including embedded blanks). Start-of-message is marked by first nonblank character encountered. End-of-message is signified by the last character prior to an end of line, a comment, or 50-character limit, whichever occurs first.

When the message is displayed, it is prefixed with the run-id.

The continuation character (;) and the blank-period-blank sequence which introduces a comment are not permitted as part of the message. The carriage return character, however, may be used in formatting the message. This character causes both a carriage return and a line feed.

Options Character	Description
C	Displays message on communications console.
H	Displays message on hardware confidence console.
I	Displays message on I/O console.
N	Suppresses message display. Message appears only in the users print file. N option overrides W option.
S	Displays message on operator's (system) console.
W	Holds run until operator responds to message display. The message specified by the user is displayed and requires a response by the operator. A batch run may be aborted by answering X. Any other response allows the run to continue.

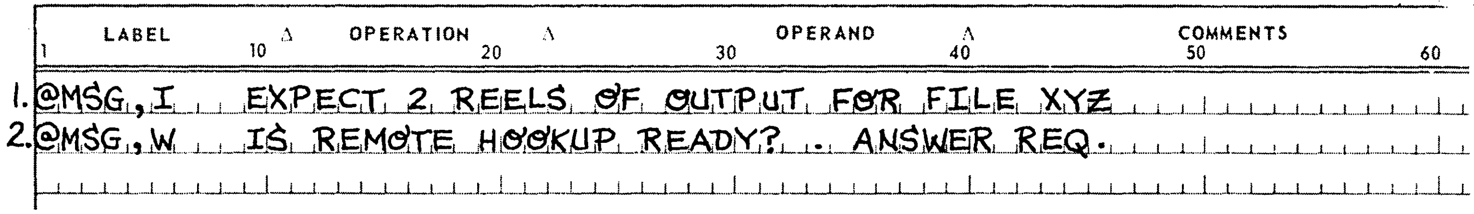
Table 3–3. @MSG Control Statement, Options

Description:

The message can be directed to one console only. The order of precedence is I, C, H, and S. If none of these options is present, the message is displayed on the operator's console.

The W option can be used to direct the operator in loading and general management of an arbitrary device for which the loading, and so forth, is not taken care of automatically by the executive.

Examples:



```
   LABEL        △   OPERATION   △              OPERAND     △            COMMENTS
 1              10            20            30            40            50            60
1.@MSG,I   EXPECT 2 REELS OF OUTPUT FOR FILE XYZ
2.@MSG,W   IS REMOTE HOOKUP READY? . ANSWER REQ.
```

1. Illustrates a 37-character message to be displayed at the I/O console (I option). The message is strictly an informational one and no operator response is expected.

2. This 23-character message requires a reply from the operator (W option). Since no particular console has been specified, the message is displayed on the operator's console; the program halts; and it will not continue until the operator replies to the message. The comment is: ANSWER REQ.

3.5.2. INSERTING INFORMATION IN THE MASTER LOG (@LOG)

Purpose:

Places user-specified information in the master log.

All parameters are *required* except label.

Format:

@label:LOG message

Parameters:

message

Contains a user-specified message to be placed in the master log. The length of this parameter is variable with a maximum limit of 132 characters including embedded blanks. The first nonblank character encountered marks the start of the message. The end of the message is signified by the last character prior to an end of line, a comment, or 132-character limit, whichever occurs first.

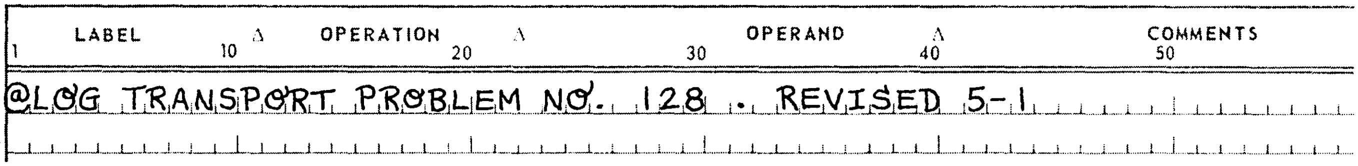
When encountered, the executive extracts the message, prefixes it with the program identification, date, and time, and enters it in the master log.

The continuation character (;) and the blank-period-blank sequence which introduces a comment are not permitted as part of a message.

Description:

See CSF$ (4.8.1) for the executive linkage which permits this function to be called from within the user program.

Example:



```
  LABEL     Δ   OPERATION    Δ          OPERAND      Δ          COMMENTS
1         10              20           30           40           50
@LOG TRANSPORT PROBLEM NO. 128 . REVISED 5-1
```

In the example, the message consists of 26 characters (the blank preceding the period is considered part of the message) and is terminated by the comment:

REVISED 5–1

3.6. SYMBIONT DIRECTIVE STATEMENTS

The following paragraphs describe the control statements related to symbiont operations. See 2.3.3. ,and Section 5 for additional discussion of symbionts.

3.6.1. PRINT OUTPUT HEADING CONTROL (@HDG)

Purpose:

Provides a means of printing a heading on successive pages of primary printer output along with the print file's cumulative page number and the current date.

All parameters in the @HDG control statement are optional.

Format:

@label:HDG,options heading

Parameters:

options

The options are:

N – Suppresses printing of heading, date, and page number

P – Starts page numbering with PAGE 1

X – Suppresses printing of date and page number heading — Contains the heading which is to appear within the top margin of each page (two print lines prior to the first logical print line of the page). This parameter is of variable length and is limited to a maximum of 96 characters including embedded blanks.

Description:

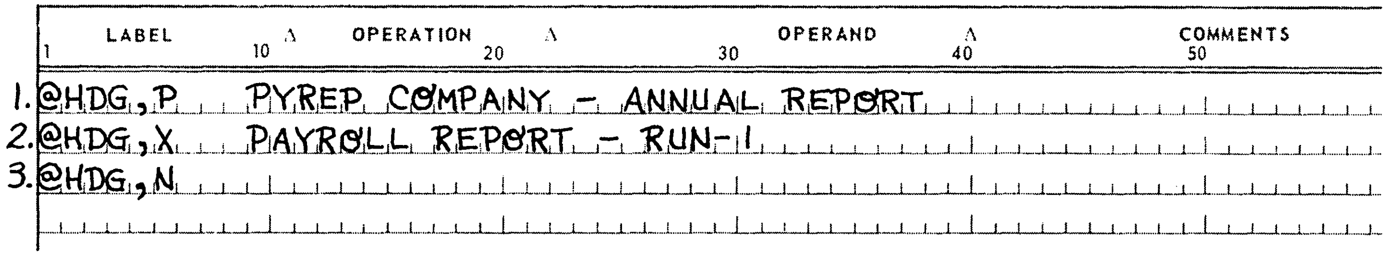
The heading starts with the second character after the operations field. Leading blanks are permitted in the heading. The end of the heading is signified by the end of the statement, a period, or the 96-character limit, whichever occurs first.

If the margin at the top of a page is nonexistent or consists of only one print line, the header is suppressed and not printed. Unless suppressed (N or X options), the date and page number are printed to the right of the header. Page numbering begins with the page count current to the print file unless the P option is specified, in which case page numbering begins with PAGE 1.

Any number of @HDG control statements may appear in the run stream.

The continuation character (;) and the blank-period-blank sequence which introduces a comment are not permitted as part of the heading format.

Examples:



```
   LABEL      Δ   OPERATION     Δ              OPERAND       Δ              COMMENTS
1          10                 20              30              40              50
1.@HDG,P    PYREP COMPANY - ANNUAL REPORT
2.@HDG,X    PAYROLL REPORT - RUN-1
3.@HDG,N
```

1. The heading PYREP COMPANY – ANNUAL REPORT is printed at the top of each printed page along with the date and the page number. Page numbering begins with page number 1 (P option).

2. The specified heading appears at the top of each printed page but page numbers and date are suppressed (X option).

3. The N option suppresses further printing of the heading, date, and page numbers.

3.6.2. SYMBIONT FILE BREAKPOINTING (@BRKPT)

3.6.2.1. PRIMARY OUTPUT FILE BREAKPOINT

Purpose:

Used to partition and redirect the primary output files, PRINT$ and PUNCH$.

All parameters in the @BRKPT control statement are optional except @ and BRKPT.

Format:

```
@label:BRKPT,options   generic-name/part-name
```

Parameters:

options — L – Used to provide labeling of parts when stacking multipart output on magnetic tape.

generic-name — Identifies the primary symbiont output file being breakpointed; must be PRINT$ or PUNCH$.

part-name — An internal filename (see 2.6.2) identifying a new user-assigned file to which subsequent primary output is to be written, or in the case of the L option, this parameter is required and specifies a label to be written identifying a new part to be written on a previously breakpointed magnetic tape file. Omission of part-name directs subsequent primary output to an executive-controlled file.

Description:

The @BRKPT control statement closes the previous file part. If the part is executive controlled, it is queued for printing or punching. If the part is a user-defined file, the file is closed by writing an EOF mark; no other action occurs for the file (that is, it is not freed, rewound, and so forth), and in the case of a magnetic tape file, the tape is positioned such that a new file may be started.

A user-defined mass storage file should only be breakpointed once; attempts to write multiple parts into such files causes overwriting of previous parts.

User-defined breakpoint files are not automatically printed; the user must use the @SYM control statement (see 3.6.3) to queue such files for printing.

See 3.6.3 for examples of the use of the @BRKPT and @SYM control statements.

See 4.8.1 for the linkage used to invoke a @BRKPT control statement from within a user program by means of the CSF$ service request.

The @BRKPT control statement may be used from a demand terminal. However, when breakpointing PRINT$ to a user-defined file, conversational mode is lost until the PRINT$ file is redirected to executive control (that is, the terminal) by a subsequent breakpoint.

3.6.2.2. ALTERNATE SYMBIONT FILE BREAKPOINT

Purpose:

Used to close or partition alternate print, punch, and read files defined by the user (see 5.1.2).

All parameters in the @BRKPT control statement are optional except @ and BRKPT.

Format:

```
     @label:BRKPT,option   internal-filename
```

Parameters:

option — E – Inhibits EOF positioning for alternate read files on magnetic tape.

internal-filename — Identifies the alternate read, print, or punch file being breakpointed.

Description:

The discussion on primary file breakpointing (3.6.2.1) is generally applicable to alternate file breakpoints. The differences are:

- ◘ The alternate file is closed in that it is no longer known to the symbionts.

- In the case of input tape files, breakpoint is normally used to prematurely terminate reading of the file; in the absence of the E option, the input tape is positioned forward to the next EOF mark so that a subsequent file on the tape may be initiated as a new alternate read file. Using the E option avoids needless tape movement when the user is finished with the tape.

- Stacking of output on tape is possible, but the user must provide his own part labelling by varying the internal-name (see 3.7.5) for each alternate file written to a tape.

- Error and status codes applicable to the @BRKPT control statement can be found in Appendix C.

3.6.3. SYMBIONT OUTPUT FILE QUEUING (@SYM)

Purpose:

Directs the queuing of previously-created symbiont files to a specified device, or group of devices, for printing or punching.

All parameters in the @SYM control statement are optional except @ and SYM.

Format:

```
@label:SYM,options   filename.,device,part-name-1/part-name-2,.../part-name-n
```

Parameters:

options

The options are:

C – Directs the file to the card punch at the remote site specified in the device field. If omitted implies printing when @SYMing to a remote site.

U– Inhibits decataloguing of the file when processing is completed; applicable only to user-defined files.

filename

Specifies the file to be processed. If filename is a user-defined file, it must be a catalogued public file. Otherwise, filename must be a generic name (PRINT$ or PUNCH$).

device

Specifies the device on which the file is to be printed or punched. This may identify a specific onsite device, a specific remote site, or a group of onsite devices (that is, the group might be all onsite punches, or all onsite 1004 printers). Device group and remote site identifiers are defined at system generation. If omitted, the devices associated with the run initiation device are assumed.

part-names

Specifies the labels (see 3.6.2.1) of the symbiont file parts of a multifile tape to be printed or punched. If omitted, only the first part on the tape is processed. This parameter is not applicable to mass-storage files.

Description:

At system generation, an association of output devices to input devices is established to allow the system to direct output files created by run stream execution to the proper output device. The @SYM control statement is used to direct a standard PRINT$ or PUNCH$ file to a device or group of devices other than that specified by system generation, or to direct a user file to a device for processing. The @SYM control statement may be used to queue any SDF-formatted file.

All user-defined files processed by the @SYM control statement are decatalogued after processing unless the U option is specified. If multiple @SYM control statements are submitted for a given file, each @SYM control statement must contain the U option to ensure that the file is not decatalogued between the processing of the individual @SYM statements.

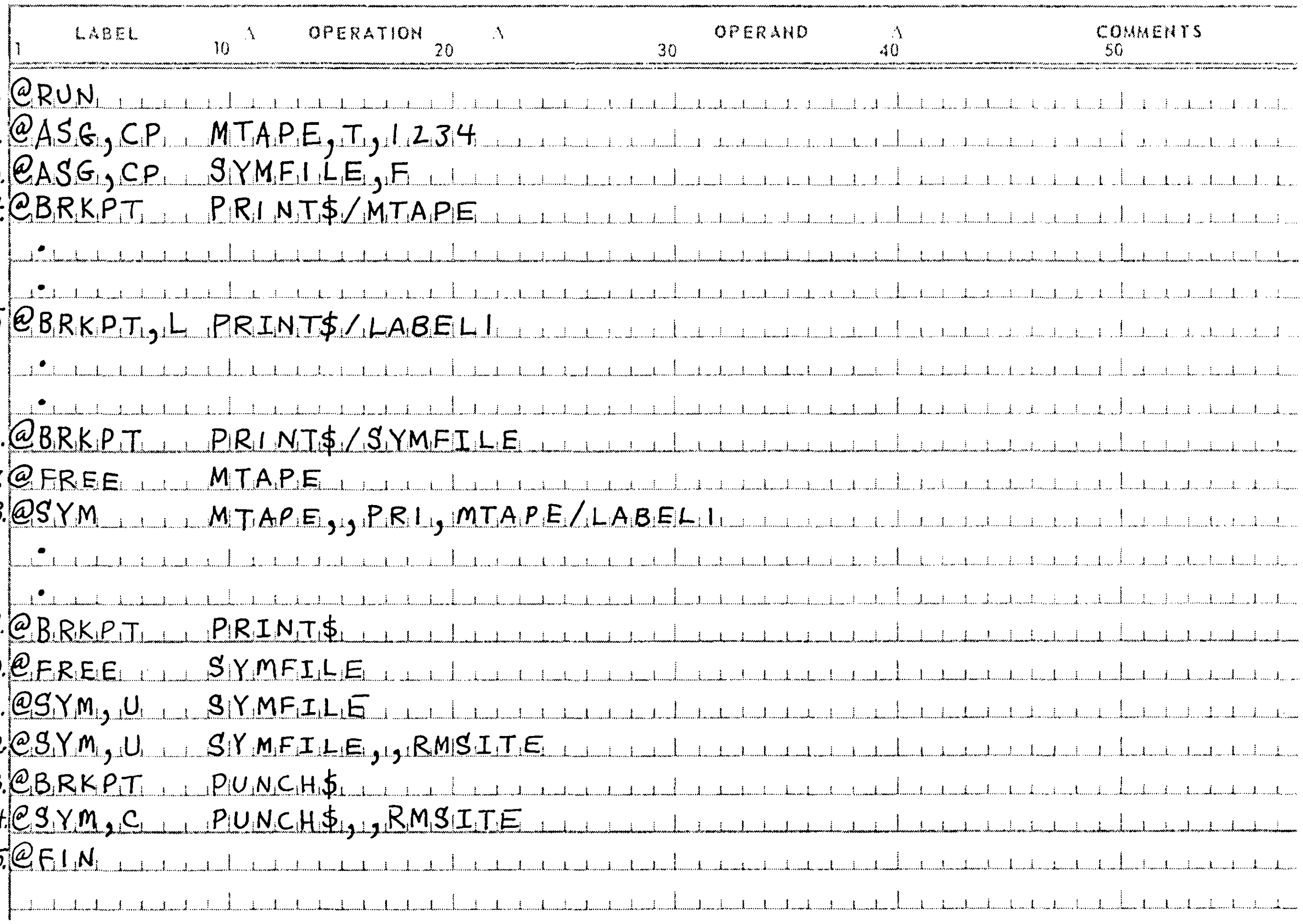
When filename is PRINT$ or PUNCH$, the directive applies only to the current primary output (print or punch) part being created. In this case, the primary output cannot have been breakpointed to a user-defined file.

The order in which the part names are specified on the @SYM control statement must correspond to the order in which the parts are located on tape. However, not all parts on the tape need be processed; any parts located on tape between those named on the @SYM control statement are bypassed.

Error and status codes applicable to the @SYM control statement can be found in Appendix C.

3.6.4. @BRKPT/@SYM CONTROL STATEMENT USAGE

The following example illustrates the usage of the @BRKPT and @SYM control statements.


LABEL
1
10
OPERATION
20
30
OPERAND
40
COMMENTS
50

```
1.  @RUN
2.  @ASG,CP    MTAPE,T,1234
3.  @ASG,CP    SYMFILE,F
4.  @BRKPT     PRINT$/MTAPE
     .
     .
5   @BRKPT,L   PRINT$/LABEL1
     .
     .
6.  @BRKPT     PRINT$/SYMFILE
7.  @FREE      MTAPE
8.  @SYM       MTAPE,,PR1,MTAPE/LABEL1
     .
     .
9.  @BRKPT     PRINT$
10. @FREE      SYMFILE
11. @SYM,U     SYMFILE
12. @SYM,U     SYMFILE,,RMSITE
13. @BRKPT     PUNCH$
14. @SYM,C     PUNCH$,,RMSITE
15. @FIN
```

This run partitions its primary print output into five parts and its primary punch output into two parts.

The first and last print parts, and the first punch part, are handled automatically by the executive, and are processed on the devices associated with the run initiation device.

The second and third print parts are stacked on the magnetic tape file MTAPE, with the labels MTAPE and LABEL1 respectively. Line 8 prints these parts on onsite printer PR1. Tape file MTAPE is then decatalogued.

The fourth print part is written on mass storage file SYMFILE. Line 11 prints this part on one of the devices associated with run initiation. Line 12 prints the same part at remote site RMSITE. SYMFILE is not decatalogued (note requirement for U option to allow multiple @SYM control statements).

Line 7 and 10 are required, since cataloguing does not occur until a file is freed.

Line 14 punches the second punch part at remote site RMSITE. Note that the C option is required; otherwise, printing would occur. Also note that no other user directives (such as assignment of a file) are required to direct this part to the remote site, since it is executive-defined.

3.6.5. CARD READER MODE CONTROL (@COL)

Purpose:

Permits the user to switch read mode from the defined system standard to the read mode specified by the first parameter on the @COL control statement. The @COL control statement is only valid when read from an onsite card reader.

All parameters on the @COL control statement are optional except xx.

Format:

```
     @COL   xx,sentinel
```

Parameters:

xx — Specifies the mode in which the input data following the @COL control statement is to be read from the control stream. The user must specify the characters CB to switch to column binary mode when input is from 900-cpm reader or 1004 subsystem. The specified input mode remains in effect until a termination sentinel is encountered or the end of the input stream is detected, at which time the standard system mode is restored.

sentinel — Specifies user-defined sentinel for terminating the nonstandard read mode data input stream. The sentinel may consist of from one to five characters.

The nonstandard read mode is terminated by encountering a termination control statement in the input run stream. The termination control statement consists of the one-to-five-character sentinel in the sentinel field of the @COL control statement preceded by the master space (@) character. If omitted, the standard system sentinel (ENDCL) is assumed. Use of the characters FIN results in termination of the input run stream in addition to the nonstandard read mode.

Description:

To properly condition the input media for handling the change of input mode, the control stream must be arranged so that the @COL control statement, @ENDCL control statement, or sentinel statements are followed by three blank cards which, in turn, are immediately followed by the data cards to be read in the new input mode. The termination control statement containing the end sentinel image for the input mode must follow the data cards. The @COL control statement and its accompanying END sentinel statement are always processed at input time and in no case is action delayed until execution time. In addition, the three blank cards which trail these two statements are eliminated from the run stream.

Examples:

```
      LABEL      Λ    OPERATION    Λ            OPERAND    Λ              COMMENTS
   1          10                20              30          40               50
1. @COL     CB
2. @COL     CB,NEWSN
```

1. The input mode is switched to accept column binary input. The standard system end sentinel (ENDCL) is assumed (sentinel omitted). The run stream associated with this statement should appear as follows:

   @COL CB

   Three Blank Cards

   Binary Data Cards

   @ENDCL

   Three Blank Cards

   Continuation of Run Stream

2. The user-specified terminating (ending) sentinel NEWSN is used to terminate the column binary input mode. The run stream for this example should appear as follows:

   @COL CB,NEWSN

   Three Blank Cards

   Binary Data Cards

   @NEWSN

   Three Blank Cards

   Continuation of Run Stream

3.7. FACILITY CONTROL STATEMENTS

3.7.1. ASSIGNING FILES AND PERIPHERAL DEVICES (@ASG)

The @ASG control statement is used to name a file, state its I/O facility requirements, and assign it to the requesting run, under the given external filename. If the file is catalogued, the facility requirements are known and need not be specified when assigning the file. The information pertaining to files and file naming presented in 2.6 is a prerequisite to assigning and cataloguing files.

The variety of I/O devices available makes several formats necessary for this statement. The five basic formats are:

(1) FASTRAND @ASG control statement (see 3.7.1.1)

(2) Magnetic tape @ASG control statement (see 3.7.1.2)

(3) Word addressable drum @ASG control statement (see 3.7.1.3.1)

(4) Unit assignment of mass storage @ASG control statement (see 3.7.1.3.2)

(5) Arbitrary device @ASG control statement (see 3.7.1.4)

All user files must be assigned prior to being referenced for I/O operations. The assignments may occur in one of three ways:

(1) by an @ASG control statement

(2) by an executive request from within a user program

(3) by an executive request from within a part of the system itself, such as a system processor.

The only instances where a file can be referenced without using an @ASG control statement are when catalogued files are being named on control statements or being named in the source language input to a system processor, such as the collector. The actual assignment is then made by that part of the system handling the given control statement, and information concerning the assignment is taken from the master file directory. If any information is needed for the file assignment other than its external name, an @ASG control statement must be used to assign the file. The @ASG control statement must also be used when the file is not a catalogued file. The user is always free to assign a file prior to referencing it on a control statement. In this case, the part of the system handling control statement will detect that the assignment has already been made. The @ASG and @FREE control statements can be placed anywhere in the run stream. Dynamic @ASG and @FREE control statements may appear anywhere in the user program. These features allow the user to assign and free files as required, without tying up the files and facilities from the beginning of the run until its completion. However, the user might be forced to wait until the facility or file is made available when the request is for one of the following:

(1) a magnetic tape unit that is being used by another run

(2) an arbitrary device that is being used by another run

(3) a catalogued magnetic tape file that is being used by another run.

(4) exclusive use of a catalogued file which is being used by another run

(5) a catalogued file which is being used exclusively by another run

To prevent the possible prolonged wait of a run when requesting an exclusive-use facility, and yet not force a run to specify all requirements before the first program (task) of the run stream, the executive does:

(1) not open a run for execution until all the @ASG control statements located before the first task in the run stream have been satisfied

(2) not start the execution of a program until all the @ASG control statements located before the program call statement in the run stream have been satisfied.

On magnetic tape @FREE control statements (see 3.7.4), there is an option which releases just the file and not the physical unit. The saved physical unit is placed in the facility pool of the run and is available for reassignment at any point in this run. The unit is not returned to the executive facility pool (made available to all runs) until it is reassigned and completely released or until run termination. The user reassigns facilities through normal means, confident that the request can be immediately honored, since the run facility pool is always referenced before the executive facility pool. By using this option and, before the first program of the run, specifying the maximum amount of each type of magnetic tape the run will require at any one given time, the user has the ability to place @ASG control statements or dynamic control statements anywhere in the run stream or programs and be assured that the run or programs will always immediately receive the facility requested.

The files referenced by most of the @ASG control statements may be catalogued with read and write keys. At a later time, when the catalogued file is assigned to a run, the keys must be given in the @ASG control statement in order to read from, write into, or delete the file. The following table shows system action according to the keys specified at cataloguing time and the keys given in the @ASG control statement. The entries FAC WARN and FAC REJ in the table indicate that FAC WARNING dddddddddddd and FAC REJECTED dddddddddddd messages are displayed and inserted into the PRINT$ file (see Appendix C for the meaning of these messages).

		KEYS SPECIFIED AT ASSIGN TIME			
		READ	WRITE	BOTH	NEITHER
KEYS SPECIFIED AT CATALOGUING TIME	READ	Read Write Allowed	Abort Run FAC REJ	Abort Run FAC REJ	Write
	WRITE	Abort Run FAC REJ	Read Write	Abort Run FAC REJ	Read
	BOTH	Read FAC WARN	Write FAC WARN	Read Write	FAC WARN
	NEITHER	Abort Run FAC REJ	Abort Run FAC REJ	Abort Run FAC REJ	Read Write

The basic formats for the @ASG control statement are discussed in the following paragraphs. See the CSF$ request (4.8.1) for linkages used to call this control statement from within a user program.

3.7.1.1. FASTRAND-FORMATTED FILE ASSIGNMENT

Purpose:

Assigns FASTRAND-formatted mass storage files to a particular run.

All parameters on the @ASG control statement are optional except @ and ASG.

Format:

```
@label:ASG,option   filename, type/reserve/granule/maximum,packid-1/packid-2. . ./packid-n
```

Parameters:

options

See Table 3–4. The cataloguing options (C,G,P,R,U,V, and W) are only valid during catalogued file creation. The C and U options are used to initiate cataloguing action. The G, P, R, V, and W options place restrictions on the file when it becomes catalogued.

The option set A, D, K, Q, X, and Y is only valid when used with files which are currently catalogued. Use of the A option with any of the rest of the set guarantees their validity. Omitting the A option results in an attempt to find the file in the master file directory. A find assigns the file from the master file directory and honors the remainder of the set. A no-find results in a temporary file assign. Any attempt to delete a catalogued file (use of D or K option) requires the specification of the read and write keys, if there are any assigned to the file.

The B, E, H, and M options control the saving and restoration of catalogued FASTRAND-formatted files in connection with checkpoint/restart. The B option controls file saving whereas the E, H, and M options control file restoration. The file is always restored at restart if the E and H options are omitted.

Options Character	Description
Options For Cataloguing	
C	Catalogues file if the run terminates normally. If the file is freed prior to termination, the file is catalogues at that time. If a file by this filename already exists in the master file directory, the run is placed in error mode.
G	Indicates that when this file is catalogued, it is to be guarded against having its read key, write key, and similar protection overridden by privileged runs. (A privileged run may be initiated by the site manager for such purposes as producing a backup copy of the file on tape to use in catalogued file recovery.)
P	Catalogues file as a public file. If omitted, file is catalogued as a private file and can be accessed only by those runs having the same project-id as the run which created the file.
R	Catalogues file as a read-only file. A file catalogued with the R option cannot be overwritten. The file can only be read or decatalogued. Any activity requesting to write in the file is placed in error mode (see 4.1.4).
U	Same as C option except that the file is catalogued at run termination (regardless of the manner of termination beyond this statement). The @FREE control statement causes cataloguing prior to the termination.
W	Catalogues file as a write-only file. The file can only be written into, and in the process, be extended.
V	Indicates that when this file is catalogued, its text is not to be unloaded to tape at any time.
Options for Catalogued Files	
A	Specifies that the file is currently catalogued and ensures that the file is not treated as a temporary file if the name cannot be found. The run is terminated if the name cannot be found in the master file directory.
D	Deletes catalogued file from the master file directory (decatalogue) if the run terminates normally or when a @FREE control statement is encountered prior to run termination. This option is only meaningful if used with the A option.
K	Same as D option except that the file is decatalogued at run termination regardless of the manner of termination. A @FREE control statement decatalogues the file prior to termination. The option is only meaningful if used with the A option.
Q	Requests that this file assignment be honored even if the system has disabled the file.
X	Specifies that this run is to have exclusive use of the file until the run has terminated or the file is released by a @FREE control statement. (If the file is not currently catalogued, the X option is not needed because the run has exclusive use of temporary files.
Y	Requests that this file be assigned only for the purpose of examining the master file directory. Exclusive use is overridden by this option but the file cannot be read or written when this option is used.

Table 3–4. FASTRAND @ASG Control Statement, Options
(Part 1 of 2)

Option Character	Description
Options for Temporary Files	
T	Specifies that the file is temporary and allows it to have a name the same as that of a catalogued file. The user need not be concerned if a currently catalogued file has the same name. If this option is omitted for temporary files, the executive attempts to find the file in the master file directory. If a find is made, the assignment is made from the master file directory.
Checkpoint/Restart Options for Control of Catalogued FASTRAND Files	
B	Dumps the file as a part of any checkpoint
E	Reloads the file if any other run has referenced the file since the checkpoint
H	Relaods the file only if no other run has referenced the file since checkpoint
M	If a catalogued file by this name exists when reloading, make the reloaded file available to this run as a temporary file.

Table 3–4. FASTRAND @ASG Control Statement, Options (Part 2 of 2)

filename

Specifies the external name of the file to be assigned (see 2.6.1).

type

Specifies that the @ASG control statement is for a FASTRAND-formatted file and identifies the specific device required. If the device type specified is for a catalogued file, it is checked for compatability. If not compatible, the control statement is rejected. Permissable entries for this parameter are:

FCS	– FASTRAND mass storage simulated in unitized channel storage
F4	– FASTRAND mass storage simulated on FH-432 drum
F17	– FASTRAND mass storage simulated on FH-1782 drum
F8	– FASTRAND mass storage simulated on FH-880 drum
F2	– FASTRAND mass storage, Model II and Model III
F	– FASTRAND formatted mass storage, type independent
F14	– FASTRAND mass storage simulated on 8414 disc

FASTRAND mass storage simulated on drum or disc has all the characteristics of a FASTRAND file except for sector padding on write functions.

When space is not available for specified device type, another type is substituted which satisfies the request. The following chart illustrates the order in which requests are satisfied.

Device Requested	Order of Satisfying Request
F	F4, F17, F8, F2 ,F14
F2	F2, F14
F4	F4, F17, F8, F2 ,F14
F8	F8, F2 ,F14
F17	F17, F8, F2 ,F14
F14	F14
FCS	FCS, F4, F17, F8, F2, F14

reserve

An integer specifying the number of granules required by the file (not to exceed 262,143). This parameter should give a reasonable estimate of the space needed to create or update the file. The value used for a file update must include those granules already in use. Files contained within the limits of the reserve are guaranteed creation without the delays involved when the executive must find and allocate the space dynamically. Specification of a reserve aids the executive allocation routines as the space is allocated in contiguous granules, if possible. Omission causes the executive to dynamically allocate the granules as they are required by the file. If the file does not extend to the highest granule reserved, the empty granules after the highest granule referenced are returned to the available status when the file is freed. For catalogued files, the reserve value is placed in the master file directory for future file updates. This parameter is ignored for catalogued read-only files, however, for write-enabled files, the recorded value is overridden and replaced by the value given in this parameter.

granule

Specifies granule size. It may be

TRK – One track (64 sectors)

POS – One position (64 tracks)

If omitted, TRK is assumed. If the file is currently catalogued, this parameter is ignored. The granularity is recorded in the master file directory. For most efficient use of mass storage, all program files should be allocated as TRK granularity because POS granularity creates unused space in files (64 contiguous tracks assigned for POS).

maximum

Specifies the maximum allowable length (in granules) of the file. Permissable values are as for the reserve parameter. When specified, this parameter overrides the system standard maximum specified at systems generation. If omitted, the reserve parameter value or system standard is used, whichever is larger.

If a maximum was supplied when the file was catalogued, its value and the number of granules currently in use are recorded in the master file directory and used whenever the file is referenced. If a maximum is supplied on the referencing of an @ASG control statement, it is used and recorded in the master file directory and it replaces the previous maximum.

This parameter is used to indicate that the run is to be terminated if the length of the file exceeds the number of granules specified. It is used primarily to ensure that a run-away-file situation does not occur during debugging. However, it may also be used to override the system standard for all files.

packids — Specifies the removable disc packs required for the file. Packids consist of from one to six characters. The packids for catalogued files are recorded in the master file directory. Packid is applicable only to removable discs. If omitted, fixed disc is assumed.

Note that many jobs may specify the same set of removable disc packs for unique files.

Description:

The device type of a FASTRAND-formatted file can be changed to a new type when extending a file. To make the change, the file must be reassigned as it was previously assigned, but with a different equipment code (device type). For example:

```
@ASG,C   FILEA,F
(user program writes 100 tracks)

@ASG,C   FILEA,F2
(user program writes 200 tracks)

@FREE FILEA
```

In the given example, the device type was changed from F to F2 after 100 tracks were written. The end result was 100 tracks on type F (drum if available) and 200 tracks on type F2 (FASTRAND mass storage).

The following rules apply:

(1) The file must be currently assigned to the run when the @ASG control statement with the new device type is submitted.

(2) If space is not available on the new device type, allocation occurs on a slower device providing space is available.

(3) The new device type is used on the first occurrence of additional space acquisition. The switch is allowed for both track (TRK) and position (POS) granularity.

(4) There is no restriction on the number of times a file can be switched to a different device.

Examples:

```
   LABEL         OPERATION                 OPERAND               COMMENTS
  1         10            20          30            40            50
1.@ASG,CR       FILEA,F/5
2.@ASG,DA       FILEB/A2294B
3.@ASG,T        FILEC,F/4/POS/5
4.@ASG,XA       FILED,/6//8
```

1. If the run terminates normally or a @FREE control statement for FILEA is processed, FILEA is catalogued as a read-only file. Five tracks are assigned initially and the system-maximum size is assumed, as no maximum was specified.

2. FILEB is currently catalogued and is to be decatalogued if the run terminates normally. The key A2294B is required to read the file.

3. FILEC is a temporary file requiring four FASTRAND positions to be reserved initially. Termination is to occur if more than five positions are required.

4. FILED is currently catalogued and this run is to have exclusive use of the file for updating. A reserve of six tracks is specified, and the run is to be terminated if more than eight tracks are used.

3.7.1.2. MAGNETIC TAPE ASSIGNMENT

Purpose:

Assigns a magnetic tape file to a run.

All parameters on the @ASG control statement are optional except @, ASG, and filename.

Format:

```
@label:ASG,options  filename,type/units/log/noise/MSA-trans/unit-trans/format,;
                    reel-1/reel-2 . . . /reel-n,expiration-period
```

Parameters:

options — See Table 3–5.

The A,C,D,G,K,P,Q,R,T,U,W, and Y options have the same meaning as on the FASTRAND @ASG control statement (see 3.7.1.1). The remaining options control the mode in which the file is recorded and read, and tape labeling.

In the absence of overriding mode options on seven-track tape assignments, the H and O options are assumed. For these assignments, mode option V is invalid.

For nine-track tape assignments, excluding UNISERVO 12/16 nine-track assignments, the H and O options are assumed with B,E,I,L,M, and V mode options being invalid.

For UNISERVO 12/16 nine-track assignments, the V and O options are assumed. Mode options B,E,I,L, and M are considered invalid.

filename — The function and use of this parameter is the same as that specified for the FASTRAND @ASG control statement (see 3.7.1.2).

Option Character	Description
	Option for Pooling Facilities
S	Retains physical assignment for the file. That is, a @FREE control statement releases the file, but the tape unit is saved for future use by the program.
	Mode Options
B	Binary (translation not required)
E	Even parity (assumed when the I option is specified and translation is performed by software). Not recommended if file manipulation is via UNIVAC-supplied software.
H	High density tape (not available for UNISERVO 12/16 nine-track if the hardware dual density feature does not exist on the unit).
I	Decimal (translation required). The translation of BCD to Fieldata on input and Fieldata to BCD on output is performed by hardware, if available. Otherwise, standard system conversion routines are used for translation. The E option is assumed when software performs translation.

Table 3–5. Magnetic Tape @ASG Control Statement, Options (Part 1 of 2)

Option Character	Description
L	Low density tape (not available for nine-track subsystems)
M	Medium desnity tape (not available for nine-track subsystems)
O	Odd parity (assumed when type parameter specifies nine-track requirement)
V	Density mode of 1600 FPI (UNISERVO 12/16 nine-track subsystems only)
Options For Tape Labeling	
F	Allows the user to assign any previously unassigned tape. Any reel is accepted even though the volume and file header do not exist. If the volume and file headers exist on a tape assigned with the F option, normal label checking is performed.
J	Specifies that the reel loaded must not be a labeled tape. If a label exists on the assigned tape, an error message appears.

Table 3–5. Magnetic Tape @ASG Control Statement, Options (Part 2 of 2)

type

Specifies that the @ASG control statement is for a magnetic tape device and identifies the specific type of unit required. Permissible entries for this parameter are:

T	– tape unit, type independent
C	– UNISERVO IV-C, VI-C, and VIII-C tape units
U	– UNISERVO VI-C and VIII-C tape units
2A	– UNISERVO II-A tape unit
3A	– UNISERVO III-A tape unit
4C	– UNISERVO IV-C tape unit
6C	– UNISERVO VI-C tape unit
8C	– UNISERVO VIII-C tape unit
12	– UNISERVO 12 tape unit
16	– UNISERVO 16 tape unit
12N	– UNISERVO 12 nine-track tape unit
16N	– UNISERVO 16 nine-track tape unit
12D	– UNISERVO 12 dual density nine-track tape unit
16D	– UNISERVO 16 dual density nine-track tape unit

Magnetic tapes are always assigned with the following rules of allocation:

Type	Order of Satisfying Request
T	8C, 8CB, 6C, 6CB, 12, 16, 8C9, 6C9 12N, 16N, 4C, 4CB, 3A, 2A
C	8C, 8CB, 6C, 6CB, 4C, 4CB
U	8C, 8CB, 6C, 6CB
8C	8C, 8CB
6C	6C, 6CB
4C	4C, 4CB
CB	8CB, 6CB, 4CB
UB	8CB, 6CB
U9	8C9, 6C9
12N	12N, 12D
16N	16N, 16D

The following magnetic tape assignments do not have a second choice (12, 16, 12D, 16D, 8CB, 6CB, 4CB, 8C9, 6C9, 3A, 2A).

The use of type T or C is encouraged as it gives the system more freedom in assigning units. When using type T, only those functions and options compatible with all types of units may be specified.

Some installations may not have nine-channel/frame capabilities on all units. In addition, there may not be translation hardware on all tape channels. In order to select this equipment, the character 9 or the character B may be added to the type symbols to indicate nine-channel/frame unit or translate channel, respectively (with the exception of the UNISERVO 12/16 tape units). As an example, if a UNISERVO VIII-C unit with nine-channel capabilities is needed (but not available on all units), the type subfield would contain 8C9. The symbol 6C9 would call for a UNISERVO VI-C nine-channel. The symbol 6C would call for a UNISERVO VI-C channel with the hardware translation feature. The combination of TB and T9 is not allowed.

Software translation is not assumed if a unit with the hardware capability is selected unless the I option is specified or translation is called for by either an @MODE control statement or the set mode function of the magnetic tape handler. See Description for absolute assignment.

units

Specifies the number of tape units required, and may be integers 1 or 2. If omitted or a number other than 1 or 2 is specified, the executive assumes that one unit is required. The number of units assigned is not retained in the master file directory upon cataloguing. See Description for absolute assignment.

log

Assigns a single letter indicating a logical channel. The executive attempts to assign all files with the same letter to the same physical channel and those with different letters to different channels. The letter specified is not placed in the master file directory upon cataloguing. This parameter permits more efficient I/O operation because of the separate channels.

noise — Specifies an integer from 1 to 99 which overrides the standard system noise constant. If omitted, the standard system noise constant is assumed if parameter is omitted.

MSA-trans — (UNISERVO 12/16 only) Specifies the type of translator needed in the MSA. The MSA translator mnemonics are:

EBCDIC	– Fieldata to or from EBCDIC
ASCII	– Fieldata to or from ASCII
XSEBCD	– XS-3 to or from EBCDIC
XSASCI	– XS-3 to or from ASCII
OFF	– turns off translator if assign is from the master file directory and file was catalogued with a translator specification.

unit-trans — (UNISERVO 12/16 only) Specifies the type of translator needed in the control unit (not available for UNISERVO 12/16 nine-track tape units). The control unit translator mnemonics are:

BCD	– EBCDIC to or from BCD
DC	– Three-to eight-bit bytes converted to or from four- to six-bit tape characters.
OFF	– turns off translator if assign is made from the master file directory and the file was catalogued with a translator specification.

format — (UNISERVO 12/16 only) Specifies the data transfer format for the word-to-byte conversion in the MSA. The data transfer mnemonics are:

Q	– quarter word
6	– six-bit packed
8	– eight-bit packed

reels — Specifies the identifier for each tape reel required. Each reel identifier is limited to six characters. Reels are used and catalogued in the order specified. If the file is to be catalogued, all reel identifiers are recorded in the master file directory. If omitted and cataloguing is indicated, the executive directs the operator to mount blank reels on the appropriate tape units to provide reel identifiers for each reel. If additional tape reels are requested by a TSWAP$ request (see 7.2.8), operator is requested to load the required blank reels and provide identifiers for each reel.

For currently catalogued files, the reel parameter is normally void, indicating that the reels listed in the master file director are to be used in the order in which they were created. If reel numbers are supplied, they must be of the set listed in the master file directory, but may be a subset and listed in any order, allowing the user to omit or access them in any order. If an invalid reel number is supplied, the @ASG control statement is not honored. In either case, when the known reels are exhausted and additional reels are requested, blank reels are used and their numbers added to the master file directory. (This is not allowed if the file is catalogued in the read-only state.)

For temporary files, the reel parameter is not specified and the operator is requested to mount blank reels (reel numbers are not required). If reel numbers are given on the @ASG control statement, they are used in the given order. When additional reels are requested, blanks are used, but reel numbers are not requested.

expiration-period — Specifies the number of days that this file is to be retained. The maximum number allowed is 4,095 days.

Description:

The executive normally controls the selection of units for the assignment of tape files. However, the user may direct the tape assignment to a particular subsystem or unit by specifying the absolute subsystem and unit. Absolute subsystem and unit specification is not the recommended procedure for assigning tapes, however, maintenance routines, real time programs, or special hardware requirements may dictate the need for such specifications.

A unit that has been placed in the reserved state by a RV unsolicited console keyin can only be assigned by specifying the absolute subsystem and unit on the assign statement. However, it is not necessary to have a unit in a reserve state in order to assign it absolutely.

For absolute tape assignment, all parameters except type, units, and log retain their meaning as described earlier. The contents of these three parameters are:

type — Contains subsystem in the format Sxxx where xxx is the subsystem number (1 to 127)

units — If a particular unit is required, specifies the unit in the format Uyy where yy is the unit number (0 to 15). If the executive is to choose the units, this parameter has the same meaning as described earlier.

log — If a second unit is required, specifies the unit in the format Uzz where zz is the unit number (0 to 15). In all other cases of absolute assignment, the contents of this parameter are ignored.

Examples:

```
     LABEL       Δ   OPERATION    Δ                OPERAND       Δ               COMMENTS
   1           10              20                30             40               50
1. @ASG,A         FILEX
2. @ASG,T         FILEA,///36
3. @ASG,TEL       FILEB,6C/2,N432
4. @ASG,CR        FILEC,8C9
5. @ASG,DA        FILED/4B96,8C//A,N212
6. @ASG,U         FILEE/492671/RA1234,8C/2,707/708/709/710
7. @ASG,T         FILEF,T,$CRTCH
8. @ASG,T         FILEY,S12/2
9. @ASG,T         FILEY,S12/U6,29416
10.@ASG,T         FILEY,S12/U6/U4
```

1. FILEX is catalogued and all necessary options, facility requirements, and reel numbers are taken from the master file directory. The project-id of the current run is used as a qualifier.

2. FILEA is a temporary file requiring one unit selected by the executive; one or more blank reels are used. The noise constant is to be set to 36 characters.

3. FILEB is a temporary file requiring two UNISERVO VIII-C tape units. It is recorded in even parity and low density. Reel number N432 is specified.

4. FILEC is to be catalogued in the read-only mode if the run terminates normally. One UNISERVO VIII-C tape unit with nine-channel capabilities is required.

5. FILED is currently catalogued but is to be released if the run terminates normally. A key of 4896 is required to read this file. The UNISERVO VIII-C tape unit is to be on logical channel A and reel N212 is to be used.

6. FILEE is to be catalogued. It requires two UNISERVO VIII-C tape units on any channel. Reels 707 through 710 are to be used. The file is locked by the specified read (492671) and write (RA 1234) keys.

7. FILEF is a temporary file and the symbol SCRTCH is used as a reel number.

8. FILEY is assigned two units of the systems choosing from subsystem 12.

9. FILEY is assigned unit 6 on subsystem 12; reel 29416 is to be used.

10. Units 6 and 4 from subsystem 12 are assigned to the file.

3.7.1.3. WORD ADDRESSABLE DRUM ASSIGNMENT

3.7.1.3.1. NORMAL ASSIGNMENT

Purpose:

Assigns word addressable mass storage and simulated word addressable mass storage to a particular run.

All parameters are optional on the @ASG control statement except filename.

Format:

```
@label:ASG,options    filename,type/reserve/granule/maximum,packid-1/packid-2/. . ./packid-n
```

Parameters:

With the exception of the following differences, the parameters of this statement are basically the same as those for the FASTRAND @ASG control statement (see 3.7.1.1).

options — Same as 3.7.1.1. except no distinction is made between file types except in the conversion of logical to physical addresses. Word-addressable files cannot be used as program files.

filename — Specifies the external name of the file to be assigned.

type — Specifies that the @ASG control statement applies to word addressable drum format and names the specific type of recording equipment to be used. Permissible parameters are:

D	– Word-addressable storage, type independent
D4	– Word-addressable storage, FH-432 drum
D8	– Word-addressable storage, FH-880 drum
D17	– Word-addressable storage, FH-1782 drum
DCS	– Word-addressable storage, unitized channel storage
D14	– Word-addressable storage, simulated on 8414 disc

Use of the D entry is recommended since it allows the executive freedom in allocating file space.

The use of the D14 entry forces the executive to simulate word addressability which introduces additional overhead each time the file is accessed.

reserve — Same as 3.7.1.1

granule — Same as 3.7.1.1.

maximum — Same as 3.7.1.1. except the entry is the number of words needed for the file.

packids — Same as 3.7.1.1.

Description:

The mass storage allocation routine attempts to satisfy word addressable drum requests in the same manner as it satisfies FASTRAND-formatted requests. Allocations are made on contiguous granules if possible, thereby permitting dynamic expansion of a word-addressable file. Although dynamic expansion is available, the user is not allowed to dynamically contract a word-addressable file (read and release, and release I/O functions are not allowed) (see 6.1.1).

Examples:

```
   LABEL      Δ  OPERATION    Δ              OPERAND    Δ            COMMENTS
1          10             20            30            40            50
1. @ASG,CR    FILEA,D4/275/TRK
2. @ASG,T     FILEB,D/2217/TRK
3. @ASG,UP    FILEC,D17/17560/TRK
```

1. FILEA is to be catalogued in the read-only mode on the FH-432 drum upon normal termination of the run or when a @FREE control statement (see 3.7.4) is encountered. Initial length of FILEA is one track. The standard system maximum is assumed for maximum subfield.

2. FILEB is a temporary file requiring an initial reserve of two tracks and residing on a drum selected by the executive.

3. FILEC is to be catalogued on the FH-1782 drum as a public file at run termination (regardless of the manner of termination), or when the @FREE control statement is encountered. Initial length of the file is ten tracks. Standard system maximum is assumed.

3.7.1.3.2. WHOLE UNIT ASSIGNMENT

A mass storage unit in the reserve state (see 7.2.3) may be assigned as a facility to the run. Only the operator can place units in the reserved state.

The following format is used to assign a word-addressable, unit-granular file:

```
@label:ASG,options    filename, type, unit-1/unit-2/. . . /unit-n
```

All parameters are interpreted identically to the normal @ASG control statement except type. The valid entries for the type parameter are:

D4U – FH-432 drum

D8U – FH-880 drum

D7U – FH-1782 drum

The equipment on the requested subsystem/unit need not be the same as the mnemonic. For example, a type of D7U may have FH-432 and FH-880 units requested.

Any number or type of allowable units may be requested. The unit parameters (unit-1, unit-2,) have the format nnnUnn where nnn is the decimal subsystem number; the U is the separation character; and nn is the decimal unit number. All six characters need not be supplied, but the separation character must have decimal numbers on both sides.

For example, the request @ASG,T FN,D8U,3U1/12U5/3U3 assigns the file FN as a word-addressable, unit-granular file with the three requested units.

3.7.1.4. ARBITRARY DEVICE ASSIGNMENT

Purpose:

Assigns all devices except FASTRAND-formatted mass storage, drum, and magnetic tape units. Used primarily for the assignment of special I/O devices and communications equipment.

All parameters on the @ASG control statement are required.

Format:

```
@label:ASG   filename,type
```

Parameters:

filename

Specifies the external name of the file.

type

Specifies:

(1) The mnemonic definition of a class of devices; the executive selects the specific unit if more than one unit exists

(2) absolute subsystem; the executive selects specific unit

(3) absolute subsystem/unit

Mnemonic definitions of standard devices other than magnetic tape units or mass storage devices are:

CRD	– Card reader system
PTP	– Paper tape subsystem
PRT	– Printing device
HSP	– High speed printer
1004	– 1004 reader/printer/punch

Mnemonic defintiions used to assign communications devices must agree with those defined at systems generation. For example, communications devices are defined, at system generation, as units under a group class identity called the LT group.

This group identity, when specified in the type parameter, causes the executive to select that LT group to satisfy the request.

For absolute subsystem assignment, the type parameter contains the subsystem number (1 to 127) in the format:

```
Sxxx
```

For absolute unit assignment, the type parameter contains both the subsystem number and the unit number in the format:

```
Sxxx/Uyy
```

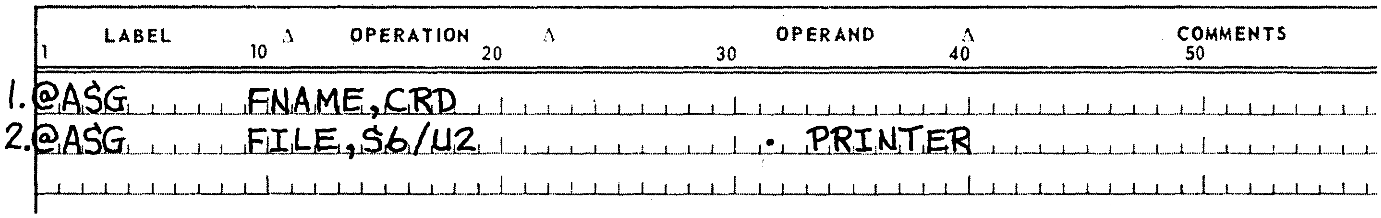
If absolute subsystem and unit are used for communications devices, the unit specified is assumed to be the input rather than output or dial (see Section 12).

A disc can be assigned for use with the arbitrary device handler by using the absolute subsystem and unit form of the arbitrary device @ASG control statement. The format is:

```
@ASG,options    filename,Sxxx/Uyy,packid
```

The requested unit must be in the reserved state to satisfy this type request (units can only be put in the reserved state by the operator. The packid parameter is used in a load messsage to instruct the operator to mount a specific pack on a specific unit.

Examples:



```
   LABEL      Δ   OPERATION    Δ              OPERAND     Δ          COMMENTS
1                10            20           30            40           50
1.@ASG        FNAME,CRD
2.@ASG        FILE,S6/U2                      . PRINTER
```

1. The card subsystem is assigned, and the specific device is selected by the executive.

2. The printing device identified as unit 2 of subsystem 6 is assigned to the run.

3.7.2. TAPE UNIT MODE CONTROL (@MODE)

Purpose:

Changes the mode settings initially set by a previous tape @ASG control statement. The file must be currently assigned to the run.

All parameters on the @MODE control statement are optional except filename.

Format:

```
@label:MODE,options    filename,noise/MSA-trans/unit-trans/format
```

Parameters:

options	Same as mode options in Table 3–5.
filename	Specifies external name of tape file to which mode change applies.
noise	Same as 3.7.1.2.
MSA-trans	Same as 3.7.1.2.
unit-trans	Same as 3.7.1.2.
format	Sames as 3.7.1.2.

Description:

With the @MODE control statement, options (modes) are never assumed in the absence of others. The specified options are not placed in the catalogue, since they apply only to the current assignment.

See the CSF$ request (4.8.1) for the linkage used to call this control statement from within a user program.

Example:

```
    LABEL      Δ    OPERATION     Δ             OPERAND      Δ            COMMENTS
1          10                 20            30             40              50
@MODE,BEL      FILEY,30
```

The initial mode settings assigned to FILEY are overridden by the B, E, and L options specified in the @MODE control statement. A noise level of 30 is also specified for FILEY.

3.7.3. INDEPENDENT CATALOGUING OF FILES (@CAT)

Purpose:

The @CAT control statement is used to catalogue one or more files without having them assigned to the run. This may be the case when building the initial master file directory or when a previously prepared tape file is to be catalogued. The file is catalogued but is not assigned to the run and no facilities are assigned.

Formats:

The @CAT control statement has two formats: one for cataloguing tape files and one for cataloguing FASTRAND files. Both formats are presented.

All parameters on the @CAT control statement are optional except filename.

Format for cataloguing tape files:

```
@label:CAT,options    filename,type/noise/MSA-trans/unit-trans/format,reel-1/reel-2/ . . . /reel-n
```

Parameters:

options	See Table 3–6.
filename	Specifies the external name of the file to be catalogued.
type	Same as 3.7.1.2. If omitted, the executive assumes that the request is for a FASTRAND-formatted file.

Options Character	Description
B	Binary code (no translation required)
E	Even parity (if not specified, standard system value is used)
G	Indicates that when this file is catalogued, it is to be guarded against having its read key, write key, and similar protection overridden by privileged runs. (A privileged run may be initiated by the site manager for such purposes as producing a backup copy of the file on tape to use in catalogued file recovery.)
H	High density (if no tape density is specified, standard system value is used)
I	Decimal code (translation required). Conversion is performed by hardware, if available. Otherwise, standard software conversion routines are used to translate BCD to Fieldata (input) and Fieldata to BCD (output). When software conversion routines are used, the E option is assumed.
L	Low density (if no tape density is specified, standard system value is used)
M	Medium density (if no tape density is specified, standard system value is used)
O	Odd parity (if not specified, standard system value is used)
P	Catalogue file as public file
R	Place in read-only state
V	Indicates that when this file is catalogued, its text is not to be unloaded to tape at any time.
W	Place in write-only state

Table 3–6. @CAT Control Statement, Options

noise	Same as 3.7.1.2.
MSA-trans	Same as 3.7.1.2.
unit-trans	Same as 3.7.1.2.
format	Same as 3.7.1.2.
reels	Same as 3.7.1.2.

Format for cataloguing FASTRAND-formatted or word-addressable files:

```
@label:CAT,options   filename,type/reserve/granule/maximum,packid-1/packid-2/. . . /packid-n
```

Parameters:

options	The G,P,R, and W options are the only valid options (see Table 3–6).

filename	Same as 3.7.1.1 and 3.7.1.3.

type	Same as 3.7.1.1 or 3.7.1.3, depending on file format. If omitted, the standard system entry (F) is assumed.

reserve	Same as 3.7.1.1 and 3.7.1.3

granule	Same as 3.7.1.1 and 3.7.1.3

maximum	Same as 3.7.1.1 and 3.7.1.3

packids	Same as 3.7.1.1 and 3.7.1.3

See the CSF$ request (4.8.1) for the linkage used to call this control statement from within a user program.

Examples:

```
   LABEL      Δ   OPERATION   Δ            OPERAND     Δ            COMMENTS
 1          10              20           30           40           50
1. @CAT,P    FILEY//WLOCK,T,N247/N248
2. @CAT,W    FILEX//WLOCK,F/7/TRK/10
3. @CAT,R    WADDRUM,DC$/1700/TRK/2000
```

1. FILEY is to be catalogued as a public file with the write-key WLOCK. The device type is specified as tape with the specific tape device selection made by the executive. The standard system noise level is used as this parameter has been omitted from the statement. The file is recorded on the two reels identified as N247 and N248.

2. File FILEX is to be catalogued in the write-only mode. The key WLOCK is required to write in the file. The recording device is specified as FASTRAND with the specific device selected by the executive. At assign time, seven FASTRAND tracks are initially reserved for the file. The run is terminated if more than 10 tracks are used.

3. File WADDRUM is to be catalogued in the read-only state. The recording device is specified as word-addressable format, unitized channel storage. 1700 words are reserved for the file and the run is terminated if the file exceeds 2000 words.

3.7.4. RELEASING FILES AND PERIPHERAL DEVICES (@FREE)

Purpose:

Deassigns files and releases their facilities, reels, and exclusive use areas assigned to the run. Files and facilities should be released the moment they are not needed. If they are not released by means of a @FREE control statement, they are retained until run termination.

All parameters on the @FREE control statement are optional.

Format:

```
@label:FREE,options   filename
```

Parameters:

options	See Table 3–7.
filename	Specifies the internal or external name of the file to be deassigned. This entry must agree with the filename specified by the @ASG control statement (see 3.7.1) or equated to the file by the @USE control statement (see 3.7.5).

Option Character	Description
A	Releases only the specified internal name relationship to the file.
B	Releases only the specified internal name associated with the file unless it is only the internal name attachment, in which case the entire file is freed.
D	Deassigns a catalogued file. The file must have been assigned with the correct keys.
E	Sets the first file header in the tape label back to skeleton format to logically set it to a blank tape.
I	Inhibits final cataloguing action if the file was assigned by an @ASG control statement with a C or U option.
R	Releases the file assigned but retains the internal name relationships to the filename and the F-cycle.
S	Frees the file but retains the physical tape unit.
X	Releases exclusive use of the file. The file, however, is not deassigned from the run.

Table 3–7. @FREE Control Statement, Options

Description:

A file that is deassigned by a @FREE control statement can no longer be referenced by the run. It can, of course, be reestablished by an @ASG control statement provided its facility requirements can be met.

The actions taken by the system when a file is deassigned by a @FREE statement (and the S option was not specified) are discussed below.

For a temporary file (not catalogued or to be catalogued):

FASTRAND	– The FASTRAND area is made available as file space for other runs.
Drum	– Same as FASTRAND mass storage.
Disc	– Same as FASTRAND mass storage.
Tape	– Units are released for use by other runs.

Other equipment (communications and so forth) – The device is released for use by other runs. Always temporary.

For a file being catalogued (C or U option on @ASG control statement):

FASTRAND – Catalogue entry is made in the master file directory and FASTRAND area containing the file is held. The file can now be referenced by other runs.

Disc – Same as FASTRAND mass storage.

Tape – Catalogue entry containing reel numbers is made; units are released for other runs.

For a file being de-catalogued (D or K option on @ASG control statement):

FASTRAND – Same as for a temporary file except that the file area is not released until all runs currently using the file have also finished. It is no longer available for assignment.

Disc – Same as FASTRAND mass storage.

Tape – Units are released for use by other runs. The file is no longer available for assignment.

See the CSF$ request (4.8.1) for the linkage used to call this control statement from within a user program.

Examples:

```
   LABEL      OPERATION           OPERAND           COMMENTS
1         10            20          30           40          50
  @ASG,C     FILEA,F/S
  @ASG,T     FILEB,F/4/POS/5
  @ASG,X     FILEC,/6//8
  @ASG,CR    FILEX,8C//A,R121
  @ASG,T     FILEY,8C//B,R212
  @ASG,D     FILEZ,8C//C,R111


1.@FREE,I    FILEA
2.@FREE,C    FILEB
3.@FREE,X    FILEC
4.@FREE,S    FILEX
5.@FREE,R    FILEY
6.@FREE      FILEZ
```

1. FILEA is to be catalogued upon normal run termination or by execution of the @FREE control statement as specified by the C option on the @ASG control statement. The @FREE control statement, however, inhibits cataloguing of FILEA because of the I option.

2. FILEB is deassigned and all filenames are released (with no special considerations).

3. Exclusive use of FILEC (currently catalogued FASTRAND file designated as exclusive use for current run) is released. The exclusive use obtained by the @ASG control statement is released by the X option on the @FREE control statement.

4. FILEX (designated to be catalogued at normal termination of the run or by execution of the @FREE control statement) is deassigned. The UNISERVO VIII–C tape unit is retained for run use (S option). The reel number (R121) is recorded in the master file directory.

5. Temporary tape file FILEY is deassigned. The internal filename relationship to the file is retained (R option specified). The F–cycle is also retained.

6. FILEZ is decatalogued from the master file directory. The UNISERVO VIII–C tape unit is not retained for run use (S option).

3.7.5. ATTACHING INTERNAL FILENAMES (@USE)

Purpose:

Equates filenames so that any particular file can be referenced by more than one filename. This need arises when:

(1) run construction can be simplified by using a shorter internal filename in place of a long external filename

(2) identical filenames must be differentiated

(3) internally programmed filenames must be connected to external filenames

The information presented in 2.6.2 on file naming is a prerequisite for understanding internal and external filename relationships.

The @USE control statement has two formats: equating internal filenames to external filenames (Format 1), and equating internal filenames to internal filenames (Format 2).

All parameters are *required* except label.

Format 1:

```
@label:USE   internal-filename,external-filename
```

Format 2:

```
@label:USE   internal-filename,internal-filename
```

Parameters:

internal-filename — Specifies the name by which the file can be referenced within the run after the @USE control statement is encountered in the control stream.

external-filename — Specifies the full external name of the file. The external name always takes the form qualifier*filename.

Description:

All internal filenames equated to an external filename are listed and maintained for the run. Once equated, the user can reference the file by its internal or external filename from within a program or the run stream. If a conflict of filenames exists, it is the user's responsibility to attach an internal name to the file (with the conflicting external name before any references to that file are attempted). The internal filename list is always searched first on an I/O reference.

If a no-find condition occurs on the internal names, the external filename list is searched.

Multiple internal filenames can be attached to an external filename.

See the CSF$ request (4.8.1) for the linkage used to call this control statement from within a user program.

```
   LABEL        Δ    OPERATION    Δ              OPERAND      Δ             COMMENTS
1           10              20              30              40              50
1.@USE   FILEB,COMPANY*PAYROLL
2.@USE   COST,FILEB
```

1. The internal filename FILEB is equated to external filename COMPANY*PAYROLL. The file can now be referenced for I/O by either the internal name FILEB or the external name PAYROLL.

2. The internal name COST is now a third association to the file for I/O.

3.7.6. SPECIFYING FILENAME QUALIFIER (@QUAL)

Purpose:

Specifies a standard filename qualifier for implied usage on succeeding control statements involving filenames.

All parameters in the @QUAL control statement are *required* except label.

Format:

@label:QUAL   qualifier

Parameters:

qualifier	Specifies name extension used to qualify subsequent filenames which are immediately preceded by an asterisk (*).

Description:

Any number of @QUAL control statements can appear throughout the control stream. Each time a @QUAL control statement is encountered, the new qualifier overrides the qualifier specified in the previous @QUAL control statement.

See the CSF$ request (4.8.1) for the linkage used to call this control statement from within a user program.

Examples:

```
   LABEL        Δ    OPERATION    Δ              OPERAND      Δ             COMMENTS
1           10              20              30              40              50
1.@QUAL   1STQUAL
  @FOR    *FILEA.DO/ABC
2.@QUAL   2NDQUAL
  @FREE   *FILEA
```

1. The @QUAL control statement provided in this example specifies that all subsequent filename references, which have a preceding asterisk, be interpreted as having the qualifier 1STQUAL. For example, the @FOR control statement element shown in the example is interpreted as:

   @FOR   1STQUAL*FILEA.DO/ABC

2. This @QUAL control statement overrides the @QUAL control statement in example 1. Therefore, subsequent filename references which have preceding asterisk will have the implied qualifier 2NDQUAL. For example, the @FREE control statement is interpreted as:

```
@FREE   2NDQUAL*FILEA
```

3.8. DATA PREPARATION CONTROL STATEMENTS

3.8.1. DIRECT CREATION OF CARD IMAGE FILES (@FILE)

Purpose:

The @FILE control statement creates SDF-formatted mass storage or magnetic tape files while the input symbiont is reading the run stream. Creating a file in this manner rather than by means of the ELT or DATA processors reduces overhead since it is accomplished at input time rather than at execution time. The data is handled only once in creating the file rather than being read again from auxiliary storage with the run stream and then transferred to the file by the DATA or ELT processors at execution time. The file into which the images are placed must be FASTRAND formatted or magnetic tape.

Format:

For each storage device, the format of the @FILE control statement is identical to the @ASG control statement for that device (except that the label field is not used).

If the device type specifies FASTRAND format mass storage the format is:

```
@FILE,options   filename, type/reserve/granule/maximum, packid-1/packid-2/...packid-n
```

NOTE:
The packid's are only used if the type field specifies removable disc.

If the device type is magnetic tape, the format is:

```
@FILE,options   filename(F-cycle),type/units/log/noise/MSA-trans/;
                unit-trans/format,reel-1/reel-2 . . . /reel-n expiration period
```

Parameters:

options   When device type is mass storage, the options are:

A	– Already catalogued
G	– Guard modes
P	– Catalogue as public
R	– Catalogue as read only
W	– Catalogue as write only
C or U	– Catalogue (assumed if not present)
T	– Temporary (used on tape files only)

When device type is magnetic tape, the options are:

B	– Set no translate mode
E	– Set even parity
F	– Check label only if present
H	– 800 FPI density
I	– Set translate mode
J	– Label not present
L	– 200 FPI density
M	– 556 FPI density
O	– Odd parity
V	– 1600 FPI density

Description:

For a more complete explanation of these options and the remainder of the control statement, see the @ASG statement for FASTRAND mass storage or magnetic tape (see 3.7.1.1 and 3.7.1.2).

@FILE control statement processing is terminated upon encountering an @ENDF control statement, a @FIN control statement, or another @FILE control statement. Data images and all control statements except @COL and its accompanying end sentinel are placed into the created file (the @COL control statement and sentinel are processed immediately, and the file is marked when the mode switch is mode).

If the device type is magnetic tape and a @FILE control statement is encountered while processing a previous @FILE control statement for the same filename, then the current file is closed, and an EOF mark is written and the second file follows immediately on the same reel.

Since the file created by the @FILE control statement is not available until it is closed (by an @ENDF, @FIN or second @FILE control statement), the user should physically structure his input run streams so as to not access the file until it is available, that is, calls on this file by any language processor or user requests should physically follow the @FILE, @ENDF sequence which created it.

3.8.2. TERMINATING THE FILE MODE (@ENDF)

Purpose:

Marks the end of the images for a file created by the @FILE control statement.

Format:

```
@ENDF
```

Description:

@ENDF control statements cannot have labels and cannot be continued.

3.9. DYNAMIC RUN STREAM MODIFICATION

3.9.1. DYNAMIC RUN STREAM EXPANSION (@ADD)

Purpose:

Provides a means of inserting images into the run streams from any file currently assigned to the user or any catalogued file, provided that it is a FASTRAND-formatted file in SDF format, or from any source element of a program file created by such means as the:

- DATA processor
- ED processor
- ELT processor
- user program

Images being added may be data or any control statement normally allowed in a run stream with the exception of those control statements which are acted upon at input time, such as:

@RUN

@COL

@FIN

@FILE

@ENDF

All parameters on the @ADD control statement are required.

Format:

```
@label:ADD,options    name
```

Parameters:

options — The options are:

D — Allows the insertion of files or elements when operating under the DATA or ELT,D processors.

E — Return control at the EOF address of the READ$ request as if an @EOF control statement had been encountered when the end of the added file or element is reached (see 5.2.1).

P — The @ADD control statement is to be printed in the program listing.

name — Names the file or element to be added (see 2.6). If a filename is intended, the filename must be followed by a period, otherwise it is interpreted as an element name.

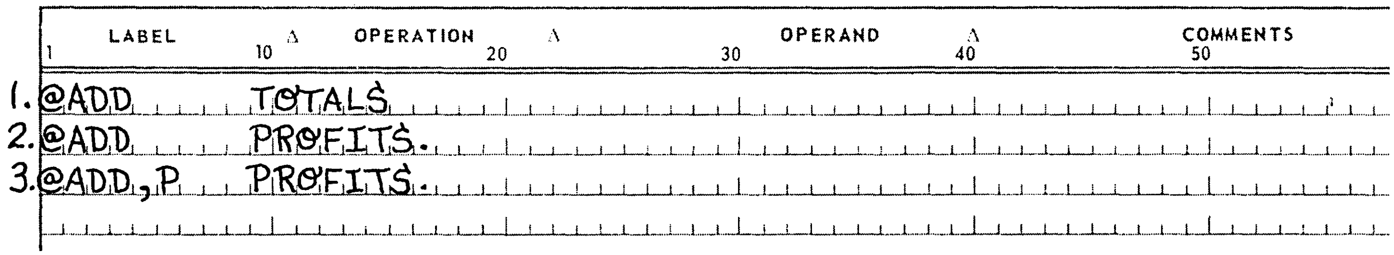
Description:

When the @ADD control statement is encountered, the first image of the file or element being added replaces the @ADD control statement image. All subsequent run stream images are taken from the file or element being added until either end-of-file or end-of-element is encountered. Subsequent images are obtained from the file in which the @ADD control statement was encountered.

@ADD control statements may be nested to the level specified at systems generation. However, a given file or element may not be referenced twice in the same nest. When this occurs, or when a nonexistent file or element is referenced, the run is placed in the error mode and processing continues.

This control statement is a valuable tool for remote users (batch or demand) because control statements or data need be submitted only once but may be used in many subsequent runs. The prestored, partial control streams can be corrected prior to their addition by placing correction lines after an @ELT,D or an @DATA control statement.

Examples:



```
     LABEL      Δ  OPERATION   Δ           OPERAND     Δ         COMMENTS
   1            10            20          30          40          50
1. @ADD         TOTALS
2. @ADD         PROFITS.
3. @ADD,P       PROFITS.
```

1. The images in file TPF$, element TOTALS replace the @ADD control statement.

2. The images in file PROFITS replace the @ADD control statement.

3. The images in file PROFITS replace the @ADD control statement. The @ADD control statement is to appear in the program listing (P option).

3.9.2. CONDITIONAL STATEMENTS

Conditional statements are those executive control statements which are used for the dynamic adjustment of the control stream while it is being executed. The conditional control statements provided are:

@SETC control statement (see 3.9.4.1)

@TEST control statement (see 3.9.4.2)

@JUMP control statement (see 3.9.4.3)

Through the use of these control statements, the user can set values in a condition word that is maintained for each run, test that value, and depending upon the results of that test, skip a portion of the control stream, or direct an individual task of the run to vary its execution. The condition word can also be accessed or altered by either the executive or by any of the user programs within the run. The outstanding feature of this conditional network is that it allows a given run stream to produce many different results with only minor modifications to the stream.

3.9.3. STATEMENT LABELING

Every control statement, including those registered by a CLIST$ executive request (see 5.5) may have a label specified. The label provides the means by which portions of the run stream can be skipped with control passed to the statement having a particular label. Note that labels contained on the @COL, @END, @ENDF, and @FILE control statements cannot be used for passing control since these control statements are not entered in the run stream. Control statements have only one label; however, additional labels can be attached to a statement by means of the use of label statements.

A label statement is any control statement containing a label but no other parameters; that is, just the recognition character (@), the label, and the colon(:).

As an example, the @XQT control statement, which follows, can be referenced by any of the labels TAG, MARK, or LAST, which are attached to it by label statements. As each label statement is encountered, no operation is performed and run stream processing continues until a control statement with a command is found.

```
@TAG:

@MARK:

@LAST:XQT   PROGX
```

If the same label appears more than once within a run, the first forward occurrence is taken as the proper label.

3.9.4. CONDITION WORD

The condition word format is:

T1	T2	T3
error-condition-bits	0-or- value-set-by-@SETC	0-or- value-set-by-ER-SETC$

The condition word is divided functionally into thirds, as follows:

- T1 is set by the excutive to indicate various error conditions and states with the following bit settings:

  Bit 30 — Inhibit run termination when a program error terminates (set by @SETC,I and cleared by @SETC,A-see 3.9.4.1.

  26 — Most recent activity termination was an ABORT$ (not EABT$).

  25 — Most recent activity termination was an error termination.

  24 — One or more previous activity termination of the current task (previous task if between executions) was an error termination (see 4.3.2).

For example, a value of 4 in S2, between tasks, means that the last activity of the previous task did an ABORT$ request and that all other activities of that task (if any) terminated normally. Note that bits 26 and 25 cannot both be set.

- T2 is set by the @SETC control statement (see 3.9.4.1). It may also be set by means of the set parameter on the @START control statement (see 3.4.3).

- T3 is set by means of the SETC$ request (see 4.4.1).

While the entire condition word may be examined, either in the run stream (@TEST-see 3.9.4.2) or by an executing task (COND$-see 4.4.2), alteration is limited to individual thirds, where T1 can be modified only by the executive; T2 only by a control statement, and T3 only by an executive request.

The condition word is set to all zeros at the start of a run, unless the run was started by a @START control statement with a set parameter, in which case T2 initially contains the value of that parameter.

The run stream path may be varied using the condition word in combination with the @SETC, @TEST, and @JUMP control statements (see 3.9.4.1, 3.9.4.2, and 3.9.4.3, respectively).

3.9.4.1. CONDITION WORD CONTROL (@SETC CONTROL STATEMENT)

Purpose:

Stores (set) a value in the T2 portion of the condition word.

All parameters are optional on the @SETC control statement except value.

Format:

```
@label:SETC,options   value/j
```

Parameters:

options — The options are:

A – Clears bit 30 of the condition word which allows a normal ERR$ termination of this run. For example, if an error termination occurs, the run is terminated after processing @PMD and conditional control statements. This option is in effect when the run is initiated (opened).

I – Sets bit 30 of the condition word which inhibits run termination after a program error terminates. Normal processing continues after any error terminations that occur while this option is in effect.

The I option should not be specified unless the user is willing to assume CPU costs beyond time of error detection. This option permits runs with independent tasks or tasks with nonfatal errors to continue. It also permits tests in which error conditions are expected to be encountered, to continue.

value — Specifies a positive octal value not exceeding four digits in length to be entered into the designated portion of the condition word. Value is right-justified, zero filled prior to a partial word store into the designated portion. If the magnitude of value exceeds the capacity of the designated portion, truncation occurs.

j — Designates portion of condition word into which value parameter is to be stored. Permissible entries are: T2, S3, or S4. If omitted, T2 is assumed.

Examples:

```
       LABEL       Δ    OPERATION     Δ          OPERAND      Δ            COMMENTS
   1              10               20          30            40              50
1. @SETC          6
2. @SETC          10/S3
3. @SETC,I        4/S4
```

1. Loads 6_8 into T2 of condition word. (T2 is assumed since j parameter is omitted.)

2. Loads 10_8 into S3 of condition word. The value is treated as octal even when the leading zero is omitted.

3. Loads 4_8 into S4 of condition word.

3.9.4.2. CONDITION WORD TESTING (@TEST)

Purpose:

Tests the value of the condition word to select the particular control statements to be executed or skipped.

All parameters on the @TEST control statement are optional except the first occurrence of f and value.

Format:

```
@label:TEST   f/value/j,f/value/j . . .
```

Parameters:

f

Specifies the type of comparison to be made. If the test is met, the next control statement is skipped. If test is not met, the next control statement is executed. Permissible entries are:

TE	–	Test for equal
TNE	–	Test for not equal
TG	–	Test for greater than
TLE	–	Test for less than or equal

value

Specifies a positive, octal value not exceeding 12 digits to be compared with that portion of the condition word specified by the j parameter.

j

Specifies the portion of condition word to be tested. Permissible entries are: U, H1, H2, T1 through T3, and S1 through S6. If omitted, T2 is assumed.

Description:

When more than one test is to be made, the control statement is scanned until a test is met or all parameters are exhausted. When a test is met, the control statement immediately following the @TEST control statement is skipped.

Examples:

	LABEL 1	OPERATION 10 20	OPERAND 30 40	COMMENTS 50
1.	@TEST	TG/6		
2.	@TEST	TE/6,/4		
3.	@TEST	TG/10/H2		
4.	@TEST	TLE/4/T2		

1. S3 of the condition word is tested to see if it is equal to a value of 6_8, if equal, the next control statement in stream is skipped. If unequal, the next control statement is executed.

2. T2 of the condition word is tested for two conditions: greater than 6_8 or equal to 4_8. If either condition is met, the next control statement in the stream is skipped. If neither condition is met, the next control statement in the stream is executed.

3. H2 of the condition word is tested for a greater than 10_8 condition. If the condition is met, the next control statement in stream is skipped; otherwise, the next control statement is executed. The value is treated as even when the leading zero is omitted.

4. T2 portion of the condition word is tested to see if it is less than or equal to 4_8. If so, the next control statement in the stream is skipped; otherwise, it is executed.

3.9.4.3. BRANCHING FROM WITHIN RUN STREAM (@JUMP)

Purpose:

Advances control to the specified labeled statement within the control stream.

All parameters in the @JUMP control statement are *required* except label.

Format:

```
@label:JUMP    label
```

Parameters:

label	Specifies the label or name attached to a subsequent control statement to which control is passed. This parameter may instead be a decimal value indicating the number of labeled control statements to be advanced. Those control statements which cannot have labels must not be included in the count. A numeric value of 0 as a parameter is not permitted.

Description:

The @JUMP control statement must refer in the forward direction to a statement not yet processed.

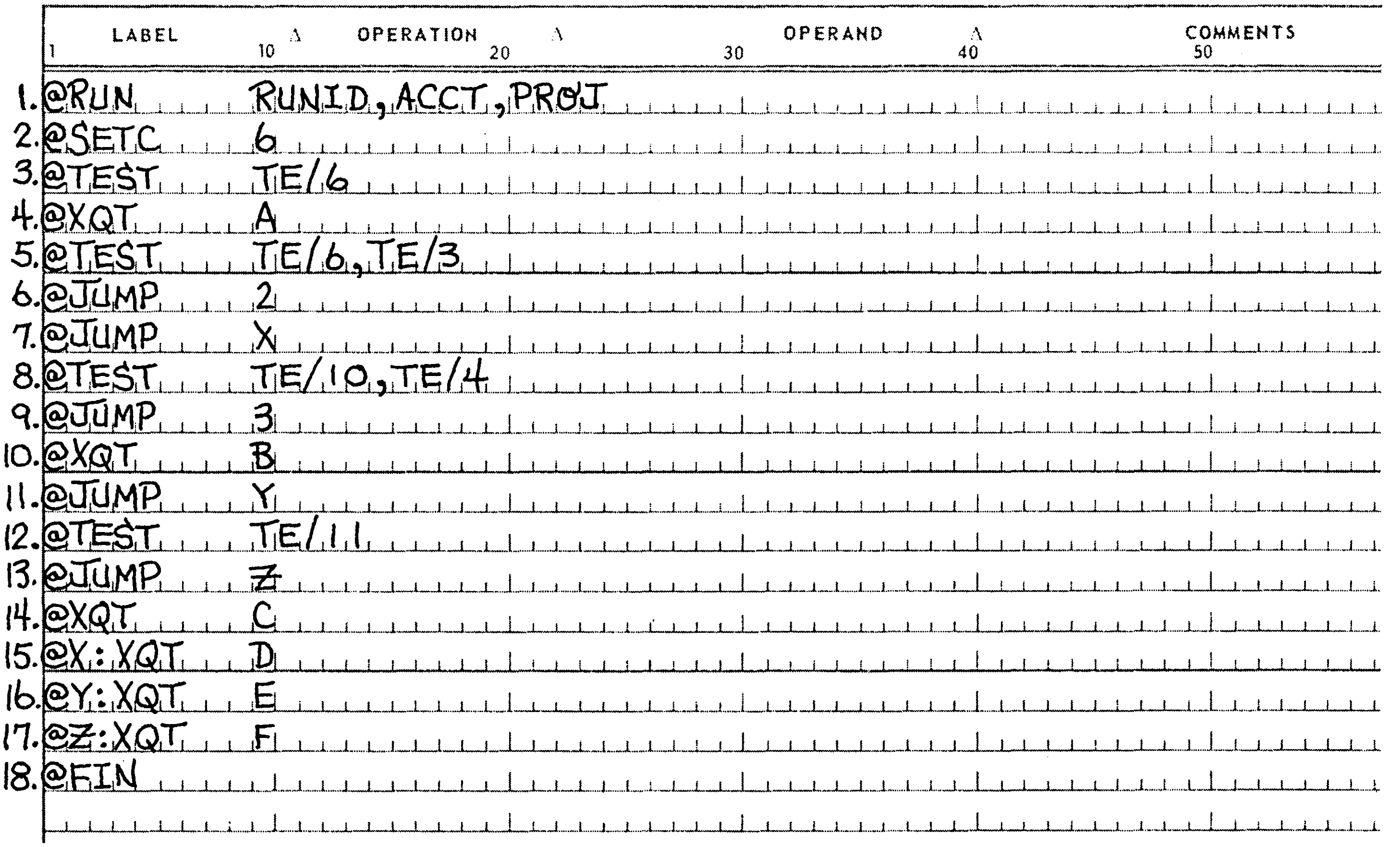
Examples:

```
   LABEL      Δ    OPERATION     Δ              OPERAND      Δ               COMMENTS
 1           10               20              30               40               50
1. @JUMP   TAG
2. @JUMP   4
```

1. Control is advanced to the control statement containing the label TAG.

2. Control is advanced to the fourth control statement following the execution of the @JUMP control statement.

3.9.4.4. CONDITIONAL RUN STREAM EXAMPLE



```
  LABEL       OPERATION                OPERAND                  COMMENTS
1         10            20          30           40          50
 1.@RUN     RUNID,ACCT,PROJ
 2.@SETC    6
 3.@TEST    TE/6
 4.@XQT     A
 5.@TEST    TE/6,TE/3
 6.@JUMP    2
 7.@JUMP    X
 8.@TEST    TE/10,TE/4
 9.@JUMP    3
10.@XQT     B
11.@JUMP    Y
12.@TEST    TE/11
13.@JUMP    Z
14.@XQT     C
15.@X:XQT   D
16.@Y:XQT   E
17.@Z:XQT   F
18.@FIN
```

As shown in this example, lines 1,2,3,5,7,15,16,17, and 18 are processed in this sample run stream, and programs D, E, and F are executed in that order.

Line 2 might be changed to set other values to produce different run stream processing, as follows:

Line 2	Lines processed	Programs executed
@SETC 3	1,2,3,4,5,7,15,16,17,18	A,D,E,F
@SETC 4(or 10)	1,2,3,4,5,6,8,10,11,16,17,18	A,B,E,F
@SETC 11	1,2,3,4,5,6,8,9,12,14,15,16,17,18	A,C,D,E,F
@SETC 1	1,2,3,4,5,6,8,9,12,13,17,18	A,F

4. EXECUTIVE SERVICE REQUESTS

4.1. INTRODUCTION

This section discusses the fundamental interfaces between an executing program and the executive. These interfaces are the executive request (ER) mechanism, and the contingency mechanism. The material in this section is primarily of interest to the machine (assembler) language programmer.

The Executive Request instruction has the general form:

```
     ER   function-id
```

The function-id is coded in the instruction u field, as a symbolic, system-defined name. At collection, this name is converted into an absolute ER index. The numeric index associated with each name is specified in the system relocatable element ERU$.

The function-id identifies to the executive a particular service to be performed for the requesting activity. This service may be as simple as a clock reading, or a complex file handling operation. See 4.2 for a summary of all ERs provided by the executive.

4.1.1. CODING RESTRICTIONS

Only the u field may be used when executing an ER instruction. No indexing, index modification, or indirect addressing is permitted. Also, an ER instruction must not be executed remotely with an Execute (EX) instruction.

4.1.2. CALLING SEQUENCE CONVENTIONS

The calling sequences for ERs may, in general, be used reentrantly. This means that parameters are passed in control registers, either directly or indirectly, by a packet whose address is passed in a control register. The use of Test and Set (TS) instructions across an ER to achieve reentrancy is poor practice, and in the case of real time activities may cause a system stall which can only be relieved by operator intervention.

ER parameters that specify numeric values such as packet lengths are binary unless otherwise stated.

Normally, when a control register is required in ER calling sequences to hold parameters or results, register A0 is used. Additional registers, if needed, are usually allocated in the sequence A1, A2, . . .

Control register contents (including parameters) are not altered by the execution of an ER, unless specific resultant values are defined in a control register.

The coding sequences shown for particular ERs are optimal in most instances, but any coding sequence that achieves the same register loading is permitted. Note that many ERs require just a packet address in register A0, which means only H2 of register A0 is significant; however, if the coding sequence given clears H1 of register A0 to zero, then it must be zero. This principle applies to all calling sequences.

When a parameter must be the address of another parameter, the second parameter must not be in a control register.

4.1.3. ER SYNCHRONY

For most ERs, processing is completed prior to returning control to the requesting activity. Such requests are divided into two types: synchronous and immediate. Immediate ERs are of a short, simple nature and do not normally cause switching, whereas synchronous ERs are of sufficient duration or complexity to require suspension of the requesting activity while various executive components perform the requested service. With the exception of timing considerations pertinent to real time applications, the differences between synchronous and immediate ERs does not influence programming logic.

A few ERs return control to the requesting activity prior to completing the required processing. These ERs are asynchronous ERs. In general, control is returned immediately after processing has been initiated. The activity may then do other processing in parallel. At some future point, the program (usually the same activity) must synchronize itself with the completion of the requested service; this is most commonly done by checking a status value in a packet associated with the request. Asynchronous ERs are generally ones which perform I/O.

Table 4–1 specifies the type of each ER.

4.1.4. ERROR HANDLING

Programming errors in an executing program are detected in two basic ways. The first method is hardware detection of errors such as divide fault, illegal operation, guard mode violation, and so on. The second method is software detection, within the executive, of errors in ER usage.

Such errors cover a wide range, from simple mistakes like forgetting to assign a file to such subtle errors as allowing all activities to deactivate waiting for each other to do something.

In a few error cases, it is not feasible for the executive to do anything but abort the run. However, the vast majority of errors are at least potentially recoverable.

In the case of recoverable errors, the offending activity is placed in error mode, at which point a contingency occurs and the activity is either error terminated or, if a contingency routine has been registered to handle error mode contingencies, the activity is given control at that routine. See 4.9 for details of contingency handling and error termination.

In most cases, errors are detected immediately. However, certain kinds of errors for asynchronous ERs may be detected after control has been returned from the ER, in which case the contingency may occur at (and capture) an instruction address far removed from the offending instructions.

Error mode errors are not to be confused with errors not normally attributable to programming errors such as a parity error on an I/O operation; these do not cause error mode contingencies.

Documentation of the individual ERs in this manual gives all restrictions and warnings pertinent to their use, but generally does not include all possible associated error cases and error codes. In many cases, errors are common to many unrelated ERs; for example, all packet addresses are checked against program storage limits. See Appendix C for a complete list of error codes and diagnostic messages.

4.2. SUMMARY OF EXECUTIVE REQUESTS

Table 4–1 lists the name, octal function code, description, ER type, and a cross-reference for each ER. ERs that are fundamental to activity and program control, and the miscellaneous ERs are covered in this section. The remaining ERs are described in the sections covering the specific area with which the ER is associated.

The ER type designations are as follows:

A	= Asynchronous
I	= Immediate
S	= Synchronous
—	= Not applicable

ER Name	Octal Function Code	Description	Type	Cross Reference
ABORT$	12	Abort run	–	4.3.2.3
ACT$	47	Activity activation	S	4.3.3.4
ADACT$	154	CADD$ and ACT$ (ESI only)	I	15.5.1
APCHCA$	77	ASCII punch control alternate	S	5.4.8
APCHCN$	75	ASCII punch control	S	5.4.6
APNCHA$	73	ASCII punch alternate	S	5.3.8
APRINT$	70	ASCII print	S	5.3.2
APRNTA$	71	ASCII print alternate	S	5.3.4
APRTCA$	76	ASCII print control alternate	S	5.4.4
APRTCN$	74	ASCII print control	S	5.4.2
APUNCH$	72	ASCII punch	S	5.3.6
AREAD$	166	ASCII read	S	5.2.2.
AREADA$	167	ASCII read alternate	S	5.2.4
AWAIT$	134	Wait for other activities to terminate	S	4.3.3.1
BBEOF$	36	Set block buffering end-of-file	S	13.3.2.8
CADD$	57	Add communications buffer	I	15.4.2.3
CEND$	100	Terminate contingency status	I	4.9.4.2
CGET$	56	Get communications buffer	S	15.4.2.2
CJOIN$	151	Expand communications buffer pool	S	15.4.2.4
CLIST$	153	User access to control statements	S	5.5
CMD$	51	Dial communciations line	A	15.4.1.2
CMH$	52	Hang-up communications line	A	15.4.1.9
CMI$	47	Initiate communications input	A	15.4.1.3
CMO$	50	Initiate communications output	A	15.4.1.4
CMS$	45	Line terminal initiation	S	15.4.1.1
CMSA$	53	Initiate communications input and output	A	15.4.1.5
CMT$	46	Terminate communications line	S	15.4.1.10
COM$	10	Console output and solicited input	S	4.6.1
COND$	66	Retrieve condition word	I	4.4.2
CPOOL$	55	Create communiations buffer pool	S	15.4.2.1
CREL$	152	Release communications buffer pool	S	15.4.2.5
CSF$	17	Control statement function	S	4.8.1
DACT$	150	Activity deactivation	S	4.3.3.3

Table 4–1. Available ERs (Part 1 of 3)

ER Name	Octal Function Code	Description	Type	Cross Reference
DATE$	22	Retrieve time and date	I	4.5.1
EABT$	26	Error terminate run	–	4.3.2.4
ERR$	40	Error terminate activity	–	4.3.2.2
EXIT$	11	Normal activity termination	–	4.3.2.1 10.4.5.3
EXLNK$	173	Return to calling reentrant processor or main program	I/S	10.4.5.1
FACIL$	114	Retrieve file assignment information	S	7.2.7
FACIT$	143	Retrieve file assignment information	S	7.2.7
FITEM$	32	Retrieve file assignment information	S	7.2.6
FORK$	13	Create new activity	S	4.3.1.1
IALL$	101	Register contingency routine	I	4.9.3.1
II$	27	Unsolicited console input	S	4.6.2
IO$	1	Initiate I/O	A	6.3.3
IOARB$	21	Initiate arbitrary device I/O	A	6.9.2
IOAXI$	20	Exit and initiate arbitrary device I/O with interrupt activity	S	6.9.3
IOI$	2	Initiate I/O with interrupt activity	A	6.3.4
IOW$	3	Initiate I/O and wait	S	6.3.5
IOWI$	24	Initiate I/O with interrupt activity and wait	S	6.3.6
IOXI$	25	Exit and initiate I/O with interrupt activity	S	6.3.7
LABEL$	31	Manipulate tape labels	S	7.3.1
LCORE$	44	Release program storage	S	4.7.2
LINK$	171	Attach to reentrant processor	I/S	10.4.4.1
LOAD$	111	Load program segment	S	10.2.4.5.1
MCORE$	43	Acquire program storage	S	4.7.1
MCT$	41	Retrieve master configuration table	S	4.8.3
MSCON$	125	Master file directory manipulation	S	22.3
NAME$	146	Name an activity	S	4.3.3.2
NRT$	62	Terminate real time status	I	4.3.4.2
OPT$	63	Retrieve options	I	4.8.2
PCHCA$	165	Punch control alternate	S	5.4.7
PCHCN$	164	Punch control	S	5.4.5
PCT$	64	Program control table retrieval	I	4.8.3
PFD$	106	Delete an element from a program file	S	24.3.1.3

Table 4–1. Available ERs
(Part 2 of 3)

ER Name	Octal Function Code	Description	Type	Cross Reference
PFI$	104	Insert an element into a program file	S	24.3.1.1
PFS$	105	Find an element in a program file	S	24.3.1.2
PFUWL$	107	Update next program file write location	S	24.3.1.4
PFWL$	110	Obtain next program file write location	S	24.3.1.4
PNCHA$	145	Punch alternate	S	5.3.7
PRINT$	16	Print	S	5.3.1
PRNTA$	144	Print alternate	S	5.3.3
PRTCA$	155	Print control alternate	S	5.4.3
PRTCN$	137	Print control	S	5.4.1
PSR$	157	Processor state register control	I	4.8.4
PUNCH$	130	Punch	S	5.3.5
READ$	15	READ	S	5.2.1
READA$	42	Read alternate	S	5.2.3
RLINK$	172	Attach to a reentrant processor	I/S	10.4.4.2
RLIST$	175	Reentrant processor registration	S	10.4.3
ROUTE$	133	Line terminal transfer	S	15.4.3
RT$	61	Establish real time status	I/S	4.3.4.1
SETC$	65	Set condition word	I	4.4.1
SNAP$	120	Snapshot dump	S	4.8.5
TDATE$	54	Retrieve time and date	I	4.5.2
TFORK$	14	Create new activity with timed wait	S	4.3.1.2
TIME$	23	Retrieve time of day	I	4.5.3
TINTL$	136	Initialize tape file to beginning of first reel	S	7.2.9
TREAD$	102	Print and read	S	5.2.5
TSWAP$	135	Swap reels of tape file	S	7.2.8
TWAIT$	60	Timed wait	S	4.3.5
UNLCK$	67	Terminate interrupt activity status	S	6.3.8
UNLNK$	174	Return to main program from reentrant processor	I	10.4.5.2
WAIT$	6	Wait for completion of I/O request	S	6.3.1
WANY$	7	Wait for any I/O completion	S	6.3.2

Table 4–1. Available ERs
(Part 3 of 3)

4.3. ACTIVITY AND PROGRAM CONTROL

4.3.1. ACTIVITY REGISTRATION

4.3.1.1. CREATE A NEW ACTIVITY (FORK$)

Purpose:

Register and initiate an asynchronous program activity.

Format:

```
     L    A0,(parameter-word)
     ER   FORK$
```

Description:

Each activity of a program executes independently of all other activities. A FORK$ request creates a new activity and schedules it for execution. Parameter-word describes characteristics to be associated with the new activity and specifies the program address at which it is to be given control. The format of parameter-word is:

S1	S2	S3	H2
[RT-priority]	[activity-id]	registers	entry-address

Entry-address is the program address at which the new activity is to begin execution.

The registers field specifies the set of control registers which must be saved for the activity and which initially have the same contents as the corresponding registers of the initial activity. A value of zero indicates that only the minor set of registers (X11, A0–A5, R1–R3) are required. A nonzero value indicates that all X, A, and R registers (except X0 and R0) are required. Once selected, the control register set remains with the activity until it is terminated. Note that if an activity with only the minor set of registers creates an activity with the complete register set, the initial register contents are only defined for the minor set. The space and time required, within the executive to maintain a minor register set activity is significantly less than for an activity with the entire register set.

The activity-id field is used to associate a numeric identity with the new activity. If used, the activity-id must be unique within the program and have a value from 1 through 35 and must not currently be in use. A zero specifies that the activity is not to have an activity-id (note that the initial activity of a program has no activity-id). See the discussion of AWAIT$ (4.3.3.1) for the use of activity-id's.

The RT-priority field allows a real time priority to be assigned to the new activity. If used, the value must be in the range 2 through 35 and within the range allowed to the account number. Note that at least one other activity must previously have elevated itself to real time by an RT$ request before this method can be used.

See 4.3.4 and Section 17 for additional information on real time activity/program control and real time processing.

4.3.1.2. CREATE A NEW ACTIVITY WITH TIMED WAIT (TFORK$)

Purpose:

Creates a timed activity. A TFORK$ request is similar to a FORK$ request (see 4.3.1.1) except that the new activity does not begin execution for a specified amount of time.

Format:

```
L     A0,(parameter-word)
L     A1,(wait-time-in-milliseconds)
ER    TFORK$
```

Description:

The format and meaning of the paramter-word is identical to the parameter-word used in a FORK$ request (see 4.3.1.1). The wait time may be any value from 2 to 30,000 (2 milleseconds to 30 seconds). If the value exceeds 30,000, 30,000 milliseconds is used. If the new activity is real time, the wait time may exceed the 30,000 millisecond limit. Note that the wait time is simply a minimum elapsed time "by the clock", and is not influenced by the amount of processing (if any) devoted to other activities of the program (for example, program might be time swapped); for this reason, the TFORK$ request is primarily intended for real time applications.

4.3.2. ACTIVITY TERMINATION

The following ERs provide various forms of activity termination. When an activity terminates, it ceases to exist for the program and the system. The activity-id and name are released for reuse by any other activities. When the last activity of a program terminates, the program is terminated.

4.3.2.1. ACTIVITY NORMAL TERMINATION (EXIT$)

Purpose:

Terminate the current activity.

Format:

```
ER    EXIT$
```

Description:

The current activity is terminated, and the program is also terminated if this is the last activity.

4.3.2.2. ACTIVITY ERROR TERMINATION (ERR$)

Purpose:

Place the requesting activity in error mode (normally causes error termination).

Format:

```
ER    ERR$
```

Description:

See 4.1.4 and 4.9 for a complete discussion of error termination and error mode.

4.3.2.3. ABORT RUN (ABORT$)

Purpose:

Unconditionally terminate the program and the run.

Format:

```
    ER    ABORT$
```

Description:

All activities of the program, including the requesting activity, are unconditionally terminated. No register dumps are provided for the activities. In addition, the run is terminated and PMD requests are not honored. For demand runs, only the program is terminated and not the run.

If an abort contingency routine has been registered, a single new activity is created after all the program's activities are terminated. The new activity is given control with a complete set of control registers (contents not saved) at the program's contingency routine address. If any of the terminated activities were real time, the new activity is given the highest real time priority allowed to the RUN's account number.

See 4.9 for a discussion of contingencies.

4.3.2.4. PROGRAM ERROR TERMINATION (EABT$)

Purpose:

Unconditionally error terminates all activities but allows error diagnostics.

Format:

```
    ER    EABT$
```

Description:

The EABT$ request is similar to the ABORT$ request (4.3.2.3) except that register dumps are provided for all terminating activities and PMD requests are honored. The program is error terminated instead of being aborted (see SETC control statement and SETC$ executive request, 3.9.4.1 and 4.4.1, respectively). Abort contingency also applies to the EABT$ request.

4.3.3. ACTIVITY SYNCHRONIZATION

In programs which use asynchronous activities to achieve parallel processing, it is often necessary for an activity to wait upon the completion of processing which is being performed by other activities. Several ERs are provided to achieve the desired program synchronization.

Activity synchronization is accomplished by removing the requesting activity from consideration for CPU time (deactiviating it) until some other activities indicate that the desired processing is complete.

The programmer must be careful that all activities do not simultaneously go into synchronization waits (see AWAITS - 4.3.3.1 and DACT$ - 4.3.3.3); if this occurs, the program and run are unconditionally aborted with an AWAIT/DEACT AMBIGUITY error diagnostic message.

There are two I/O requests, IOXI$ and IOAXI$ (see 6.3.7 and 6.7.3, respectively), which convert existing activities into interrupt activities. In these cases, the interrupt activity retains the activity-id and name associated with the original activity. Conversely, no activity-id or name retention occurs for when a new interrupt activity is created (by IOI$ - 6.3.4, IOWI$ - 6.3.6, or IOARB$ - 6.7.2).

4.3.3.1. JOINING OF ACTIVITIES (AWAIT$)

Purpose:

Deactivate requestor until all specified activities are terminated.

Format:

```
L     A0,(activity-id-mask)
ER    AWAIT$
```

Description:

The AWAIT$ request is used when further execution of the requesting activity is not desired until all specified activities have terminated.

The requesting activity must have an id number (note this rules out initial program activity) as must all activities for which it desires to wait (see 4.3.1.1). The activities to be waited upon are specified by setting the bits in the activity-id mask corresponding to the activity-id number. (An activity with an id of 4 corresponds to bit 4 in the mask.) Bit 0 must not be used. The requesting activity is deactivated until all specified activities have terminated by means of EXIT$ or ERR$.

4.3.3.2. ACTIVITY NAMING (NAME$)

Purpose:

Attaches a name to an activity for identification purposes and for future referencing by ACT$ or DACT$ requests (see 4.3.3.4 and 4.3.3.3, respectively). The attached name is not the same as that used in conjunction with an AWAIT$ request (see 4.3.3.1).

Format:

```
L,U   A0,18-bit-activity-name
ER    NAME$
```

Description:

The 18-bit name loaded into H2 of register A0 is expanded to 36 bits (full word) by the executive and includes the three user-supplied characters in H2 of register A0. The user-supplied portion of the activity name must be unique for each named activity as the executive does not otherwise guarantee uniqueness. The full 36-bit name, which is returned in A0, must be used with subsequent ACT$ requests.

4.3.3.3. ACTIVITY DEACTIVATION (DACT$)

Purpose:

Deactivates the calling activity which must have been previously named by the NAME$ request (see 4.3.3.2).

Format:

```
ER    DACT$
```

Descriptions:

Reactivation of this activity requires that an ACT$ request (see 4.3.3.4) be made from some other activity. Control is immediately returned to the next instruction following the DACT$ reference of the deactivated activity.

If some other activity has performed an ACT$ request specifying the requestor's name before the requestor performed the DACT$ request, the requestor is not deactivated but is returned control immediately. This is necessary as there is no way for the activity performing the ACT$ request to determine if the requestor has performed the DACT$ request.

4.3.3.4. ACTIVITY ACTIVATION (ER ACT$)

Purpose:

Activates an activity which must have been previously named by the NAME$ request (see 4.3.3.2).

Format:

```
L     A0,activity-name
ER    ACT$
```

Description:

The 36-bit name returned as a result of the NAME$ request is used to activate the activity. The requesting activity need not be a named activity.

Prior to issuing the ACT$ request, the user must load register A0 with the contents of a main storage location in which the name attached to the requested activity has been stored.

If the activity being activated is already active it is flagged such that when it next executes a DACT$ request, it is automatically reactivated.

4.3.4. REAL TIME PROGRAM/ACTIVITY CONTROL

4.3.4.1. CHANGING PROGRAM/ACTIVITY TO REAL TIME STATUS (RT$)

Purpose:

Raises the status of a program to real time if the run's account number permits such action. The RT$ request also allows a real time activity to change its switching priority level within the real time class.

Format:

```
L,U   A0,switching-priority-level
ER    RT$
```

Description:

Real time status is provided for programs servicing communications lines. To allow for the time critical nature of these programs, a program/activity which is real time is afforded privileges which non real time programs/activities do not enjoy. Namely they:

- have top priority for CPU switching and I/O;

- are not swapped out of main storage;

- have access to certain ER's, primarily the communications requests.

The allowable switching priority (2 through 35) for each program's activities is controlled by account number. If a switching priority higher than that permitted is requested, the activity is placed in error mode. When the requesting activity currently has real time status, the ER is used only to control that activity switching priority.

A program is considered real time when any of its activities acquire real time status. Because the program may not be swapped, an RT$ request from within a program causes it to be positioned in main storage in such a way so as to cause minimum impact on the total system. Additional RT$ requests from within the program do not cause such relocation.

4.3.4.2. REMOVAL OF PROGRAM/ACTIVITY REAL TIME STATUS (NRT$)

Purpose:

Reduces an activity/program from real time status.

Format:

```
    ER   NRT$
```

Description:

The activity is returned to the original program type (batch or demand). A program retains real time status until all real time activities are reduced in status or terminated, at which time the program also returns to its initial type.

4.3.5. TIMED ACTIVITY WAIT (TWAIT$)

Purpose:

Places the activity in a wait state (delays execution) for a specified period of time.

Format:

```
    L    A1,(wait-time-in-milliseconds)
    ER   TWAIT$
```

Description:

The activity must load register Al with the desired wait time period to making the TWAIT$ request. This value has the same meaning as the wait time for the TFORK$ request (see 4.3.1.2).

4.4. CONDITION WORD CONTROL

The program condition word which contains program status information supplied by the executive and information inserted by user-supplied control statements can also be modified dynamically from within an executing program. A complete description of the format and content of the condition word can be found in 3.9.4.

4.4.1. SETTING THE CONDITION WORD (SETC$)

Purpose:

Dynamically sets T3 of the program condition word.

Format:

```
    L,U  A0,value
    ER   SETC$
```

Description:

This ER transfers T3 of register A0 to T3 of the condition word. The intital contents of T3 of the condition word are 0. This ER performs a function similar to the @SETC control statement which sets T2 of the condition word (see 3.9.4).

4.4.2. CONDITION WORD RETRIEVAL (COND$)

Purpose:

Transfers the program condition word to register A0.

Format:

ER COND$

Description:

The program condition word is placed in register A0. This does not modify the condition word itself.

The condition word format returned by the COND$ request is:

T1	T2	T3
error-condition-bits	*0-or-value-set-by-@SETC*	*0-or-value-set-by-ER-SETC$*

Error-condition-bits:

Bit 30 – Inhibit run termination on program error terminations (set by @SETC,I control statement and cleared by @SETC,A (see 3.9.4.1).

26 – Last activity termination was an ABORT$ (not EABT$).

25 – Last activity termination was an error termination.

24 – Some previous activity of the current task terminated in error.

4.5. RETRIEVAL OF THE TIME AND DATE

4.5.1. TIME AND DATE IN FIELDATA (DATE$)

Purpose:

Places the fieldata date and time into registers A0 and A1, respectively.

Format:

ER DATE$

Description:

Register A0 contains:

T1	T2	T3
month (01-12)	*day-of-month* (01-31)	*last-two-digits-of-the-year*

Register A1 contains:

T1	T2	T3
hour (00 through 23)	*minutes* (00-59)	*seconds* (00-59)

4.5.2. TIME AND DATE IN BINARY (TDATE$)

Purpose:

Places the binary date and time into register A0.

Format:

```
ER   TDATE$
```

Description:

Register A0 contains:

S1	S2	S3	H2
month (1-12)	*day* (1-31)	*year* (MODULO 1964)	*time-in-seconds-past-midnight*

S3 is the last two digits of the year, Modulo 1964, the current year can be computed by adding 1964 to the value returned in S3 of A0.

4.5.3. TIME IN MILLISECONDS (TIME$)

Purpose:

Places the current time past midnight in milliseconds into control register A0 in binary.

Format:

```
ER   TIME$
```

Description:

Register A0 contains:

35	0
time-in-milliseconds-past-midnight	

4.6. CONSOLE COMMUNICATIONS

4.6.1. CONSOLE OUTPUT AND SOLICITED INPUT (COM$)

Purpose:

To request use of the onsite operator's console to display output messages and solicit operator input.

Format:

```
L,U  A0,pkeaddr
ER   COM$
```

Description:

Pktaddr is the address of a packet whose format is:

	S1	S2	S3	H2
Word 0	*error-code*	console-class	0	*actual-input-char-count*
1	0	output-char-count (max. 50)		output-buffer-addr
2	expected-input-char-count (max. 50)			input-buffer-addr

Word 0

error-code

Contains a COM$ request error code (see Appendix C) if an error is detected in the packet (the activity is also placed in error mode).

console-class

The user may direct the message to any console by specifying the appropriate console class code. The codes are:

0_8 – System console

1_8 – I/O activity console

2_8 – Communications console

3_8 – Hardware confidence console

actual-input-char-count

Contains the number of input characters received. Always less than or equal to expected-input-char-count.

Word 1

output-char-count

The number of characters in the message to be displayed. The message is restricted to 50 characters maximum. Each character is edited and master spaces (@) are deleted from the message (they must be included in the character count). If this field is zero, the COM$ request is ignored. If a character count greater than 50 is specified, the output message is truncated at 50 characters.

output-buffer-addr

The address of the program buffer containing the output message. The characters of the message are obtained from successive sixths of a word, beginning with S1 of the first word of the buffer.

Word 2

expected-input-char-count

When this field contains a nonzero value, a console operator response is solicited. The activity executing the COM$ request is placed in a wait state until the input message is complete. If the input message exceeds the expected character count, the input message is discarded and the console operator is requested to retype the message. When no input message is desidered, set this field to zero. The maximum number of characters permitted in the input message is 50.

input-buffer-addr	The address of the program buffer that will hold the input message. Input characters are stored in successive sixths of a word starting with S1 of the first word of the buffer. If the last word of the input message does not contain six characters, the remainder of the word is filled with Fieldata blanks (05_8). The end-of-message (EOM) symbol is not transferred to the buffer.

4.6.2. UNSOLICITED CONSOLE INPUT (II$)

Purpose:

Provides a means to define the activity which is to accept any unsolicited input directed to the program.

Format:

```
ER   II$
```

Description:

The activity executing the II$ request is deactivated as for a DACT$ request (see 4.3.3.3), however, the activity need not be named. If named, it may be reactivated using an ACT$ request (see 4.3.3.4). An II$ request when an II activity has already been defined is not allowed.

Unsolicited console input of up to six characters is stored (left-justified, space filled) in the activity's A0 register, and the activity is activated.

After activation (by either ER ACT$ or console input), the activity is no longer defined as the unsolicited console input activity. The same activity or some other activity must execute another II$ request to redefine the unsolicited console input activity.

Unless the program is guaranteed that unsolicitated input will occur to cause the activation of the II$ activity, the activity must be named and activated by an ACT$ request prior to program termination. Failure to do this aborts the program and the message AWAIT/DACT AMBIGUITY is placed in the program's PRINT$ file.

The console input activity is also activated by the remote terminal BREAK keyin. Since no input is actually received, register A0 is space filled.

4.7. PROGRAM STORAGE EXPANSION AND CONTRACTION

4.7.1. MAIN STORAGE EXPANSION (MCORE$)

Purpose:

Permits user program to request additonal main storage for the I bank or D bank.

Format:

```
L,U   A0,highest-addr-required
ER    MCORE$
```

Description:

The address requested by the activity is assumed to be in the I bank if the address is less than the first D bank address produced at collection time. Otherwise, the address is assumed to be a D bank address.

If the main storage requested is already assigned to the program, the activity's storage limits are adjusted and control returns to the requesting activity. If the storage is not already assigned, the requesting activity is deactivated until storage can be made available by swapping this or some other program.

Additional storage cannot be obtained if any activity of the program is in real time status, unless the storage can be obtained without moving the program. Nonreal time programs are swapped, if necessary, to make the requested storage space available. Requests for less space than is currently available to a program are ignored, and control returns to the user activity.

Each activity of a multiple activity program must perform a MCORE$ request before additional storage is made available to that activity. This provides for automatic synchronization and assures each activity of storage space when needed.

The storage limits for ESI completion activities are not automatically expanded by a MCORE$ request. The user can cause this expansion by setting bit 35 in register A0 at time of the request.

The MCORE$ request permits the user to request an I bank or D bank where there previously was none. The I bank may be expanded to the first D bank address generated when the program is collected. The D bank may be expanded to beyond 65K. If expanded beyond 65K (177777_8), an index register must be used to address those locations in excess of 65K.

The expanded storage space allocated is maintained for the life of the program.

The additional main storage space obtained through the MCORE$ request is cleared to zero unless the program collection specified the B option on the @MAP control statement (see 10.2.1).

4.7.2. MAIN STORAGE CONTRACTION (LCORE$)

Purpose:

Releases unneeded main storage in the I bank or D bank.

Format:

```
L,U  A0,highest-address-required
ER   LCORE$
```

Description:

The address specified is assumed to exist in the I bank if the address is less than the first D bank address produced at collection time. Otherwise, the address is assumed to be a D bank address.

The entire I bank or D bank can be released by specifying the first address of the respective bank. Before programming a release of the D bank, however, the @MAP listing for the program should be checked to ensure that necessary collector-produced tables are not contained in the D bank.

Main storage is released to that spanned by the largest I and D bank of any current activity's storage limits. In making the storage limits check, one complete storage limits value is used for all ESI completion activities. When main storage is actually released, the segment load table is updated to show all segments that lay totally outside the program's area as not being in main storage. If the segment is in main storage at the time of the release and any part of the segments I and D bank is still within the program's area, the segment is left marked in main storage. If I/O is outstanding for this activity at the time of the LCORE$ request, request satisfaction is delayed until the I/O is completed.

In a multiactivity program, all activities whose storage limits span the area to be released must perform an LCORE$ request before the program size can actually be reduced.

4.8. MISCELLANEOUS EXECUTIVE REQUESTS

4.8.1. DYNAMIC REQUEST OF CONTROL STATEMENTS (CSF$)

Purpose:

Permits the user program to submit certain control statements for interpretation and processing during program execution rather than from the run stream.

Format:

```
    L    A0,(image-length,image-addr)
    ER   CSF$
```

Parameters:

image-length	Length in words of the control statement image
image-addr	Address of the buffer that contains the image

Description:

The submitted image must be in the identical Fieldata format, including the character @ in S1 of the word 0, as it would have been if it had been submitted as a regular control statement in the input run stream.

Termination of scan results from whichever occurs first: a comment of blank-period-blank is encountered, a blank following the last allowable parameter field is encountered, or the image-length in the H1 of register A0 has been exceeded.

Maximum allowed value for image-length is 40 words; 14 is assumed if 0 is given.

The control statements which may be processed by the CSF$ request are:

@ADD
@ASG
@BRKPT
@CAT
@CKPAR
@CKPT
@FREE
@LOG
@MODE
@QUAL
@RSPAR
@RSTRT
@START
@SYM
@USE

Control statement syntax and other errors generally result in error mode termination with contingency type 12_8 (see 4.9.4), error type 4, and error code as follows:

Error Code	Description
40_8	Syntax error
41_8	Image length greater than 40
42_8	Control statement is not one that can be processed by the CSF$ request
43_8	Invalid address given for the image buffer
44_8	Too many @LOG control statement entries given for the program

When certain control statements are submitted by the CSF$ request, register A0 is returned containing status or error information. For the facility request statements (@ASG, @CAT, @FREE, @MODE, @QUAL, and @USE), bits set in register A0 upon return from the CSF$ request indicate that either the request was rejected or that it was accepted with precautionary warnings (see 4.8.1 for interpretation of bit settings in A0). The meaning of the bits set in register A0 upon return from processing the @BRKPT and @SYM symbiont control statements are described in section 3.6.4. The status codes returned in register A0 for a @CKPT and @CKPAR CSF$ request are described in 17.4; on return from a @CKPT request, H1 of register A0 contains the checkpoint number.

On return from a CSF$ @START request, register A0 contains codes as follows:

0_8 – Request processed normally

1_8 – Request rejected due to improper run stream in file

2_8 – Request rejected due to file unobtainable

3_8 – Request rejected due to element obtainable

4_8 – Request rejected due to filename not specified

If errors are encountered while processing an @ADD control statement submitted by the CSF$request, control is not returned following the CSF$ request. Rather, error mode termination is entered with contingency type 12_8 (see 4.9.4), error type 2_8, and error codes as specified in 3.9.1.

For CSF$ requests for processing @LOG, @RSPAR, and @RSTRT control statements, no status information is returned in register A0.

Example:

The following example illustrates how an @ASG control statement (see 2.3.2.1.1) is submitted by an executive request to the CSF$ function.

Assume the user wants to assign a temporary FASTRAND-formatted scratch file named FILEA and to reserve two granules. This can be coded as follows:

```
   LABEL      Λ   OPERATION     Λ             OPERAND         Λ               COMMENTS
1          10               20             30              40              50
         L,U       AO,LADD
         ER        CFS$
         .
         .
         .
IADD     '@ASG,T   FILEA,F2 . '
```

The blank-period-blank construction terminates the image scan.

4.8.2. RETRIEVING @XQT CONTROL STATEMENT OPTIONS (OPT$)

Purpose:

Makes available options specified on the @XQT statement (see 2.3.3.2).

Format:

```
ER   OPT$
```

Description:

When control is returned, the specified option letters are set in register A0 in master bit notation, that is, letter A sets bit 25; letter B sets bit 24; letter C sets bit 23;. . . letter Z sets bit 0. Bits 35–26 are always returned as zero.

4.8.3. PROGRAM CONTROL TABLE RETRIEVAL (PCT$)

Purpose:

Makes all or specified portions of the information stored in the program control table (PCT) available to the requesting program.

Format:

Two formats are available:

To transfer a maximum of 1000$_8$ words starting at word zero of the main block:

```
    L     A0,(word-count,buffer-addr)
    ER    PCT$
```

To transfer all or part of the PCT starting at any PCT-relative address:

```
    L     A0,(0,buffer-addr)
    L     A1,(n,relative-addr)
    ER    PCT$
```

Parameters:

buffer-addr — Address within the program where the PCT is to be transferred.

n — Number of words to be transferred.

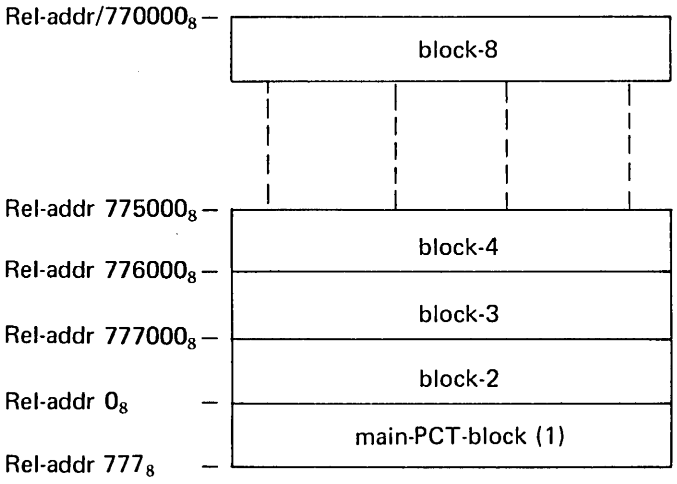
relative-addr — Address relative to the start of the PCT main block from which the transfer should occur.

Description:

For information concerning the contents and internal format of the PCT, refer to the latest version of the *UNIVAC 1100 Series Systems Memorandum.* This ER must be used with caution in that UNIVAC reserves the right to change the content or format of the PCT without notice.

A PCT's size is determined by program requirements with a normal maximum of nine main storage blocks (512 words each). The structure and addressing of the PCT is illustrated by the following examples.

Assume an eight-block PCT:



```
      L     A0,(0,buffer)
      L     A1,(02000,0777000)
      ER    PCT$
```

This call transfers block-2 and the main-PCT-block, in that order, to the address BUFFER.

```
      L     A0,(0,buffer)
      L     A1,(0500,0100)
      ER    PCT$
```

Transfers 500_8 words to BUFFER starting at relative address 100_8 in the main-PCT-block.

```
      L     A0,(0,buffer)
      L     A1,(0100,0777400)
      ER    PCT$
```

Transfers 100_8 words to BUFFER from relative address 400_8 through 477_8 of PCT-block-2.

4.8.4. ALTERING PROCESSOR STATE REGISTER (PSR$)

Purpose:

Permits an activity to dynamically set and clear those bits within the processor state register (PSR) which establishes the following standard control modes:

- quarter-word mode (bit 17)
- double-precision underflow mode (bit 32)
- floating-point compatibility mode (bit 35)

Format:

```
      L     A0,(parameter-word)
      ER    PSR$
```

Description:

Bit 0, 2, and 3 in the parameter word control the modification of PSR bits 17, 32, and 35, respectively. When a control bit is 0, 2, and 3, the associated bit in the PSR is set to the value of the corresponding bit in the parameter word. For example, if bit 0 in the parameter-word is set, the content of bit 17 of the parameter-word is placed in bit 17 of the PSR.

Upon returning from the PSR$ request, register A1 contains the program's former PSR contents.

Examples:

The following examples illustrate typical parameter words loaded into control register A0 and their interpretations for a PSR$ request.

Parameter Word (Octal)	Description
000000000000	No modification of existing PSR; enables readout of PSR in control register A1.
440000400015	Set bits 17, 32, and 35 of PSR (initiates quarter-word mode, double-precision underflow mode, and floating-point compatibility mode)
000000000015	Clears bits 17, 32, and 35 of PSR
440000400010	Sets bit 35 of PSR (initiates floating-point compatibility mode); all other modes remain the same (bits 17 and 32 are not interpreted when control bits 0 and 2 are not set)
040000400011	Clears bit 35 of PSR (floating-point compatibility mode); sets bit 17 of the PSR (quarter-word mode); other modes remain the same.

4.8.5. SNAPSHOT DUMP (SNAP$)

Purpose:

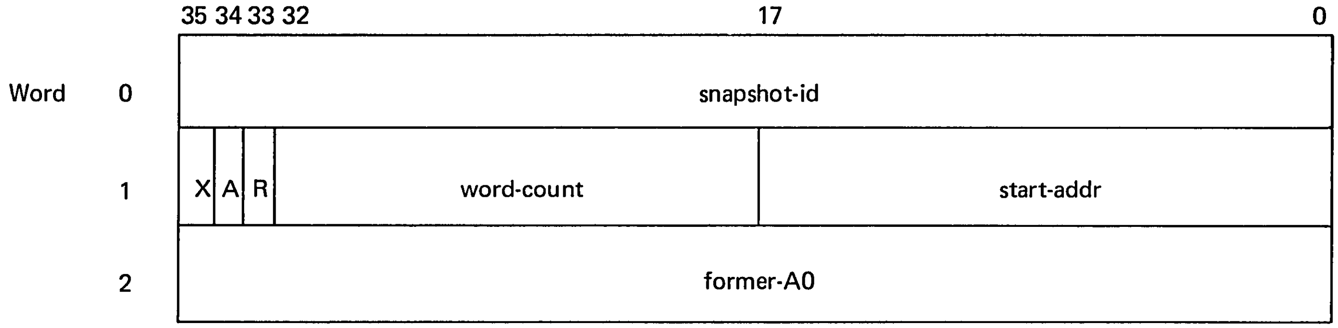
Provides a snapshot dump printout of the contents of selected control registers and program storage as an aid for debugging.

Format:

```
S    A0,pktaddr+2
L,U  A0,pktaddr
ER   SNAP$
```

Description:

Pktaddr is the address of a packet whose format is:



where:

snapshot-id — Six character Fieldata name used to identify the dump.

X,A,R — Used to designate registers to be dumped as follows:

- If bit 35=1, dump all X registers.
- If bit 34=1, dump all A registers.
- If bit 33=1, dump all R registers.

If all three bits are equal to 0, no registers are dumped.

word-count — Number of words of main storage to be dumped.

start-addr — Starting program address of the main storage area to be dumped.

former contents of-A0 — Save area for register A0. This is needed to capture the entire environment. Register A0 is restored using this value before returning control to the program.

Be careful when using the SNAP$ request in a multi-activity program because packet usage is not reentrant.

4.8.6. MASTER CONFIGURATION TABLE RETRIEVAL (MCT$)

Purpose:

To retrieve information from the master configuration table (MCT) or to read/update the program entry area in the MCT. This is a special application ER which is useful only to certain specialized programs. As a result, only those programs with a privileged account number are permitted to execute MCT$ requests. The contents of the MCT are subject to change as new executive requirements are defined. The current contents and format of the MCT are described in the *1100 Series Systems Memorandum.*

Format:

```
     L    A0,packet
     ER   MCT$
```

Description:

The contents (packet) of register A0 when executing a MCT$ request are:

S1	S2	S3	H2
status		type	buffer-addr

where:

status — Indicates the status of the request. The status codes are:

0_8 — Normal

1_8 — Invalid buffer address

2_8 — Invalid type specified

type — Specifies the type of operation to be performed. The type codes are:

0_8 — Read contents of the MCT (excluding program entry) into the specified buffer.

1_8 — Read contents of the program entry into the specified buffer. The format and content of the entry is program dependent.

2_8 — Write contents of the specified buffer into the program entry.

buffer-addr — Specifies starting address of a buffer. Buffer usage is determined by the entry in the type field.

The number of words transferred on a MCT$ request is determined by either the length of the MCT or the length of the program entry maintained in the MCT. On a read request, the size of the specified buffer must be equal to or larger than that specified in the MCT. For a write request, only the number of words specified in the program entry length field are transferred to the program entry. Bothhthe program entry and the MCT length are defined at system generation.

The program entry is an optional entry in the MCT used for recording information related to a programs application. The information recorded in this entry and its format is the responsibility of the user since only user-supplied information is recorded in the entry.

4.9. CONTINGENCIES

4.9.1. INTRODUCTION

A contingency is an abnormal condition, often associated with an interrupt, which may occur during execution of a program. Typical examples are illegal operation, unsolicited console input, and error mode.

The executive allows a program to preregister routines to process contingencies, and transfers control to the appropriate routine should any occur. In the absence of such registration, the executive provides a system standard action for each contingency type.

It should be clearly understood that in almost every case, a contingency action involves diversion of the execution path of an existing activity, rather than creation of a new activity to handle the contingency. Also, the diverted activity is normally the one to which the contingency specifically pertains.

4.9.2. CONTINGENCY TYPES AND STANDARD ACTION

Contingencies are classified into ten different contingency types. These, with their associated standard action, are listed in Table 4–2. Contingency types 10 and 12 may be further broken down into error types, (see Table 4–3). Finally, in the case of error mode (see 4.1.4), error types are broken down into many error codes; these are given in Appendix C.

The mnemonics listed in Tables 4–2 and 4–3 are standard abbreviations that appear in various system diagnostic messages; they may not be program referenced.

Contingency Type (Octal)	Mnemonic	Description	Standard Action
1	IOPR	Illegal operation (machine instruction undefined)	Error termination of offending activity.
2	IGDM	Guard mode fault	Error termination of offending activity.
3	IFOF	Floating-point overflow	Clear A,A+1 registers to zero except for the following instructions: FCL: Clear A only DFP: Clear A+1, A+2 LCF,DSF: Clear A+1 only
4	IFUF	Floating-point underflow	
5	IDOF	Divide fault (divide overflow)	
6	IRST	Restart	See checkpoint/restart (Section 17)
7	IABT	ABORT$ (also EABT$)	Program and run termination (see 4.3.2.3 and 4.3.2.4)
10	IINT	Console interrupt (see also Table 4–3)	Onsite keyin: II NOT ACTIVE operator message Remote BREAK key: unconditional program termination (see 4.6.2)
11	ITS	Test And Set (TS) instruction interrupt (real time only)	Control returned to TS instruction (see Section 16)
12	IERR$	Error mode (see also Table 4–3)	Error termination of offending activity

Notes:

(1) Contingency types 1 through 5 are hardware detected. They are discussed in *UNIVAC 1108 Multi-Processor System Processor and Storage Programmers Reference, UP-4053* (current version). Test and Set (TS) instruction operation is also hardware oriented (see 16.4).

(2) Error termination is discussed in 4.9.2.1.

(3) Arithmetic fault (types 3,4,5) A-register clearing on standard action is done by examining the a field of the offending instruction. No clearing occurs if an Execute Remote (EX) instruction was used to execute the offending arithmetic instruction.

Table 4–2. Contingency Types

Error Name	Error Type (Octal)	Mnemonic
Contingency Type 10.		
II Onsite Keyin	1	II
Remote BREAK Key	2	RBK
Contingency Type 12.		
I/O Call Error	1	I/O
Symbiont Call Error	2	SYMB
ERR$ Call (ER ERR$)	3	ERR$
Invalid or Bad ER	4	ER
Console Call Error	5	CONS
Communications Error	6	COM2
Communications Error	7	COMM
Reentrant Processor Error	10	REP

Table 4–3. Error Types

4.9.2.1. ERROR TERMINATION CONSIDERATIONS

When an activity error terminates, it does not necessarily mean immediate termination of all activities of a multi-activity program, although in practice a program is usually unable to proceed much further when it loses an activity in an error situation. Of course, any activity termination in a single activity program does mean immediate program termination.

Activity error terminations produce a diagnostic message in the run's print file defining the error, the point at which it occurred, and identification of the activity (name or id). In batch mode, a complete control register dump is provided.

When one or more activities of a program error terminate, bits are set in the run condition word (see 3.9.4) indicating this fact, and the run is marked in error. When the program ultimately terminates, further run stream processing is normally limited to processing a post-mortem dump, provided a @PMD control statement (see 11.2.1) is the next (nontransparent) control statement. A @PMD,E control statement (dump only if an error occurs) is honored in this case (but not after a normal program termination). The run is terminated after PMD processing is completed, unless a @SETC,I control statement (continue in spite of errors – see 3.9.4.1) is in effect.

4.9.3. CONTINGENCY REGISTRATION (IALL$)

Purpose:

To register a routine to handle one or more contingency types, either for the entire program or for just the requesting activity.

Format:

```
L     A0,(contingency-parameter)
ER    IALL$
```

Description:

The format of contingency-parameter is:

T1	S3	H2
selection-mask	contingency-application	contingency-routine-addr

selection-mask

A bit mask that indicates which contingencies are to be processed. The bit settings are:

Contingency Type (Octal)	Bit Set	Contingency
1	24	Illegal Operation
2	25	Guard Mode
3	26	Floating-Point Overflow
4	27	Floating-Point Underflow
5	28	Divide Fault
6	29	Restart
7	30	Abort
10	31	Console Keyin
11	32	Test and Set (Real Time Only)
12	33	Error Mode

The selection-mask field must be set to 0 for ESI contingencies or when contingencies are being cancelled. A zero mask should not be used in any other instance.

contingency-application

Specifies the scope of the operation and may take one of the following values:

0_8 – *Program.* The contingency routine being registered is to apply to all activities of the program (except ESI contingencies).

1_8 – *Activity.* The contingency routine is to apply only to contingencies pertaining to the requesting activity. In a particular contingency situation, if an applicable routine (including selection mask setting) is registered for both the offending activity and the program, then the activity routine is selected. Otherwise, the program contingency routine is selected, or if it is also not applicable, standard action occurs.

2_8 – *ESI.* The contingency routine is to apply to all contingencies during ESI completion processing for the program. ESI contingencies are handled somewhat differently from nonESI contingencies (see 4.9.5).

contingency-routine-addr

Specifies the address of the first word of the contingency routine. A zero address specifies no registration for the application specified in the contingency-application field.

A program may have as many different contingency routines registered at one time as there are activities, plus a program contingency routine, and plus an ESI contingency routine.

The same address may be registered as the contingency routine address for different applications or activities (the difference being the selection-mask settings). This does not include ESI contingencies.

A contingency registration completely cancels any previous registration for the same application (just for the same activity for activity applications).

Restart and console interrupt contingencies (types 6 and 10) are by nature not associated with any particular activity and must be registered as program contingencies (application 0). The executive may divert any activity to process these contingencies, if registered. To avoid an arbitrary activity diversion to process the II/BREAK keyins (type 10), it may be preferable to use the II$ request (see 4.6.2).

The operator keyins X and E cause actions similar to the ABORT$ and EABT$ requests (see 4.3.2.3 and 4.3.2.4, respectively). No contingency registration, however, is applicable to these keyins.

4.9.4. CONTINGENCY PROCESSING (NON-ESI)

4.9.4.1. THE CONTINGENCY ROUTINE

The first two words of the contingency routine are used as a contingency packet, in which the executive stores status information concerning the contingency prior to giving the offending acticity control to process the contingency routine that starts at the third word of the packet. The packet format is:

		S1	S2	S3	H2
Word	0	*error-type*	*error-code*	*contingency-type*	*error-addr*
	1	not used			*ER-packet-addr*
	2	(first instruction of the contingency routine)			
	n	(last instruction of the contingency routine)			

where:

error-type
See Table 4–3.

error-code
See Appendix C

contingency-type
See Table 4–2.

error-addr
The address of the offending instruction, or in the case of asynchronous contingencies (see 4.9.4.3), the address of the last instruction prior to diverting to the contingency routine. Thus, if it is desired to return to the original execution path after the contingency is processed, the error-addr must be increased by one for use as a reentry address. The error address may not be meaningful for guard mode errors because the hardware does not guarantee a valid interrupt address.

ER-packet-addr
The address of the ER packet associated with the offending instruction. Applicable only for I/O and console error types (see Table 4–3).

A contingency routine is entered with all control registers for the offending activity loaded as they were when the contingency occurred. Preservation of these registers, if they are needed, is the responsibility of the contingency routine.

Contingencies are processed serially for the entire program. While an activity is executing a contingency routine, it is considered to be in the contingency mode, and no other contingency processing is allowed to occur. For this reason, contingency processing should be kept as short as possible.

When the activity has completed contingency processing, it must notify the executive so that other contingencies may be processed. This is accomplished by executing any ER (CEND$ request – see 4.9.4.2 – is provided especially for this purpose). Once contingency mode is terminated, the contingency packet may be overwritten as the result of another contingency.

4.9.4.2. CONTINGENCY MODE TERMINATION (CEND$)

Purpose:

To notify the executive that the requesting activity has completed contingency processing.

Format:

```
ER   CEND$
```

Description:

Although any ER can terminate contingency mode, the CEND$ request is designed for that specific purpose, and is the most efficient method unless the service provided by another ER is needed.

4.9.4.3. ADDITIONAL CONTINGENCY CONSIDERATIONS

- Nested Contingencies

  A nested contingency occurs when an activity encounters a synchronous contingency from within a contingnecy handler. (A synchronous contingency is one that occurs at the same point as its cause.) All nested contingencies are given standard action for the particular contingency type, regardless of registration. If the standard action is termination, the activity is removed from contingency mode, terminated, and some other contingency (if any) is processed. Otherwise, the activity is left in contingency mode. ERR$ is considered a nested contingency, and standard error action occurs. ABORT$ and EABT$ cause standard action but if an ABORT contingency routine is registered, it will get control in the standard manner.

- Multiple (Non-nested) Contingencies

  As stated previously (see 4.9.4.1) contingencies are processed serially within a program. If multiple contingencies occur, one is processed and the rest are queued. The queue is ordered by the switching priority of the offending activity, thus assuring real time activities proper treatment. When an activity terminates contingency mode, the next contingency on the queue is processed.

  It is possible for multiple (non-nested) contingencies to occur for a single activity, due to asynchronous contingencies. (An asynchronous contingency is one which occurs at a point unrelated to its cause, such as a console interrupt or restart, or an error relating to an asynchronous ER-see 4.1.3 and 4.1.4.) In such cases, each contingency is queued individually and the activity is subject to multiple successive diversions to process each contingency in serial.

  Determination of the applicable routine (if any) is made when a contingency is actually processed, and not at time of occurrence (that is, while a contingency is queued, an IALL$ request (see 4.9.3.1) may have been executed and changed the registration).

  When an activity terminates for any reason, any contingencies queued for it are discarded.

- Test-And-Set (TS) contingency processing is available only to real time activities. This type of contingency is particularly prone to interlock situations, and the programmer should use caution to ensure against such problems. Note that a TS conflict within a contingency routine is given standard action, while a TS conflict outside of a contingency routine, when the program is in contingency mode, results in stalling the conflicting activity. See 16.4 for more details on real time TS processing.

- A checkpoint is not allowed while a program is in contingency mode if a restart contingency is registered.

- For ABORT$ or EABT$ requests (see 4.3.2.3 and 4.3.2.4, respectively), the contingency is not honored until all activities of the program have terminated. The user then regains control at the contingency routine with a new activity.

4.9.5. ESI CONTINGENCIES

The communications handler provides the user with the capability of processing various contingency conditions that might occur while executing an ESI activity. To establish an ESI contingency, the user real time program must register the contingency via the IALL$ request (see 4.9.3). All contingency types that can occur within an ESI activity are processed by the specified ESI contingency routine.

The format of the ESI contingency packet is:

		S1	S2	S3	H2
Word	0	*error type* (7_8)	*error-code*	*contingency-type*	*error-addr*
	1	*TS-indicator*	not used		*ER-packet-addr*
	2	(first instruction of the contingency routine)			
	n	(last instruction of the contingency routine)			

Word 0:

error-type — Error-type is 7_8 for communications

error-code — The error-codes are:

60_8 — Indicates contingency type 1 through 5

61_8 — ESI ACT$ or ADACT$ request error

62_8 — ESI CADD$ or ADACT$ request error

63_8 — Invalid ER request

64_8 — ESI time-out

contingency-type — The contingency-type codes are:

1_8 — Invalid operation

2_8 — Guard mode

3_8 — Floating-point overflow

4_8 — Floating-point underflow

5_8 — Divide fault

12_8 — Error mode (see error-code). Applicable only to 61_8–63_8

error-addr — Same as for nonESI (see 4.9.4.1) except that for error mode contingencies (type 12) operation is truly reentrant (no incrementation is needed).

Word 1:

TS-indicator — Set equal to 1 by the executive prior to giving control to the contingency routine.

ER-packet-addr — The address of the ER packet associated with the offending instruction. Applicable only to contingency type 12 (see Tables 4–2 and 4–3).

As for nonESI contingencies, ESI contingencies are initiated serially. Prior to terminating contingency mode, however, the routine may enable contingency processing on another CPU by clearing the TS indicator (S1, word 1) in the contingency packet. This indicator serves to protect the packet contents from being overwritten until the contingency routine has had a chance to retrieve the information pertinent to the contingency. Termination of the ESI contingency mode also enables other contingency processing.

Should another contingency occur while in contingency mode, the ESI activity is terminated and the communication line associated with that activity is deactivated for I/O operations. The following message is issued to the console and the master RUN-LOG as an error message to indicate the terminating condition:

```
RUN ID Sxxx/Uxxx  ESI TERMINATION (error)
```

The S and U fields of the message indicated the subsystem and unit numbers respectively and the error field gives the error code. The first reference to the deactivated terminal by the real time program causes a nonESI contingency for the referencing activity (error type 7_8, code 10_8).

Terminate ESI contingency mode normally, only those executive requests specified for normal ESI activities may be used: EXIT$, ACT$, CADD$, and ADACT$. Any reference other than those indicated above result in a contingency within a contingency, causing terminal deactivation as described in the preceding paragraphs.

ESI contingencies are independent of non ESI contingencies. A program may process ESI and non ESI contingenies concurrently.

5. SYMBIONT INTERFACE REQUESTS

5.1. INTRODUCTION

The executive system contains a set of routines which provide an interface between the user and any supported unit record device. This set of routines is called the symbiont complex. These routines can be divided into two logical groups:

□ Symbionts (also called device routines)

□ Symbiont interface routines

5.1.1. SYMBIONTS

Symbionts (device routines) are available for all standard equipment. Supported equipment includes such onsite devices as high speed card readers, punches, printers, UNIVAC 9000 Series Systems; such batch remote site terminals as UNIVAC 9000 Series Systems and DCT-2000; and such demand remote terminals as teletypewriter models 33 and 35, Friden, UNISCOPE 100 and 300, DCT-500, and DCT-1000. For all batch devices, data is buffered in SDF format using mass storage to provide an effective linkage between the high speed of the CPU and the low speed unit record devices. Due to the conversational nature of demand processing, input data from demand terminals is not buffered to mass storage except for paper tape input.

During systems generation, one or more output devices are associated with each of the input devices. This logical linking of output to input devices is called device association throughout this section. The result of this association is that output files created as the result of an execution are normally outputted on only one of the devices associated with the input device which initiated the run stream. In cases where an output device is unavailable, or busy, or where a specific output device is desired, the association can be overriden for a specific file by means of the @SYM control statement.

Input to the system is separated by the @RUN control statement. As each @RUN control statement is encountered, a run file is created and information is extracted by the coarse scheduler for run scheduling. The input symbionts also interpret the @ELT,D, @DATA, @END, @FILE, and @ENDF control statements to determine if a @RUN control statement is the beginning of another input run stream or part of a file or element in the current run stream. The @COL and @ENDCL control statements are interpreted to determine if the mode of the card reader should be changed.

All files created or processed by the symbiont complex are in SDF format (see 24.2.3) and can be directly processed by either the input interface routines or by the output symbionts.

Functions which control the format of the output are inserted into the symbiont output file by means of an executive request. These functions vary according to the output device to which the file is being directed. As the control parameters are submitted, they are placed into the appropriate output file, and interpreted when the file is being processed by a symbiont (see 5.4).

The data in input files created from ASCII devices is in ASCII and the data in input files created from Fieldata devices is in Fieldata. The user may request data from these files in either ASCII or Fieldata (see 5.2) and the necessary conversion is done. Output files may be created in either ASCII or Fieldata (see 5.3). Data which is in ASCII is converted to Fieldata for Fieldata devices and requires no translation for ASCII devices. The converse holds true for data which is in Fieldata. This manipulation of data requires no special action by the user other than to make the proper executive request as described in this section. See Appendix D for translation tables.

5.1.2. SYMBIONT/USER INTERFACE ROUTINES

The symbiont user interface routines provide for data transfers in either Fieldata or ASCII. A complete set of executive requests is provided for each mode. The data transfered is always Fieldata when the Fieldata requests are used and is always ASCII when the ASCII requests are used. The user interface routines are avilable through the following executive requests.

Fieldata Executive Requests	ASCII Executive Requests
READ$	AREAD$
PRINT$	APRINT$
PUNCH$	APUNCH$
READA$	AREADA$
PRNTA$	APRNTA$
PNCHA$	APNCHA$
PRTCN$	APRTCN$
PCHCN$	APCHCN$
PRTCA$	APRTCA$
PCHCA$	APCHCA$
TREAD$	
CLIST$	

When the letter A appears as the last alphabetic character, the request pertains to an alternate file (defined in succeeding paragraphs). The letter A appearing as the first character indicates an ASCII operation. ASCII and Fieldata executive requests may be interspersed in any order. For each executive request, the user specifies the storage area in his program for the data transfer. In addition, when using executive requests for alternate files, the user must specifiy the filename in Fieldata.

The system automatically initiates three symbiont files, allowing three normal operations as follows:

Run files (READ$ file) — contains input images accessed by means of READ$ or AREAD$

Print file (PRINT$ file) — contains output images produced by PRINT$ or APRINT$

Punch file (PUNCH$ file) — contains output images produced by PUNCH$ or APUNCH$

Each of the three basic interface functions (read, print, and punch) is capable of multiple file operation. The user may define files other than the three automatically initialized by the system. The user may assign a file and direct the normal print or punch output to this file by means of the @BRKPT control statement (see 3.6.2). The user may assign a file and direct only specific print or punch output to this file by means of the alternate executive output requests (such as, the PRNTA$ request). The user may also assign a previously created file of input images and read these images in the normal mode by prior use of the @ADD control statement (see 3.9.1) or directly input from the file by means of this alternate input executive requests (such as the READA$ request). These user-defined and assigned files are called alternate files.

The @ADD control statement is used to direct the READ$ or AREAD$ requests to obtain images from the file indicated by the @ADD control statement instead of images from the system initiated run file. Subsequent READ$ or AREAD$ requests obtain images from the @ADD file until it is exhausted, at which time images again are obtained from the system initiated run file. Nesting of @ADD control statements is permitted.

The @BRKPT control statement is used to direct PRINT$/APRINT$ or PUNCH$/APUNCH$ requests to place images in a file defined by the @BRKPT control statement instead of the system initiated print or punch files. Images continue to be placed in the user specified files until another @BRKPT control statement is encountered. During run termination, the normal print image stream is always returned to the system-initiated print file. The @BRKPT control statement also may be used to close user-defined alternate files.

The output control requests, such as PRTCN$, provide specific control information describing output formatting to the device routines. The output control requests also provide a means of advising the device operator of any special action required.

A more detailed discussion of the total capabilities of the symbiont interface routines is given in the paragraphs that follow.

5.2. OBTAINING INPUT IMAGES

5.2.1. READING FIELDATA IMAGES (READ$)

Purpose:

Obtains an image in Fieldata from the run stream located in the run file.

Format:

```
L       A0,(EOF-return-addr,buffer-addr)
ER      READ$
```

These instructions can be generated by the procedure call:

```
R$EAD   (EOF-return-addr,buffer-addr)
```

Parameters:

EOF-return-addr	Address to which control is transferred when a control statement is encountered.
buffer-addr	Address of the input buffer into which the Fieldata image is placed.

Description:

If the input image is in quarter-word ASCII, READ$ converts it to Fieldata.

Normal input image length may be up to 14 words but images from an @ADD or @START file may be any length.

Input images must be noncontrol statement images except for the CLIST$ control statements (see 5.5) or processor control statements (in INFOR format – see 9.6).

After the image is transferred to the input buffer, control is returned to the address following the READ$ request.

Upon return from an @EOF control statement (see 10.3.2), bits 5–0 of register A0 contain the sentinel character that appears in column 6 of the @EOF control statement and bit 35 is not set.

If the EOF return is caused by an @ADD,E control statement (see 3.9.1), H2 of register A0 is set to zero.

Upon normal return from a READ$ request, H2 of register A0 contains the number of words transferred. The meaning of any bits set in H1 of register A0 is as described in Table 5–1.

If the run is being made in the demand mode, the program is normally placed in a wait state until the READ$ request is satisfied from the demand terminal (see Section 12).

Bit Set	Decription
35	Abnormal return taken because a control statement cannot be passed. Any further attempt to do READ$ request or a TREAD$ request within this program causes an error termination (unless the request is preceded by a CSF$ request with a @ADD statement).
34	Currently reading from a file or an element introduced by an @ADD control statement.
33	Set on an EOF return when at the end of an @ADD file or element and an E option was used on the @ADD control statement.
31	Image is in INFOR format (see 9.6).
30	Used if bit 31 is set; indicates that more INFOR – formatted words are to be read (see 9.6).
23–18	Used if a statement listed by a CLIST$ request (see 5.5) was encountered; contains the CLIST$ index value.

Table 5–1. Bit Settings In Control Register A0 For A READ$ Request

5.2.2. READING ASCII IMAGES (AREAD$)

Purpose:

Obtains an image in quarter-word ASCII from the run stream located in the run file.

Format:

```
L       A0,(EOF-return-addr,buffer-addr)
ER      AREAD$
```

These instructions can be generated by the procedure call:

```
A$READ   (EOF-return-addr,buffer-addr)
```

Parameters:

The interpretation of the parameters is identical to that for the READ$ request (see 5.2.1).

Description:

AREAD$ operation is identical to READ$ (see 5.2.1) except that input image length may be up to 20 words (@ADD and @START file images may be any length).

5.2.3. FIELDATA IMAGES – ALTERNATE FILE (READA$)

Purpose:

Obtains an image in Fieldata from a user-specified file.

Format:

```
L,U     A0,pktaddr
ER      READA$
```

These two instructions may be generated by the procedure call:

```
R$EADA   pktaddr
```

Description:

Pktaddr is the address of a three-word packet whose format is:

		H1	H2
Word	0	EOF-return-addr	buffer-addr
	1	12-character-Fieldata-filename	
	2		

Word 0

EOF-return-addr — The address to which control is returned when no more images exist in the file.

buffer-addr — The address of the input buffer into which the Fieldata image is placed.

If the input image is in quarter-word ASCII, the READA$ request converts it to Fieldata.

Upon normal return from a READA$ request, register A0 contains the number of words transferred. Images may be any length and the caller must be careful if the image length is longer than anticipated. Normal image length is 14 words.

After the image is transferred to the input buffer, control is returned to the address following the READA$ request.

The file named in the packet must have been assigned prior to the first READA$ request and must be in SDF format.

When the file is exhausted, no image is available to transfer, and the caller regains control at the EOF return address.

See 3.6.2 for the use of the @BRKPT control statement with read alternate files.

5.2.4. ASCII IMAGE – FROM AN ALTERNATE FILE (AREADA$)

Purpose:

Obtains an image in quarter-word ASCII from a user-specified file.

Format:

```
A,U    A0,pktaddr
ER     AREADA$
```

These two instructions may be generated by the procedure call:

```
A$READA   pktaddr
```

Description:

The interpretation of the parameters is identical to that for the READA$ request (see 5.2.3).

The AREADA$ operation is identical to READA$ request (see 5.2.3) except that the normal image length may be up to 20 words.

5.2.5. FIELDATA IMAGES – CONVERSATIONAL MODE (TREAD$)

Purpose:

Displays the Fieldata message supplied and obtains in Fieldata the response. This request requires less overhead than an individual PRINT$ request followed by a READ$ request and should be used for demand processing.

Format:

```
L,U    A0,pktaddr
ER     TREAD$
```

Description:

Pktaddr is the address of a two-word packet whose format is:

		T1	S3	H2
Word	0	line-spacing	image-length	output-buffer-addr
	1	EOF-return-addr		input-buffer-addr

Word 0

line-spacing — The number of lines to space before displaying the message. No spacing is performed after displaying the message.

image-length — The length in words of the message.

output-buffer-addr — The address of the output buffer from which the Fieldata message is obtained.

Word 1

EOF-return-addr — See READ$ request (5.2.1).

input-buffer-addr — The address of the input buffer into which the Fieldata image is placed.

The program is normally placed in a wait state until both the output and the input operations are accomplished at the demand terminal. During the wait period, the program is a prime candidate to be swapped to auxiliary storage.

When images that are obtained from a file introduced by an @ADD control statement, neither the output message nor the images obtained are displayed.

Normal return is identical to that for the READ$ request (see 5.2.1).

5.3. TRANSFERRING OUTPUT IMAGES

5.3.1. PRINTING FIELDATA IMAGES (PRINT$)

Purpose:

Places a Fieldata image into the system defined print file.

Format:

```
     L       A0,(PF line-spacing,nbr-of-words,image-addr)
     ER      PRINT$
```

These two instructions may be generated by the procedure call:

```
     P$RINT   (PF line-spacing,nbr-of-words,image-addr)
```

Parameters:

PF	An assembler FORM directive defined as PF FORM 12, 6, 18.
line-spacing	Number of lines to space before printing this image.
nbr-of-words	Number of data words in this image.
image-addr	Address where the Fieldata image is obtained.

Description:

The allowable values for line spacing are 0_8 to 3777_8. If the value of line spacing is greater than the number of lines remaining on the present page, the image is printed on the first printable line on the next page. If the value in line spacing is –0 (7777_8) the image is always printed on the first printable line of the next page. The first printable line on a page is defined by means of the print control margin function (see 5.4.1).

The number of words in the image is limited only by the number of characters that can be printed on the device that prints the file. For example, if the file is to be printed on a 132 character/line high speed printer, the maximum value for the nbr-of-words parameter is 22_{10}.

The control statement @SYM PRINT$ can be used to direct the current system-defined print file to a device other than the device indicated by device association. The queueing of the print file is held until it is closed by @BRKPT control statement or the run is closed.

The @BRKPT control statement is used to close and queue for printing all system defined print files and the @SYM control statement is not necessary if the user wants the file to go to the devices specified by device association.

In demand mode, the program is normally placed in a wait state until the output is accomplished.

5.3.2. PRINTING ASCII IMAGES (APRINT$)

Purpose:

Places a quarter-word ASCII image into the system-defined print file.

Format:

```
     L       A0,(PF line-spacing,nbr-of-words,image-address)
     ER      APRINT$
```

These two instructions may be generated by the procedure call:

```
A$PRINT   (PF line-spacing,nbr-of-words,image-address)
```

Parameters:

The parameters for the APRINT$ request are identical to those for the PRINT$ request (see 5.3.1).

Description:

All formats and limitations are the same as for the PRINT$ request (see 5.3.1) except a standard 132-character/line printer prints an image of 33_{10} words.

5.3.3. FIELDATA IMAGES – ALTERNATE PRINT FILE (PRNTA$)

Purpose:

Places a Fieldata print image into a user-defined print file.

Format:

```
L,U     A0,pktaddr
ER      PRNTA$
```

These two instructions may be generated by the procedure call:

```
P$RNTA   pktaddr
```

Description:

Pktaddr is the address of a three-word packet whose format is:

		T1	S3	H2
Word	0	line-spacing	word-count	buffer-addr
	1	12-character-Fieldata-filename		
	2			

where:

The meaning of the line-spacing, word-count,, and buffer-addr parameters are identical to those for the PRINT$ request (see 5.3.1) and the restrictions that apply to PRINT$ also apply to PRNTA$.

Description:

If an alternate file has been assigned prior to the first executive request for the file, the executive does not queue the file for output. If the file has not been assigned to the run, the executive assigns the file, and when the file is closed (@BRKPT or @FIN control statement) the file is automatically queued to the output device determined by device association. If the alternate output file was assigned by the user, the file must be closed with a @BRKPT control statement (see 3.6.2) and queued for output with a @SYM control statement (see 3.6.3) before the run terminates for the file to be properly outputted. All files must be catalogued before they can be referenced by the @SYM control statement.

5.3.4. ASCII IMAGES – ALTERNATE PRINT FILE (APRNTA$)

Purpose:

Places a quarter-word ASCII image into a user-defined print file.

Format:

```
L,U    A0,pktaddr
ER     APRNTA$
```

These two instructions may be generated by the procedure call:

```
A$PRNTA   pktaddr
```

Description:

The interpretation of the parameters is identical to that for the PRNTA$ request (see 5.3.3).

The APRNTA$ operation is idential to PRNTA$ (see 5.3.3) and the filename must be in Fieldata.

5.3.5. PUNCHING FIELDATA IMAGES (PUNCH$)

Purpose:

Places a Fieldata image into the system defined punch file.

Format:

```
L      A0,(nbr-of-words,image-addr)
ER     PUNCH$
```

These two instructions may be generated by the procedure call:

```
P$UNCH   (nbr-of-words,image-addr)
```

Parameters:

nbr-of-words	Number of words of data in this image.
image-addr	Address of the buffer where the image is obtained.

Description:

The number of words in the image must not exceed 63_{10} and is also limited by the number of characters that can be punched on the device. If the image is to be punched in 80-column Hollerith code, the maximum image length is 14_{10} but shorter images may be specified and are blank-filled before punching. If the image is 14_{10} words long, the last four characters must be blanks.

If the images are to be punched in column binary (see 5.4.5), an image length of 27_{10} must be used, and any column that is not required for data must be zero filled.

The control statement @SYM PUNCH$ can be used to direct the current system defined punch file to a device other than the device indicated by device association. The queueing of the punch file is held until it is closed by the @BRKPT control statement or the run is closed.

The @BRKPT control statement is used to close and queue for punching all system-defined punch files and the @SYM control statement is not necessary if the user wants the files to go to the device specified by device association.

5.3.6. PUNCHING ASCII IMAGES (APUNCH$)

Purpose:

Place quarter-word ASCII images into the system-defined punch file.

Format:

```
L       A0,(nbr-of-words,image-addr)
ER      APUNCH$
```

These two instructions may be generated by the procedure call:

```
A$PUNCH   (nbr-of-words,image-addr)
```

Parameters:

The parameters for the APUNCH$ request have the same meaning as the parameters for the PUNCH$ request (see 5.3.5).

Description:

The number of words in the image must not exceed 63_{10} and is also limited by the number of characters that can be punched on the device. If the image is to be punched in 80-column Hollerith code, the maximum image length is 20_{10}. If the image is to be punched in column binary (see 5.4.5) an image length of 27_{10} must be used, and any column not punched must be zero filled.

If the user is punching column binary, there is no difference between the PUNCH$ and APUNCH$ requests.

5.3.7. FIELDATA IMAGES – ALTERNATE PUNCH FILE (PNCHA$)

Purpose:

Place a Fieldata image into a user-defined punch file.

Format:

```
L,U     A0,pktaddr
ER      PNCHA$
```

These two instructions can be generated with the procedure call:

```
P$NCHA   pktaddr
```

Description:

Pktaddr is the address of a two word packet in the format:

		T1	S3	H2
Word	0	not-used	word-count	buffer-addr
	1	12-character-Fieldata-filename		
	2			

Word 0

The meaning of word-count and buffer-addr are the same as for the PUNCH$ request (see 5.3.5) and all the restrictions which apply to PUNCH$ apply to PNCHA$.

All rules for queueing for output of punch alternate files are the same as the rules for queueing of print alternate files (see 5.3.3).

PNCHA$ or APNCHA$ is a convenient method of building SDF-formatted files to be used later as input files in the same run or in a subsequent run. These files may be partial run streams introduced by an @ADD control statement (see 3.9.1), complete run streams referenced by a @START control statement (see 3.4.3), data to be referenced by READA$ or AREADA$ requests, or read directly by the user program.

5.3.8. ASCII IMAGES – ALTERNATE PUNCH FILE (APNCHA$)

Purpose:

Place a quarter-word ASCII punch image into a user-defined punch file.

Format:

```
L,U     A0,pktaddr
ER      APNCHA$
```

These two instructions can be generated by the procedure call:

```
A$PNCHA   pktaddr
```

Parameters:

The interpretation of parameters is identical to that for the PNCHA$ request (see 5.3.7) and the filename must be in Fieldata.

Description:

APNCHA$ operation is identical to PNCHA$ operation (see 5.3.7).

5.4. OUTPUT CONTROL FUNCTIONS

5.4.1. FIELDATA CONTROL FUNCTIONS – PRINT FILE (PRTCN$)

Purpose:

Specify a Fieldata control function to a print device routine for a print file.

Format:

```
L,      A0,(image-length,buffer-addr)
ER      PRTCN$
```

Parameters:

image-length	The length in words of the Fieldata control image.
buffer-addr	Address of the buffer from which the Fieldata control image is obtained.

Description:

The image specified in the packet consists of one or more control functions. Each control function is in Fieldata and begins with a letter followed by subfields. A comma is the field (or subfield) separator and a period terminates each control function string.

Table 5–2 lists the print control functions and their formats.

Control Function Format	Description
L,nn	Space the printer to logical line number nn-1.
H,options,page,text	Initiate printing page headings where: options The available options are: N – Do not print heading X – Suppress printing page number and date page The page number of the first page with this heading. If blank and there is no X or N option, the page numbering continues with one greater than previous page. text The heading text (maximum of 16 words) When the margin function is used, a page alignment procedure is initiated with the page length parameter. If a top margin of zero is specified, the heading is never printed.
S,text	Special forms request for processing the print file. The text is a maximum of 48 characters. When the function is encountered for onsite printers, the text is displayed on the operator's console. For batch remote devices, the text is displayed on a remote printer.
M,length,top,bottom	Margin setting information for readjusting page length, and top and bottom margins. where: length – number of lines to be printed per page top – number of blank lines used for top margin bottom – number of blank lines used for bottom margin

Table 5–2. Print Control Functions

Standard page definition is 66 lines per page with a top margin of six lines and a bottom margin of three lines, leaving 57 printable lines.

5.4.2. ASCII CONTROL FUNCTIONS – PRINT FILE (APRTCN$)

Purpose:

Specify an ASCII control function to a print device routine for a print file.

Format:

```
     L       A0,(image-length,buffer-addr)
     ER      APRTCN$
```

Parameters:

image-length	The length in words of the ASCII control image.
buffer-addr	Address of the buffer from which the ASCII control image is obtained.

Description:

The APRTCN$ request is identical to the PRTCN$ request except that the image is in quarter-word ASCII instead of Fieldata (see 5.4.1).

5.4.3. FIELDATA CONTROL FUNCTION – ALTERNATE PRINT FILE (PRTCA$)

Purpose:

Specify a Fieldata control function to a print device routine for an alternate print file.

Format:

```
     L       A0,(image-length,buffer-addr)
     ER      PRTCA$
```

Parameters:

image-length	The length in words of the Fieldata control image.
buffer-addr	The address of the buffer from which the Fieldata control image is obtained.

Description:

The PRTCA$ request is identical to the PRTCN$ request (see 5.4.1) except that the first two words in the buffer specified by the buffer address in the packet must be the 12-character Fieldata filename of the print alternate file to which the control image is directed. The remainder of the image is the control information in Fieldata (see 5.4.1).

5.4.4. ASCII CONTROL FUNCTIONS – ALTERNATE PRINT FILE (APRTCA$)

Purpose:

Specify an ASCII control function to a print device routine for an alternate print file.

Format:

```
     L       A0,(image-length,buffer-addr)
     ER      APRTCA$
```

Parameters:

image-length	The length in words of the ASCII control image.
buffer-addr	The address of the buffer from which the ASCII control image is obtained.

Description:

The APRTCA$ request is identical to the PRTCN$ request (see 5.4.1) except that the first two words in the buffer specified by the buffer address in the packet must be the 12-character Fieldata filename of the print alternate file to which the control image is directed. The remainder of this image is the control information in quarter-word ASCII (see 5.4.1).

5.4.5. FIELDATA CONTROL FUNCTIONS – PUNCH FILE (PCHCN$)

Purpose:

Specify a Fieldata control function to a punch file routine for a punch file.

Format:

```
L       A0,(image-length,buffer-addr)
ER      PCHCN$
```

Parameters:

image-length	The length in words of the Fieldata control image.
buffer-addr	The address of the buffer from which the Fieldata control image is obtained.

Description:

The image specified in the packet consists of one or more control functions. Each control function is in Fieldata and begins with a letter followed by subfields. A comma is the field (or subfield) separator and a period terminates each control function string.

The punch control functions and their formats are:

S,text — Special forms request for processing the punch file. The text is a maximum of 54 characters. When the function is encountered for onsite punches, the text is displayed on the onsite operator's console. For batch remote devices, the text is displayed on the remote printer as soon as it is idle.

C,B — Switch the mode of punching to column binary

C,E — Switch the mode of punching to 80-column (Hollerith)

5.4.6. ASCII CONTROL FUNCTION – PUNCH FILE (APCHCN$)

Purpose:

Specify an ASCII control function to a punch device routine for a punch file.

Format:

```
L       A0,(image-length,buffer-addr)
ER      APCHCN$
```

Parameters:

image-length	The length in words of the ASCII control image.
buffer-addr	The address of the buffer from which the ASCII control image is obtained.

Description:

The APCHCN$ request is identical to the PCHCN$ request except that the image is in quarter-word ASCII rather than Fieldata (see 5.4.5).

5.4.7. FIELDATA CONTROL FUNCTIONS – ALTERNATE PUNCH FILE (PCHCA$)

Purpose:

Specify a Fieldata punch control image to a punch device routine for an alternate punch file.

Format:

```
    L       A0,(image-length,buffer-addr)
    ER      PCHCA$
```

Parameters:

image-length	The length in words of the Fieldata control image.
buffer-addr	The address of the buffer from which the Fieldata control image is obtained.

Description:

The PNCHA$ request is identical to the PCHCN$ request (see 5.4.5) except that the first two words in the buffer specified by the buffer address in the packet must be the 12-character Fieldata filename of the alternate punch file to which the control image is directed. The remainder of the image is the control information in Fieldata (see 5.4.5).

5.4.8. ASCII CONTROL FUNCTION–ALTERNATE PUNCH FILE (APCHCA$)

Purpose:

Specify an ASCII punch control image to a punch device routine for an alternate punch file.

Format:

```
    L       A0,(image-length,buffer-addr)
    ER      APCHCA$
```

Parameters:

image-length	The length in words of the ASCII control image.
buffer-addr	The address of the buffer from which the ASCII control image is obtained.

Description:

The APNCHA$ request is identical to the PCHCN$ request (see 5.4.5) except that the first two words of the buffer specified by the buffer address in this packet must contain the 12-character Fieldata filename of the file to which the control image is directed. The remainder of the image is the control information in quarter-word ASCII (see 5.4.5).

5.5. LISTING USER-DEFINED CONTROL STATEMENTS (CLIST$)

Purpose:

Allows the user to define his own set of control statements and register them with the executive. On subsequent READ$ requests, the calling program is given special notification when such statements are encountered. A control statement is defined as any input image which has a master-space (7–8 multipunch – the @ character) in the first character position (column one on a punched card).

Format:

```
L      A0,list-designator
ER     CLIST$
```

Parameters:

list-designator

Directs the executive to the particular list of control statemens the user wishes to receive. This designator may be either the address of a list in which case A0 takes the format

H1	H2
zeroes	list-addr

or it may be the six-character Fieldata, alphanumeric name of one of the executive-defined lists (that is, CFOR).

Description:

The list may contain a maximum of 62 one-word alphanumeric Fieldata control statement names. The master space is assumed and does not appear as part of the name. These names appear left-justified and space filled and are followed by a list terminator of ±0. If a ±0 is not supplied as a last item in the list, a minus zero (–0) is automatically supplied in place of the sixty-third name.

Each name in the list has an associated index value which corresponds to its position in the list. This index value is returned on a READ$ or AREAD$ request (see 5.2.1) in bits 23–18 of register A0. The image is placed in the buffer specified in the READ$ or READ$ packet and is in either Fieldata of ASCII depending upon the call (see 5.2.1 and 5.2.2).

When operating in the CLIST$ mode, only those control statements in the list are passed to the user. The executive always handles all transparent control statements, including the @ADD, @JUMP, @SETC, and @TEST control statements, and does not terminate the CLIST$ mode if they are encountered. The @FIN and @EOF control statements cause an end-of-file return, and the @FIN control statement terminates CLIST$ mode.

If the list terminator is a plus zero (+0), CLIST$ mode halts as soon as a READ$ request encounters a nontransparent control statement that is not in the list and the user is given an abnormal return (see 5.2.1).

If the list terminator is minus zero (–0), all nontransparent control statements not in the list are bypassed. When an @ENDX or @FIN control statement is encountered, the CLIST$ mode is terminated.

When operating in both the @ADD and CLIST$ modes, the @ADD mode is terminated if an unacceptable control statement is encountered.

When in the CLIST$ mode, the @ENDX control statement is automatically assumed to be part of the list with an index value of 77_8

If the user attempts to read an image by means of the TREAD$ request and a control statement is rejected, the output image is reprinted.

CLIST$ mode may be terminated by a subsequent CLIST$ request with word 0 of the list specified equal to minus zero.

If the user program terminates prior to the CLIST$ mode termination, the executive continues to read the control statement until the normal manner of ending the CLIST$ mode occurs. This allows PMD's even when the program has not read all the control statements.

5.6. FIELDATA AND ASCII TRANSLATION

Tables D–1 and D–2 (see Appendix D) define the software translation between ASCII and Fieldata codes as used by the language processors and the symbiont interface routine.

6. INPUT/OUTPUT DEVICE HANDLERS

6.1. INTRODUCTION

This section describes I/O device handlers which interface with mass storage, magnetic tape, low speed onsite devices, and special handling of peripheral devices. The communications device handlers are discussed in Section 15.

The I/O packet structure, executive requests (ER's) and procedures which are applicable to both magnetic tape and mass storage devices are presented. These are followed by details relating to specific magnetic tape and mass storage applications. Finally, special device handlers are described.

6.1.1. BASIC I/O EXECUTIVE REQUEST

Magnetic tape and mass storage files are accessed through the packet mode using an executive request, and register A0 loaded with the packet address as follows:

```
     L,U    A0,pktaddr
     ER     entrance-tag
```

Parameters:

pktaddr — Address of the I/O packet (see Figure 6–1). The length of the request packet can vary from four to eight words, depending upon the operation desired.

entrance-tag — The available entrance-tags are as follows:

IO$ (see 6.3.3.)

IOI$ (see 6.3.4.)

IOW$ (see 6.3.5.)

IOWI$ (see 6.3.6.)

IOXI$ (see 6.3.7.)

Word	S1	S2	S3	H2
0	filename			
1				
2	0		int-act-id	interrupt-activity-addr
3	*status*	function	*AFC* (tape-only)	*final-word-count-returned-by-I/O*
4	G | word-count			buffer-addr
5	0		drum-addr	
6	search-sentinel			
7	0		*search-find-drum-addr*	

Figure 6–1. I/O Packet, Mass Storage and Magnetic Tape Peripherals

Words 0 and 1

The internal filename used in all references to the file. This name is either the same as some external filename of the @ASG control statement or is attached to an external filename by a @USE control statement (see 2.5.2).

Word 2

int-act-id — The numeric identity (1–35) used to identify the interrupt activity if synchronization is intended with some other activity. Must be zero if no activity-id is desired (IOI$ and IOWI$ only).

interrupt-activity-addr — The address at which the user program receives control upon occurrence of an interrupt signifying completion of the I/O operation (IOI$ and IOWI$ only).

Word 3

status — The status of the last function performed. Must be positive when it refers to an executive request (see 6.10).

function — Denotes the function to be performed (see Table 6–1).

AFC — The abnormal frame count value for magnetic tape files only (see 6.4.2.4).

final-word-count-returned-by-I/O — For any function involving data transfer, this field contains the exact number of words read or written. For magnetic tape or the end of a drum file, this number may differ from the access word.

Word 4

An I/O access word; or for GW$, SCR$, and SCRB$ functions, this word contains the number of access words in H1 and the address at which the string of access words begins in H2.

G designator (bits 34 and 35) to increment or decrement buffer-addr by 1 for each word transferred.

00_2	– increment
10_2	– decrement
01_2 and 11_2	– no increment or decrement.

word-count	number of words to transfer.
buffer-addr	main storage address at which transfer is to begin.

Word 5

For magnetic drum files, this word contains the logical mass storage address at which the described I/O operation is to start. This address is relative to the start of the mass storage file; the handler determines the absolute position. For FASTRAND-formatted mass storage files, the address is the start of a sector, and consecutive addresses are 28 words apart.

Word 6

The identifier word for search operations. This is applicable only to mass storage files.

Word 7

The find address for a mass storage search is returned in this word. The address is relative to the start of the file. This is applicable only to mass storage files.

Function	Octal	Symbol
Write	10	W$
Write end of file	11	WEF$
Contingency write	12	CW$
Skip write	13	SW$
Gather write	15	GW$
Acquire FASTRAND	16	ACQ$
Absolute write	17	ABSW$
Read	20	R$
Read backward	21	RB$
Read and release	22	RR$
Release	23	REL$
Block read drum	24	BRD$
Read and lock	25	RDL$
Unlock	26	UNL$
Track search all words	30	TSA$
Track search first word	31	TSF$
Position search all words	32	PSA$
Position search first word	33	PSF$
Search drum	34	SD$
Block search drum	35	BSD$
Search read drum	36	SRD$
Block search read drum	37	BSRD$
Rewind	40	REW$
Rewind with interlock	41	REWI$
Set mode	42	SM$
Scatter read	43	SCR$
Scatter read backward	44	SCRB$
Absolute read	47	ABSR$
Move forward	50	MF$
Move backward	51	MB$
Forward space file	52	FSF$
Backspace file	53	BSF$

Table 6–1. Octal and Mnemonic I/O Codes Defined In SYS$*RLIB$

6.1.2. INTERRUPT ACTIVITY

The interrupt activity is the same as other registered activities using the FORK$ function (see 4.3.1.1) except for the following:

- The priority of the activity is raised to the highest possible level within the program class of the user program; that is, for a batch user program, these I/O completion activities receive control before any other batch program activity.

- The interrupt routine is not interrupted in favor of any other similar activity of the same program. All are queued in a first-in/first-out list of all programs without regard to priority within the class.

- Any executive request removes the interrupt activity from the high priority list and returns it to the user program's priority.

- The control register subset in the interrupt routine is limited to registers X11, A0 through A5, and R1 through R3. Register A0 contains the I/O packet address. If the suppress recovery mode is set, register A1 is loaded with the status word from the subsystem external interrupt. No other register contents are defined.

- In the absence of any other executive request, the normal program status can be restored by using the UNLCK$ request (see 6.3.8).

When multiprogramming, every attempt is made to provide proper switching by allowing immediate access to the amount of computation required to initiate a new I/O option following any other I/O operation. The difficulty lies in preventing abuse of the high priority assigned to interrupt activities. The available facility is limited to initiate a new I/O operation after having checked the status of the previous I/O operation.

6.1.3. QUEUEING AND UNIT CONTROL

When an I/O operation is referenced, the handler controlling the desired device is entered. The handler considers the request and queues it for the particular subsystem. When the requested device becomes free, and entry is removed from the subsystem queue and the handler is entered at the appropriate point. If the subsystem is not initially busy, queueing is bypassed.

The channel request queue and interrupt queue contain information to direct the attention of the device handlers to the unit or file with which the request or interrupt is associated.

When an I/O request is made by the user, the executive sets the status word (word 3 of the packet) negative to indicate an in progress state. Before setting the word negative, a check is made to see if it is already negative which indicates a possible loop. If a loop occurs, a status code of 27_8 is placed in the I/O packet and control is transferred to the user-specified ERR mode routine. When the request is completed, a positive value is placed in the status word; no housekeeping is necessary by the user and encountering an initial negative value in the packet can be interpreted as a software logic error.

Efficient utilization of all drum types including FASTRAND-formatted mass storage dictates that servicing requests for a given file are not restricted to the order of submission. The nonsequential processing of I/O requests to mass storage results in faster servicing and more efficient utilization of the system's I/O facilities. Testing each packet is necessary to ensure completion but do not assume completion by testing a subsequent packet.

6.2. I/O PACKET GENERATION

There are two basic procedures for generating I/O packets; I$OT (see 6.2.1) is used to generate I/O packets for magnetic tape files, while I$OD (see 6.2.2) is used for mass storage file I/O packets. An I/O operation with interrupt involves inclusion of additional parameters for word 2 of the I/O packet (see Figure 6–1). The tag on the procedure line is allocated to the first word of the I/O packet.

6.2.1. MAGNETIC TAPE I/O PACKET GENERATION (I$OT)

- Magnetic Tape I/O Function Without Interrupt

Format:

I$OT 'filename',function word-count,buffer-addr[,G]

Parameters:

filename — Specifies the tape file being referenced.

function — Symbolic or octal code identifying function to be performed (see 6.4 and Table 6–1).

woud-count — Number of words to be transferred.

buffer-addr — The main storage address at which the transfer begins.

G — Increment-decrement function designator as follows:

'D' — Decrement

'N' — Inhibit incrementation or decrementation

Omit this parameter for incrementation.

Example:

```
  LABEL      ∧   OPERATION     ∧              OPERAND       ∧              COMMENTS
1           10                20             30            40             50
          I$ØT      'T101', W$    200,BFR
```

Filename T101 is entered in words 0 and 1 of I/O packet (see Figure 6–1). Symbolic function code W$ (write) is placed in the S2 portion of word 3. A total of 200 words are to be written (placed in word-count portion of word 4) starting at main storage location BFR. The omission of the G parameter indicates that incrementation is employed.

- Magnetic Tape I/O Function With Interrupt

Format:

I$OT 'filename',function,[interrupt-activity-addr] [,int-act-id] word-count,buffer-addr[,G]

Parameters:

filename — Specifies the tape file being referenced.

function — Symbolic or octal code identifying the function to be performed (see 6.4 and Table 6–1).

interrupt-activity-addr — Address of activity to which control is passed after completion of I/O function.

int-act-id — Any integer (1_{10} to 35_{10}) that identifies the interrupt activity if synchronization with another activity is desired.

word-count — Number of words to be transferred.

buffer-addr — Main storage address at which the transfer begins.

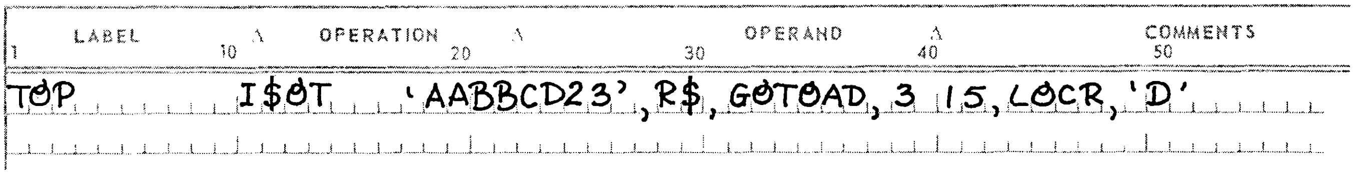
G Increment-decrement function designator as follows:

'D' – Decrement

'N' – Inhibit incrementation and decrementation

Omit this parameter for incrementation.

Example:


LABEL
OPERATION
OPERAND
COMMENTS
1
10
20
30
40
50

```
TOP      I$OT   'AABBCD23',R$,GOTOAD,3 15,LOCR,'D'
```

Filename AABBCD23 is entered in words 0 and 1 of I/O packet (see Figure 6–1). Symbolic function code R$ (read) is placed in S2 of word 3. Symbolic interrupt address GOTOAD is placed in H2 of word 2, while the interrupt-activity-id (3) is placed in S3 of word 2. The number of words (15) to be read into main storage starting at location LOCR is placed in the word-count portion of word 4; LOCR is placed in the buffer-addr portion of word 4, and the 'D' indicates a decrementing function to control the direction of the words transferred.

6.2.2. MASS STORAGE I/O PACKET GENERATION (I$OD)

Purpose:

Creates I/O packet for mass storage files.

Mass Storage I/O Function Without Interrupt

Format:

```
I$OD   'filename',function   word-count,buffer-addr,[G] drum-addr[,search-sentinel]
```

Parameters:

filename Specifies the file being referenced.

function Symbolic or octal code identifying the function to be performed (see 6.4 and Table 6–1).

word-count Number of words to be transferred.

buffer-addr The main storage address at which the transfer begins.

G Increment-decrement function designator as follows:

'D' – Decrementation

'N' – Inhibit incrementation and decrementation

Omit this parameter for incrementation.

drum-addr Identifies the logical mass storage address at which the operation starts.

search-sentinel The sentinel to be recognized when a search function is performed on the mass storage file.

Example:

```
 LABEL         OPERATION                    OPERAND                       COMMENTS
          10             20            30            40            50
LRR       I$ØD   'D222',BRD$,56,BUFS,D  04333222
```

Filename D222 is entered in words 1 and 2 of the I/O packet. A block read drum function (BRD$) is inserted in S2 of word 3. A total of 56 words are read into BUFS (placed in H2 of word 4) with decrementation chosen (2 placed in G of word 4). The word transfer starts at drum address 4333222_8 (placed in T2 and T3 of word 5).

- Mass Storage I/O Function With Interrupt

Format:

```
I$OD   filename,function[,interrupt-activity-addr] [,int-act-id] word-count,buffer-addr[,G] drum-addr[,search-sentinel]
```

Parameters:

Same as for I$OD (see 6.2.2) without interrupt, except that the interrupt addr and id number are included in the parameters.

6.3. PROGRAM – I/O SYNCHRONIZATION

An activity is synchronized with the completion of an I/O operation previously submitted by the same activity to the executive through an IO$ request by entering the executive through a WAIT$ request (see 6.3.1) or a WANY$ request (see 6.3.2). The WAIT$ request may also be used to wait for the completion of an I/O operation performed by other program activities. When interrupt activities are used, they perform the I/O, not the originating activity.

A WAIT$ request waits for completion of a particular I/O operation and must be preceded by a Test Positive (TP) instruction on word 3 of the I/O packet (see Figure 6–1). A test is made within the executive on a WANY$ request to determine if any I/O request has been completed for the requesting activity since the last time the activity was placed in an I/O wait condition by a previous WAIT$, WANY$, IOW$, or IOWI$ request.

6.3.1. WAIT FOR COMPLETION OF SPECIFIC I/O (WAIT$)

Purpose:

Delays execution of an activity until the I/O operation controlled by a specific I/O packet (see Figure 6–1) has been completed.

Format:

```
TP     pktaddr+3
ER     WAIT$
```

Description:

When an I/O executive request is submitted, the executive sets word 3 of the I/O packet (see Figure 6–1) negative; word 3 remains negative until the completion of the I/O operation. The Test Positive (TP) check is made on this word.

Because the executive performs a second test to determine the completion of the I/O request, the h and i designators of the Test Positive instruction must be set to zero.

The packet address is the specific request waited for at WAIT$.

6.3.2. WAIT FOR COMPLETION OF ANY I/O (WANY$)

Purpose:

Delays execution of an activity until an I/O operation, controlled by a specific I/O packet (see Figure 6–1) for that activity has completed. No delay occurs if the I/O operation has already occurred since the last WANY$, WAIT$, IOW$, or IOWI$ request.

Format:

```
ER   WANY$
```

Description:

At least one IO$ request (see 6.3.3) must have been submitted since the last WAIT$, WANY$, IOW$, IOWI$ request, or else it must be known that the I/O operation is outstanding for that activity, that is, the status must be found negative.

An error results if no I/O operations are still in process for that activity and none has been submitted since the last WAIT$, WANY$, IOW$, or IOWI$ request.

The following ER's cause a wait for the completion of all outstanding I/O operations for the program and affect the use of the WANY$ request in the same manner as a previous WANY$, WAIT$, IOW$, or IOWI$ request:

EXLNK$ (see 10.4.5.1)

LCORE$ (see 4.7.2)

LINK$ (see 10.4.4.1)

RLINK$ (see 10.4.4.2)

UNLNK$ (see 6.3.8)

6.3.3. INITIATE I/O AND RETURN CONTROL IMMEDIATELY (IO$)

Purpose:

To request an operation on the I/O file indicated and to return control to the executing program without waiting for completion of the I/O operation.

Format:

```
L,U     A0,pktaddr
ER      IO$
```

This linkage may be generated by the procedure call:

```
I$O   pktaddr
```

Description:

Pktaddr is the address of the I/O packet (see Figure 6–1), which controls all I/O device handler operations.

6.3.4. INITIATE I/O AND RETURN CONTROL IMMEDIATELY, WITH INTERRUPT (IOI$)

Purpose:

Same as for IO$ (see 6.3.3), except that an interrupt activity is initiated at completion of the I/O request.

Format:

```
L,U     A0,pktaddr
ER      IOI$
```

This linkage may be generated by the procedure call:

```
I$OI   pktaddr
```

Description:

Pktaddr is the address of the I/O packet (see Figure 6–1) which controls all I/O device handler operations.

6.3.5. INITIATE I/O AND WAIT FOR COMPLETION (IOW$)

Purpose:

Same as for IO$ (see 6.3.3), except that control is not returned to the executing program until completion of the I/O operation.

Format:

```
L,U     A0,pktaddr
ER      IOW$
```

This linkage may be generated by the procedure call:

```
I$OW   pktaddr
```

Description:

Pktaddr is the address of the I/O packet (see Figure 6–1) which controls all I/O device handler operations.

6.3.6. INITIATE I/O AND WAIT FOR COMPLETION, WITH INTERRUPT (IOWI$)

Purpose:

Same as for IOW$ (see 6.3.5), except that an interrupt activity initiated upon completion of the I/O operation.

Format:

```
L,U     A0,pktaddr
ER      IOWI$
```

This linkage may be generated by the procedure call:

```
I$OWI   pktaddr
```

Description:

Pktaddr is the address of the I/O packet (see Figure 6–1) which controls all I/O device handler operations.

6.3.7. INITIATE I/O AND EXIT, WITH INTERRUPT (IOXI$)

Purpose:

To request an operation on the I/O file indicated and terminate the requesting activity. Upon completion, initiate an interrupt activity. This increases the completion priority and saves the time required to store and restore the register set.

Format:

```
L,U     A0,pktaddr
ER      IOXI$
```

This linkage may be generated by the procedure call:

```
I$OXI  pktaddr
```

Description:

Pktaddr is the address of the I/O packet (see Figure 6–1) which controls all I/O device handler operations.

6.3.8. REDUCING INTERRUPT ACTIVITY PRIORITY (UNLCK$)

Purpose:

Allows an interrupt activity to reduce its priority.

Format:

```
ER      UNLCK$
```

Description:

The UNLCK$ request enables an I/O interrupt activity to reduce its switching priority to the priority of the activity which initiated the I/O request. Any other executive request executed by thė interrupt activity has the same result. However, in a time critical, multiactivity program, the UNLCK$ request provides a low overhead means of level reduction.

6.4. MAGNETIC TAPE HANDLER

6.4.1. TAPE HANDLER FUNCTIONS

The various magnetic tape functions (see Table 6–2) are controlled by a routine that is always located in main storage. The current position of each tape is kept in terms of a block count and is made available for error logging, checkpoint, and ending label routines. No provision is made for automatic treatment of mixed parity and mixed density tape files.

Utilization of the contingency write and skip write functions are automatically provided by the handler, and unless the user provides his own error recovery, these functions should not concern the user.

In order to use the handler, an I/O control packet must be generated (see 6.2).

Function	Symbol	Octal Code	Description
Write	W$	10	Starting at the address in H2 of word 4 of the I/O packet, transfer the number of words specified in H1 of word 4 to form a single block on magnetic tape. Transfer is accomplished according to the standard modes or the requested modes, that is, parity, density, and so forth. Normal completion results when all words have been transferred, except for UNISERVO VI-C/VIII-C tape units, seven-track format, even parity, where a character of zero, after translation is requested, will conclude the request for more data by the subsystem.

Table 6–2. Magnetic Tape I/O Functions and Codes (Part 1 of 2)

Function	Symbol	Octal Code	Description
Write end of file	WEF$	11	Write a sentinel on magnetic tape which, when read, results in an EOF status being returned to the program.
Contengency write	CW$	12	Write zeros, in even channels only, for 2.5 inches of tape to allow writing after reading forward (UNISERVO III-A tape units only). This function is automatically provided by the system and should be of no concern to the user unless the suppress recovery mode is employed.
Skip write	SW$	13	Erase three inches of tape, then the same as a write function. This function is automatically provided in the system for write parity recovery. Required for the suppress recovery mode or if an extended interblock gap is needed on compatible tape types.
Gather write	GW$	15	Write a single block on magnetic tape specified by a string of access words. The number of access words is specified in H1 of word 4 and the starting address of the string is specified in H2 of word 4.
Read forward	R$	20	Initiate tape motion in the forward direction and transfer the words read into the area defined by word 4 of the packet. Transfer is normally concluded by either encountering the end of block or transferring the number of words requested.
Read backward	RB$	21	Same as read forward except opposite direction.
Move forward	MF$	50	Move tape forward one block.
Move backward	MB$	51	Backspace the tape one block.
Forward Space File	FSF$	52	Move tape forward past the next EOF mark. This function is available only on the UNISERVO 12/16 tape units. It returns an EOF status if the end of the tape is not encountered.
Backspace File	BSF$	53	Move tape backward past the previous EOF mark. It returns an EOF status if the beginning of the tape is not encountered first.
Rewind	REW$	40	Reposition the tape at the load point. This is the point at which a read forward reads the first block on tape and a read backwards reports an end-of-tape status.
Rewind with interlock	REWI$	41	Reposition the tape to unload point and lock the unit against further functions
Set mode	SM$	42	Set operating mode function (see 6.4.1.1).
Scatter read forward	SCR$	43	Same as read forward except the words read are transferred into areas specified by a string of access words defined by word 4.
Scatter read backward	SCRB$	44	Same as scatter read forward except opposite motion direction.

Table 6–2. Magnetic Tape I/O Functions and Codes (Part 2 of 2)

Standard modes, in lieu of those set by the worker program, are established by the executive at initialization and reestablished when a tape is released as follows:

- high density
- odd parity
- no character translation
- 18-character noise constant
- standard recovery

In addition to the service entrance, the parity and density modes can be set by options on the @ASG control statement (see 3.7.1).

6.4.1.1. SET MODE FUNCTION

For the set mode function (SM$), the I/O access control word (word 4 of Figure 6–1) is set to a one-word buffer which defines the modes to be set as follows:

35 34	33 32	31 30	29 28	27 26	25 22	21 20	19 18	17 0
field 1	field 2	field 3	field 4	field 5	field 6	field 7	field 8	field 9

Field 1 — Density(s)

0_8 — No change

1_8 — Low (for UNISERVO 12/16 nine-track tape units - 800 FPI)

2_8 — Medium

3_8 — High (for UNISERVO 12/16 nine-track tape units - 1600 FPI)

Field 2 — Parity

0_8 — No change

1_8 — Odd (binary)

2_8 — Even (BCD)

Field 3 — Translate

0_8 — No change

1_8 — Set character translate mode (for UNISERVO 12/16 seven-track tape units-translate Fieldata to or from BCD compatible with UNISERVO VI-C/VIII-C operation)

2_8 — Discontinue translation

Field 4 — Allow noise

0_8 — No change

1_8 — Set the noise constant to the number of characters in Field 9

Field 5 – Suppress recovery

0_8 – No change

1_8 – Return external interrupt status code to the worker program in case of malfunction without attempting recovery

2_8 – Discontinue suppress recovery mode

Field 6 – MSA Translator

0_8 – No change

1_8 – Fieldata to EBCDIC

2_8 – Fieldata to ASCII-6

3_8 – XS–3 to EBCDIC

4_8 – XS–4 to ASCII-6

5_8 – EBCDIC to Fieldata

6_8 to 16_8 – Reserved for additional translate options

17_8 – Discontinue translation

Field 7 – Control unit data converter (UNISERVO 12/16 only)

0_8 – No change

1_8 – Set data converter mode

2_8 – Discontinue data conversion

Field 8 – MSA transfer mode (see Table 6–4)

0_8 – No change

1_8 – Quarter word (MSA A format)

2_8 – Six-bit packed (MSA B format)

3_8 – Eight-bit packed (MSA C format)

Field 9 – Noise constant character count

When suppress recovery mode is set, entry to I/O control is only possible by means of the IOI$, IOWI$, or IOXI$ requests.

When the suppress recovery mode is set the user is returned the following information at the time of the interrupt:

Packet status code	–	11_8
Register A1	–	Status word
Register A2 and A3	–	If an MSA error occurred, the MSA auxiliary status word is returned in register A2. If the unit check or unit exception bits are set in the status word, sense bytes 0–3 are returned in register A2 and sense byte 4 in register A3. The sense bytes are in Quarter-word format and sense byte 4 is in Q1 of register A3.

6.4.2. GENERAL CONSIDERATIONS

The following must be considered when using the magnetic tape handler for compatible magnetic tape units:

6.4.2.1. READ BACKWARD LIMITATIONS

The read backward function on the UNISERVO VI-C/VIII-C tape unit should not be used if the tape to be read has been recorded on some other type of unit. It is necessary that the recording produce a statically deskewed longitudinal check frame to prevent the read backward function from interpreting the check frame as data frames.

If a block is recorded in seven-track format with a block length greater than five frames and not a multiple of six, a read backward produces a different format than a read forward of the same block. For example, if the block length is seven frames, a read forward results in assembling frames 1 through 6 as the first word and frame 7 as the second and a read backward results in assembling frames 2 through 7 as the first word and frame 1 as the second.

The same type of buffer variation exists for a read backward function on a nine-track unit if the write buffer length is not a multiple of two words (nine frames). A one-word write on a nine-track unit results in five frames being recorded with the fifth frame containing four bits of zero padding. A read backward results in the four bits of padding appearing as the least significant four bits of the first word assembled. Furthermore, regardless of the direction of reading, if a block is written on a nine-track format unit with an odd word count in the access word, one more word is made available as input than was sent out to be written.

6.4.2.2. WRITE CONSIDERATIONS

If the user attempts to write EOF sentinels on seven-track UNISERVO IV-C, VI-C, and VIII-C tape units by doing an even parity write with truncation caused by a zero character and the translate mode is set, it is essential that the first two characters of the buffer translate to 1700_8 to cause an EOF status when read.

Since hardware translation may be available and the user has the ability to vary the translation, care must be exercised to prevent unwanted translation of a character to zero which causes truncating a write transfer when writing in the even parity mode. If software conversion is used for write operations, the words are converted in the buffer before the write operation is performed. If a block is written with less characters than the noise constant, the risk exists of bypassing the data block as noise when reading. Also, a zero as the first character results in an erroneous block count. On a zero character count, up to three words leave the computer and are considered to have been written as reflected in the count in the substatus field of the request packet (UNISERVO VI-C/VIII-C tape units).

When using the UNISERVO 12/16 tape units, there are a number of incompatibilities with the UNISERVO IV-C/VI-C/VIII-C tape operations of which the programmer should be aware. They are:

- Fieldata-to-BCD translations do not convert all the codes the same (see Table 6–3)

- The Fieldata 00_8 code in even parity does not stop the write operation as on the C-type units. Instead it is converted to a 14 BCD

- Variable length block by character can be accomplished by using the MSA A format and setting the ninth bit of the character field. This records that character and then stops the write operation.

- When using the MSA C format read and a block ends on the fifth or sixth frame, the even word is not transferred to the computer (see Table 6–4).

The recovery procedure for a parity error or certain tape hash errors on a write operation may utilize two feet of tape or twice the length of the block, whichever is larger. hence, if blocks are to be recorded which are longer than two feet (or less, depending upon whether an ending interrupt activity submits the next request or if requests are queued ahead by I/O control), it is recommended that tapes be used which have the end-of-tape warning marker placed farther from the end of tape. The normal placement is 14 feet from the end of tape, and it is recommended that at least 10 feet of tape remain on the supply side of the write head to ensure that the tape is not pulled off the supply reel.

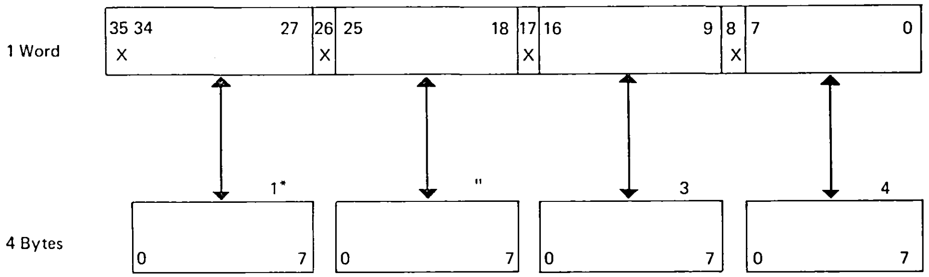
MSA TRANSLATIONS			UNISERVO 12/16 TRANSLATIONS				
Fieldata		EBCDIC			BCD		
Octal Code	HSP Symbol	Octal Code	Hexadecimal	Symbol	Octal Code	Hexadecimal	Symbol
00	@	174	7C	@	14	0C	@′
01	[	112	4A	¢			
02	]	132	5A	!			
03	#	173	7B	#	13	0B	# =
04	Δ	137	5F	¬	57	2F	Δ
05	(space)	100	40	(space)	00	00	(blank)
06	A	301	01	A	61	31	A
07	B	302	02	B	62	32	B
10	C	303	C3	C	63	33	C
11	D	304	04	D	64	34	D
12	E	305	C5	E	65	35	E
13	F	306	C6	F	66	36	F
14	G	307	C7	G	67	37	G
15	H	310	C8	H	70	38	H
16	I	311	C9	I	71	39	I
17	J	321	D1	J	41	21	J
20	K	322	D2	K	42	22	K
21	L	232	D3	L	43	23	L
22	M	324	D4	M	44	24	M
23	N	325	D5	N	45	25	N
24	O	326	D6	O	46	26	O
25	P	327	D7	P	47	27	P
26	Q	330	D8	Q	50	28	Q
27	R	331	D9	R	51	29	R
30	S	342	E2	S	22	12	S
31	T	343	E3	T	23	13	T
32	U	344	E4	U	24	14	U
33	V	345	E5	V	25	15	V
34	W	346	E6	W	26	16	W
35	X	347	E7	X	27	17	X
36	Y	350	E8	Y	30	18	Y
37	Z	351	E9	Z	31	19	Z
40	)	235	5D	)	55	2D	]
41	–	140	60	–	40	20	–
42	+	116	4E	+	76	3E	<
43	<	114	4C	<	74	3C	¤)
44	=	176	7E	=	16	0E	>
45	>	156	6E	>	36	1E	\
46	&	120	50	&	60	30	& +
47	$	133	5B	$	53	2B	$
50	*	134	5C	*	54	2C	*
51	(	115	4D	(	75	3D	[
52	%	154	6C	%	34	1C	% (
53	;	172	7A	:	20	10	Sub b̄ blank

Table 6–3. Type 5017 Fieldata/BCD Translations (Part 1 of 2)

MSA TRANSLATIONS			UNISERVO 12/16 TRANSLATIONS				
Fieldata		EBCDIC			BCD		
Octal Code	HSP Symbol	Octal Code	Hexadecimal	Symbol	Octal Code	Hexadecimal	Symbol
54	?	157	6F	?	37	1F	⧻
55	|	117	4F	|	77	3F	‡
56	, (com)	153	6B	,	33	1B	,
57	\	340	E0	\			
60	0	360	F0	0	12	0A	0
61	1	361	F1	1	01	01	1
62	2	362	F2	2	02	02	2
63	3	363	F3	3	03	03	3
64	4	364	F4	4	04	04	4
65	5	365	F5	5	05	05	5
66	6	366	F6	6	06	06	6
67	7	367	F7	7	07	07	7
70	8	370	F8	8	10	08	8
71	9	371	F9	9	11	09	9
72	' (APO)	175	7D	'	15	0D	:
73	;	136	5E	;	56	2E	;
74	/	141	61	/	21	11	/
75	. (PER)	113	4B	.	73	3B	.
76	¤	177	7F	"(quot)	17	0F	√
77	/ = (or stop)	155	6D	(und)	35	1D	γ

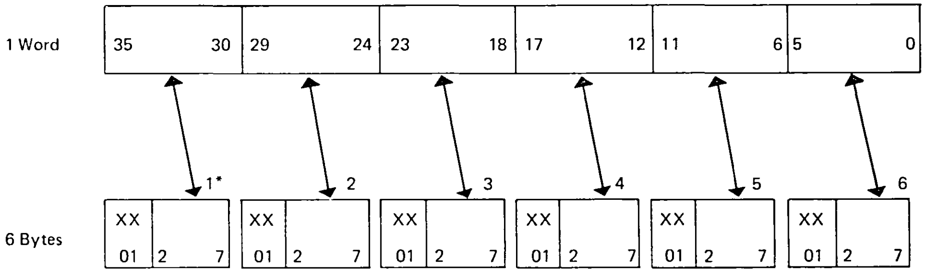
Table 6–3. Type 5017 Fieldata/BCD Translations (Part 2 of 2)

FORMAT A (QUARTER WORD)



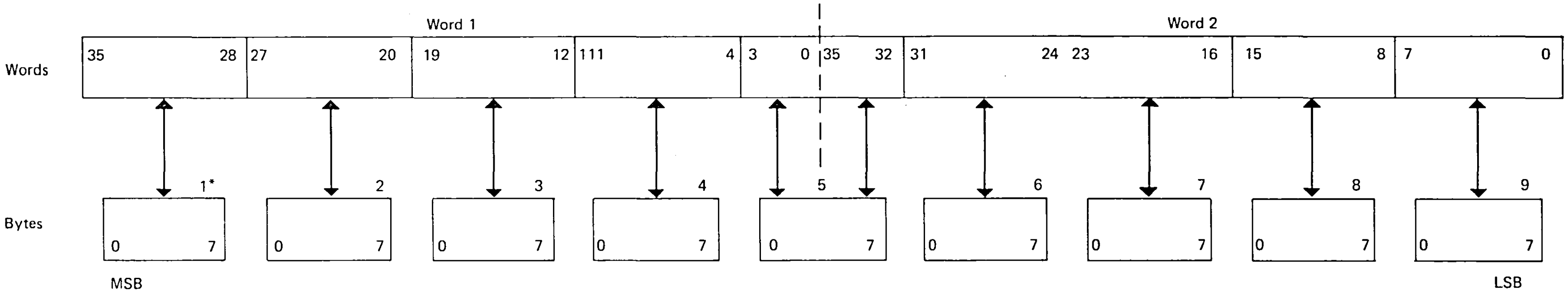
NOTE: Bits 35, 26, 17, and 8 are used for stop control on output operations and forced to binary 0 on input operations.

FORMAT B (6-BIT PACKED)



NOTE: Bit 0 and 1 become binary 0 on output and are ignored on input, for each 8-bit byte. When translation is specified, bits 0 and 1 are not forced to binary 0.

FORMAT C (8-BIT PACKED)



*Numbers on arrows indicate the order of byte transfer.

Table 6–4. MSA Data Word Formats

6.4.2.3. MOVE CONSIDERATIONS

The move forward and move backward functions are concerned with position. Parity errors are not reported and are only examined to determine noise blocks. For the UNISERVO IV-C tape units, the parity status is not returned for the backspace block function; therefore, the move backward is not recommended on the UNISERVO IV-C tape units if noise is a problem, as a lost position may result.

A cross-reference of magnetic tape functions, unit types, and packet lengths are provided in Table 6-5; Table 6-6 lists the standard tape translation.

6.4.2.4. ABNORMAL FRAME COUNT CONSIDERATIONS

The AFC field in the I/O packet (see Figure 6-1) is supplied by the executive when a status code of 4_8 is returned. The value of this field is calculated based on the following algorithm. If the character count is not a multiple of 6 (for seven channels per frame) or a multiple of 9 (for nine channels per frame), AFC contains the number of characters in the last word read (first word of the buffer for a read backward). This field is used in conjunction with a status code of 04. If the access word does not have a word count large enough to allow transfer of the entire block and a status code of 04 is returned, this field is set to zero. (For UNISERVO IV-C magnetic tape units, if the access word goes to zero, it is indeterminate whether all words were read. If all words of the block are read, as determined by the user, then the count of frames of data in the last word read is stored in the lower sixth of the last data word.) For nine-channel tapes, the count is the number of eight-bit bytes assembled and transferred to the CPU in the last two-word sequence; that is, a value of 1 indicates an odd number of words with one eight-bit byte assembled in the final word and the remainder of the word padded with zeros. A value of 5 indicates an even number of words with four data bits in the last word which are the least significant half of the eight-bit byte with the most significant four bits in the preceding word.

6.4.3. MULTIPLE-CHANNEL OPERATION

The magnetic tape handler is capable of a simultaneous operation on any number of channels involving any mixture of tape device types.

Full dual-channel operation on UNISERVO VI-C/VIII-C and UNISERVO 12/16 tape units are supported without user cognizance.

Function	Octal Code	Pkt Length	VI-C/VIII-C	IV-C	III-A	II-A	12/16	12N/16N
Read Forward	20	5	*	*	*	*	*	*
Read Backward	25	5	*	I	*	*	*	*
Scatter Read	43	5	*	*	I	I	*	*
Scatter Read Backward	44	5	*	I	I	I	*	*
Move Forward	50	4	*	*	*	*	*	*
Move Backward	51	4	*	*	*	*	*	*
Write	10	5	*	*	*	*	*	*
Write end-of-file	11	4	*	*	*	I	*	*
Contingency Write	12	4	I	I	*	I	I	I
Skip Write	13	5	*	*	I	I	*	*
Gather Write	15	5	*	*	I	I	*	*
Rewind	40	4	*	*	*	*	*	*
Rewind with Interlock	41	4	*	*	*	*	*	*
Forward Space File			I	I	I	I	*	*
Backspace File			I	I	I	I	*	*
Set Mode:	42	5	*	*	*	*	*	*
Very high density(1600 FPI)			I	I	I	I	I	*
High density			*	*	*	*	*	*
Medium density			*	*	I	*	*	I
Low density			*	*	*	*	*	I
Odd parity			*	*	I	*	*	*
Even parity			*	*	I	*	*	I
Translate			*	*	*	*	*	*
Allow noise			*	*	*	*	*	*
Suppress recovery			*	*	*	*	*	*

CODE:

* – Available

I – Invalid function, causes termination

Table 6–5. Magnetic Tape Function vs Unit Type

TAPE CODE	CPU CODE	TAPE CODE	CPU CODE
00	46	40	41
01	61	41	17
02	62	42	20
03	63	43	21
04	64	44	22
05	65	45	23
06	66	46	24
07	67	47	25
10	70	50	26
11	71	51	27
12	60	52	55
13	44	53	47
14	72	54	50
15	53	55	02
16	45	56	73
17	00	57	04
20	05	60	42
21	74	61	06
22	30	62	07
23	31	63	10
24	32	64	11
25	33	65	12
26	34	66	13
27	35	67	14
30	36	70	15
31	37	71	16
32	77	72	54
33	56	73	75
34	51	74	40
35	52	75	01
36	57	76	43
37	76	77	03

Table 6–6. Standard Tape/Processor Code Translation (Octal)

6.5. MAGNETIC DRUM AND UNITIZED CHANNEL STORAGE HANDLER

While the following discussions are oriented toward magnetic drum operation, all operations and considerations described are equally applicable to unitized channel storage.

6.5.1. HANDLER FUNCTIONS

Two general modes of drum operation are provided within the handler. The first is drum simulation of FASTRAND mass storage which allows execution of a program with files designed for FASTRAND mass storage to be allocated to flying head (FH) drum storage but handled as simulated FASTRAND mass storage. The second mode is drum as a random storage device and is word addressable. The interpretation of function codes for simulated FASTRAND mass storage is discussed in 6.7. For drum format, the functions are listed in Table 6–7.

In order to use the handler, an I/O control packet must be generated (see 6.2).

Function	Symbol	Octal Code	Description
Write	W$	10	Starting at the main storage address specified in H2 of word 4, transfer the number of words specified in H1 of word 4 to the drum area starting at the relative address in word 5 of the I/O packet. The write operation also removes any locks on the area written in the same manner as the unlock operation.
Gather write	GW$	15	Transfer the number of words specified by a string of access words specified by word 4 from the areas specified by these access words to the drum area starting at the relative address in word 5. The number of access words is specified in H1 of word 4 and the address of the access words is specified in H2 of word 4. The write operation also removes any locks on the area written in the same manner as unlock operation.
Read	R$	20	Starting at the relative drum address in word 5 of the request packet, transfer the number of words in H1 of word 4 into the area starting at the address in H2 of word 4. Normal completion (status 0_8) indicates the specified number of words have been transferred to main storage from drum.
Scatter read	SCR$	43	Starting at the relative drum address in word 5 of the I/O packet, transfer the number of words specified by a string of access words defined by word 4 to the areas specified by these access words. The number of access words is specified in H1 of word 4 and the address of the access words is specified in H2 of word 4.
Block read	BRD$	24	Starting at the relative drum address in word 5 of the I/O packet, transfer words from drum to main storage at the address in H2 of word 4 until either the number of words specified in H1 of word 4

Table 6–7. Magnetic Drum and Unitized Channel Storage I/O Functions and Codes (Part 1 of 2)

Function	Symbol	Octal Code	Description
			has been read or until the end-of-block sentinel (a word of all 1's) is read. Encountering a sentinel is noted by 1_8 status code and the sentinel word is transferred as the final word in the buffer. The substatus field indicates the number of words read. If completion is due to end of block and the buffer length is such that another word can be accepted, the overflow word (the word on drum following the sentinel) is stored in the buffer following (preceding if decrementation, or on if inhibit incrementation/decrementation) the sentinel word with the upper six bits set to 4_8.
Read and lock	RDL$	25	Perform the read operation and impose a logical lock to be placed on the area read, which prevents access to the part of the file defined by the access word and relative starting address by other activities (of either the same run or other runs) until such time as the locking activity unlocks the area. Removal of this exclusive use of a block is accomplished by writing into any part of the block, issuing an unlock request as defined in unlock, or by terminating the activity (see 6.6.1).
Unlock	UNL$	26	Remove any logical locks imposed on other activities by read and lock requests submitted by this activity for the area of the file specified by the address and length of the access for this request. Locks are maintained by block, and unlocking any part of a block unlocks the entire block. Also, one unlock request can unlock several blocks (see 6.6.1).
Block search read	BSRD$	37	Starting at the relative drum address in word 5 of the I/O packet, compare equal between the drum words and word 6 of the packet. No fund is recognized by encountering an end-of-block sentinel or the end of the granule (track or position). Upon a find, store the relative address of the find word in word 7 of the packet and transfer words as with a block read, with truncating for end-of-block sentinel or end of granule. Storing the overflow word follows the same criteria as with the block read function.
Search	SD$	34	Starting at the relative drum address in word 5 of the I/O packet, compare all words on drum until either a compare equal is made with word 6 of the packet or until the remainder of granule (track or position) has been tested. If a find is made (status 0_8), the relative address of the find is stored in word 7 of the packet.
Search read	SRD$	36	Starting at the relative drum address in word 5 of the I/O packet, compare all words on drum until either a compare equal is made with word 6 of the packet or until all remaining words of the granule (track or position) are tested. If a find is made, store the relative address of the find in word 7 of the packet and transfer the number of words specified in H1 of word 7 into the main storage area starting at the address in H2 of word 4. Truncate the read cycle if the end of assignment precedes the count in H1 of word 4.
Block search	BSD$	35	Same as a search, with the added condition that reading an end-of-block sentinel word terminates the search with a no find status code (3_8).

Table 6—7. Magnetic Drum and Unitized Channel Storage I/O Functions and Codes (Part 2 of 2)

6.5.2. GENERAL CONSIDERATIONS

The functions listed in Table 6–7 are performed on areas reserved through the use of the @ASG control statement (see 3.7.1). These assignments are fixed in length, hence, an attempt to read, write, or initiate a search past the end of the assigned area results in an error condition. A write request must be totally within the assigned area.

Search functions are terminated by the software after a time interval has elapsed; passing over the area of concern without receiving an interrupt indicates a find. The area of concern would be the end of the track or position equivalent. Thus, the length of time the subsystem is tied up for a search is nearly the same as a read or write of the same length area. The handler ensures that a search find is within the assigned area before reading thus guaranteeing file privacy. If a read after search must be truncated, a status code of 1_8 is returned to the program. A search function issued from a user's program searches a maximum of one granule of the file. If a find is made, it must be within the same granule in which the search was started. A search read drum or block search read drum is terminated if the access control word is zero. When the read phase of the SRD$ or BSRD$ functions involves going over granule boundaries, the read is completed for the user. If any part of a read after search find is outside of the assigned area, the request is truncated.

The I/O status code 5_8 for drum format results from attempting I/O in an area of a file not currently allocated. Status code 22_8 does not occur unless referencing beyond the file-maximum granule; this includes references to words beyond the last word specified at assignment by lying outside the last granule.

6.5.3. MULTIPLE-CHANNEL OPERATION

The full dual-channel operation is supported for the simultaneous FH–432/1782 subsystems without user inconvenience. The magnetic drum handler is capable of simultaneous operation on any number of channels involving a combination of magnetic drum types.

6.6. FASTRAND MASS STORAGE HANDLER

6.6.1. FASTRAND HANDLER FUNCTIONS

The various FASTRAND mass storage I/O functions are listed in Table 6–8. Although the system functions without a physical FASTRAND mass storage unit, at least some portion of magnetic drum must be set aside to simulate the FASTRAND mode in its absence. The minimum FASTRAND format area has space for symbiont input and output files, system processor data area, program file storage, and other system functions.

Space on FASTRAND mass storage is assigned in granules of one track (64 sectors) or one position (64 tracks). A file consisting of more than one granule may be considered contiguous by the programmer because the handler takes care of the processing that must occur whenever a granule boundary is passed. The handler works in conjunction with the file supervisor to convert the relative sector addresses supplied by the user program into physical channel, unit, position, and sector addresses.

An attempt to read from an area of a file which is not entirely assigned results in a status code of 5_8 being returned to the I/O packet. If the area starts within the assignment and runs beyond it, the substatus count reflects the part assigned. If granules have been released causing voids within the file, a request could generate a legal start and ending address but a void within the file and this would result in the 5_8 status being returned with only the first part of the file read. Writing into an unassigned area of a FASTRAND-formatted file causes space to be assigned to the portion of the file. The automatic expansion on a write function can be overridden by the maximum granule parameter on the @ASG control statement (see 3.7.1). In this case, a status code of 22_8 is returned in the I/O packet.

FASTRAND-formatted files can be exclusively assigned to a run. Thus an activity can read an area of the file and update it without having another activity of any program, to which the file is assigned, access the volatile data. The exclusive-use feature applies only to the area being accessed. All other areas of the file can be accessed during this period of exclusive use. The only activity permitted to access the locked out area is the activity that initially set the exclusive-use option. All other references either partially or totally within the locked out data area are suspended until the exclusive-use lock is removed.

Function	Symbol	Octal Code	Description
Write	W$	10	Starting at the relative sector address specified in word 5 of the I/O packet, transfer the number of words specified in H1 of word 4 from the main storage area starting at the address in H2 of word 4 to FASTRAND mass storage. If the count is not a multiple of 28, write zeros into the remainder of the last sector. (Zero padding is not simulated on drum; hence, the partial sector is not changed.) If the area being written into is not currently assigned, expansion of the file is automatic up to the maximum from the @ASG control statement (see 3.7.1). The write operation also removes any locks on the area written in the same manner as the unlock operation.
Gather write	GW$	15	Same as write except word 4 specifies a string of access words each specifying a word count and a main storage area.
Acquire FASTRAND	ACQ$	16	Starting at the relative sector address specified in word 5 of the I/O packet, the file is expanded by the number of granules required to hold the number of words specified in H1 of word 4. This allows expansion of a file without writing into it. Expansion of the file is automatic up to the maximum from the @ASG control statement (see 3.7.1).
Read	R$	20	Starting at the relative sector address specified in word 5 of the I/O packet, transfer the number of words specified in H1 of word 4 into the main storage area starting at the address in H2 of word 4. Reading always starts at a sector boundary but may end anywhere.
Scatter read	SCR$	43	Starting at the relative sector address specified in word 5 of the I/O packet, transfer the number of words specified by a string of access words specified by word 4 into the main storage areas specified by the access words. The number of access words is specified in H1 of word 4, and the address of the access words is specified in H2 of word 4.
Read and release	RR$	22	Same as read with the additional condition that after the read has been performed, all granules with any part within the set of addresses described by the packet are released to the available mass storage pool.
Release	REL$	23	Same as read and release, except no reading is performed.
Read and lock	RDL$	25	Perform the read operation and impose a logical lock to be placed on the area read, which prevents access to the part of the file defined by the access word and relative starting address by other activities (of either the same run or other runs) until such time as the locking activity unlocks the area. Removal of this exclusive use of a block is accomplished by writing into any part of the block, issuing an unlock request as defined in unlock, or by terminating the activity.
Unlock	UNL$	26	Remove any logical locks imposed on other activities by read and lock requests sumbitted by this activity for the area of the file specified by the address and length of the packet for this request. Locks are maintained by block, and unlocking any part of a block unlocks the entire block. Also, one unlock request can unlock several blocks.

Table 6–8. FASTRAND Mass Storage I/O Functions and Codes (Part 1 of 2)

Function	Symbol	Octal Code	Description
Track search all words	TSA$	30	Starting at the relative sector address in word 5 of the I/O packet, compare each word on FASTRAND mass storage with the identifier in word 6 of the packet until either a compare equal is made or the end of the track is encountered (sector address is the next multiple of 100_8). If a compare equal is found, store the relative sector address of the sector in which the find is made in word 7 of the packet and read as many words as are specified in H1 of word 4 (or to the end of the assignment, whichever is smaller) starting withthe beginning of the sector in which the find was made. If no compare equal is made before end of track, return a no find status code (3_8).
Track search first word	TSF$	31	Same as track search all words, except a comparison is made only on the first word of each sector.
Position search all words	PSA$	32	Same as track search all words, except comparisons are made until a sector address which is a multiple of 10000_8 is reached.
Position search first word	PSF$	33	Same as position search all words, except comparisons are made only on the first word of each sector.

Table 6–8. FASTRAND Mass Storage I/O Functions and Codes (Part 2 of 2)

Since the exclusive use of files by block (as defined by the address and access word) involves an interaction between activities, the user should ensure that proper order is maintained in submitting requests to prevent two activities from locking against each other. To aid in detecting this interlock condition, I/O control checks the length of time that an activity leaves a lock on an item. If an item is locked by any one activity for over 12 minutes, at the time of the unlock sequence (either a write or unlock function) a status code (10_8) is returned to the I/O packet, indicating that exclusive use had timed out and has been removed. Removing exclusive use by this means allows the locked activities to progress in the normal manner and the locking activity no longer interferes. If the unlock operation is the result of a write request, the write function is not performed if the 10_8 status code (see 6.10) is returned. The 10_8 status code is also observed for areas which must be unlocked if a packet format error is detected on a subsequent request when taken off of the channel list. This results from a change in the packet by the worker program while the request is listed and after any lock has been imposed for the request in error.

During normal operation, the handler prepositions the various units to keep access time to a minimum. For this reason, the position function is not needed in the user's repertoire.

The position searches are legal only if the granularity is position. The track searches are available for both granularities.

6.7. DISC HANDLER

The disc handler provides control of the 8414 disc for both FASTRAND-formatted and word-addressable files.

6.7.1. DISC HANDLER FUNCTIONS

The handler provides support for all functions that are supported by the drum (see Table 6–7) and the FASTRAND (see Table 6–8) handlers. That is, the disc handler fully supports both FASTRAND-formatted and word-addressable files. The disc hardware provides for direct support of standard read and write operations but does not provide for block type operations (for example, BSRD$) and search operations. Thus, the following functions are simulated by the disc handler software:

(1) Block Read (BRD$)

(2) Block Search (BSD$)

(3) Block Search Read (BSRD$)

(4) Search (SD$)

(5) Track Search First Word (TSF$)

(6) Track Search All Words (TSA$)

(7) Position Search First Word (PSF$)

(8) Position Search All Words (PSA$)

Space is allocated in granules of one track (64 sectors) or one position (64 tracks) as denoted on the @ASG statement (see 3.7.1). The file space may be considered contiguous by the programmer in that the handler will handle any discontiguities that exist.

The handler also provides for the mounting and dismounting of disc packs (that is, other than those specified as part of the fixed mass pools). This feature benefits the installation in that it can have an infinite amount of mass storage space. It benefits the user program in that files can be assigned to specific disc packs. The executive provides protection in this area so as to secure packs from unauthorized access.

Because of the ability to specify file placement, the programmer should be aware of the addressing mechanism provided on the packs. Disc packs are prepped, normally, so as to allow four sectors/record and 12 records/track (disc). The system provides the ability to, if desired by the individual user, prep the program's packs in either of two other methods (that is, one sector/record or two sectors/record). The alternate prep factors are provided if the programmer finds that the standard prep factor is not optimal for his application. Note, however, that these alternate prep factors cause a considerable loss in usable data space, and this should be taken into account when considering an alternate prep factor.

Once a pack is prepped, there should never be a need to reprep the pack. However, the executive does provide an override so as to enable reprepping, if necessary. The prepping of a pack is specified only at the system console and not internal to the program. Thus, the program need not concern itself with prepping in that it has already been done.

The user program need not concern itself with accessing part of a multisector record because the handler controls this.

In order to use the handler, an I/O control packet must be generated (see 6.2).

6.7.2. PREPPING THE DISC

Disc operation has been defined as a drum or FASTRAND mass storage simulation mode of operation; the data is referenced by word or sector address (28 computer words). It is assigned by FASTRAND granule (1792 or 114,688 computer words) and must be prepped to achieve random access capability.

In prepping the following points should be considered:

(1) Prepping can take 5 to 10 minutes per pack per channel and is accumulative for every pack on a subsystem (up to 64 drives). Dual access preps two packs concurrently.

(2) One, two, or four sector records (28, 56, or 112 computer words) can be selected for a pack. Four sector records (112 computer words) are supported as an optimum length when considering capacity, which decreases with smaller records.

(3) Bad tracks are mapped as unassignable at prep time.

(4) Bit map granules currently are FASTRAND mass storage oriented and represent a track of 1792 words. A downed disc track is 1344 words when using the four-sector prep and in many cases is represented by two granule bits or 3584 words.

(5) The suggested 112-word record may not be optimum and application studies will determine this. Simple system generation parameterization and operator key-in will allow the variance of the record length on a pack.

(6) Once prepped, a pack is not usually reprepped. Override parameters, however, are provided to change the prep factors and remap bad tracks.

6.8. ABSOLUTE READ/WRITE CAPABILITY

The executive provides the ability for a user program to assign and absolutely access a mass storage device under one common file name (see @ASG control statement – 3.7.1).

The devices that may be accessed in this manner are the FH–432, FH–880, and FH–1782 drums.

The devices are referenced by means of the software function codes absolute read (ABSR$ – 47_8) and absolute write (ABSW$ – 17_8). The I/O packet has the same format as for other mass storage files except for the file address specified in word 5 of the packet. In this case, the format of this field is:

35 24	23 0
subsystem-nbr	device-addr

The device-addr field contains the physical unit number and unit address appropriate to the device being accessed. I/O control ensures that the unit being referenced is truly assigned to the associated file. If the unit or file is not assigned to the requesting program, the activity is taken to the error mode routine (see 4.9).

At time of completion, word 6 of the I/O packet will contain the first external interrupt (EI) status returned by the subsystem. The normal handler recovery mechanisms are exercised in case of abnormal operation. The number of handler recovery attempts is returned in S3 of word 3 of the I/O packet. If the recovery attempts fail, a status code of 11_8 is returned in the status field of the packet and an unanswerable message is displayed on the operator's console. In the event of timeout condition, a status of 7_8 is returned in the I/O packet.

6.9. ARBITRARY DEVICE HANDLER

The arbitrary device handler (ADH) allows the user to directly control the I/O functions to a device on an I/O channel. This capability provides support for special devices where standard handlers are not provided and for special operations on devices where standard handlers are provided.

The ADH is entered through either the IOARB$ or IOAXI$ requests described in 6.9.2 and 6.9.3, respectively.

6.9.1. ADH I/O PACKET

The format for the arbitrary device I/O packet as illustrated in Figure 6–2.

	S1	S2	S3	H2
Word 0	filename			
1				
2	0		int-act-id	interrupt-activity-addr
3				monitor-interrupt-activity-addr
4	*status*	time-out	*time-ind*	function-string
5	initial-access-word-1			
6	final-word-count-1			rel-time-1
7				
8				
9	initial-access-word-n			
10	final-word-count-n			rel-time-n

Figure 6–2. Arbitrary Device Handler Packet

Word 0 and 1

The internal filename used in all references to the file. This name is either the same name as the external filename specified in an @ASG control statement or is equated to an external filename by a @USE control statement.

Word 2

int-act-id — Numeric identity (1–35) given to the external interrupt activity (IOARB$ only). This field can be left as zero as for the magnetic tape or drum handler packet if no synchronization is required with this activity. For entrance at IOAXI$, the interrupt activity has the same id as the original activity.

interrupt-activity-addr — Address at which control is to be given upon occurrence of an external interrupt.

Word 3

Monitor-interrupt-starting-addr — The address at which the interrupt activity is given control if the function string indicates a monitor interrupt is to be returned to the user, and the interrupt which indicates completion of the operation is a monitor interrupt.

Word 4

status — Status code indicating the disposition of the request.

time-out — The number of six-second intervals the subsystem should be timed before the lack of a monitor or external interrupt is to be considered an error. The value 1 corresponds to 6 seconds, 2 to 12 seconds, and so forth.

time-ind — Indicates the disposition of a timeout condition. If this field is not zero and an operation is left outstanding on a channel for a time in excess of the time-out value, a unique status code is returned to the packet. If the field is zero, a timeout message is displayed on the operator's console, and the response is returned.

function-string — Consists of a group of three-bit bytes (octal digit string) interpreted from left to right (bits 17–15 comprise the first byte). The assigned codes are:

0 – End of string

1 – Initiate function mode without monitor (LFC)

2 – Initiate function mode with monitor (LFCM)

3 – Initiate output mode without monitor (LOC)

4 – Initiate output mode with monitor (LOCM)

5 – Initiate input mode without monitor (LIC)

6 – Initiate input mode with monitor (LICM)

Words 5, 7,...,n-1

The initial access words to be used to control the channel.

Words 6, 8,...,n

final-word-count-n — Final word count as contained in access control register.

rel-time-n — Relative time between execution of the corresponding operation in the string and the execution of the next operation or the occurrence of an interrupt. The time is given in 200-microsecond increments.

Starting at the left of the function-string, the operations represented by the code are carried out as directed. As the string is interpreted, succeeding pairs of access words are referenced. The final word count of the preceding operation is updated and the initial access word for the current operation is loaded. At most, six modes can be specified in the initiation string. As a practical limit, the combined length of all external function buffers is set at 9; exceeding this count is considered a program logic error and causes reference to the error mode return point. As an example of string interpretation. If an input operation is to be performed with termination by an external interrupt, the initiation string could be 510000_8 with two sets of access words. The first operation by the ADH is to load the input channel assigned to the filename specified in the packet using the access word in word 5. This is followed by a Load Function In Channel instruction (LFC) using the access word in word 7 to locate the function word. Upon occurrence of an external interrupt, the final access word count and the relative time are stored in word 6, and the final values for LFC are placed in word 8.

The user can specify instructions in any desired order to perform a particular I/O operation. When a monitor instruction is encountered, the ADH halts further interpretation of the string until the particular monitor occurs. The user program must make certain that the proper instructions are monitored to ensure that the respective access words do not get overlaid; that is, if two successive operations initiate output transfer, the first one should be with monitor unless the time between I/O instruction executions allows for transfer of all words of the first output buffer. To determine whether or not an access word has sufficient time to count down between initiation of operations by the ADH and hence possibly allow operating at times without a monitor, the minimum time between execution of the I/O instructions by the ADH is at least 10 microseconds (this varies upward, depending upon operation, overlapping data transfers, and so forth). For such sequences as a function transfer of a single-word external function (EF) buffer followed by an output transfer, this is sufficient time for the function transfer to be completed before output transfer is initiated without the necessity of monitoring the function transfer.

The appearance of monitored modes does not necessarily indicate the need for a monitor completion activity (specified in word 3), as the ADH interprets intermediate monitor modes. A monitor activity is required if either:

- the last mode in a string is with monitor; or
- the last mode is not monitored, and no external interrrupt is expected to signal conclusion of the mode established as a result of the final mode.

If any monitored modes precede the final mode, whether a wait for external interrupt should be done after the final I/O instruction is executed is determined by a nonzero value in H2 of word 3. For example, an input drum operation is normally terminated without interrupt; hence, the sequence LFC, LICM, LFC is used, and a monitor interrupt activity is specified and executed without waiting after sending out the second function following the input monitor interrupt whereas an output drum operation is normally terminated with interrupt; hence, the sequence LFC, LOCM, LFC may be used without a monitor interrupt activity, in which case a wait for external interrupt is done after sending out the second function.

Regardless of the manner in which the ADH gives control to the interrupt activity, in all cases, the input and output active states are cleared on the particular channel by execution of the Disconnect Input In Channel (DIC) and Disconnect Output In Channel (DOC) instructions before control is given to the interrupt activity.

When a function mode is called, the ADH inserts the proper unit designator and adds the proper base address to the relative address of the function word. At that time, if the channel contains equipment shared by other assignments, it may be necessary to perform certain error checks to prevent leaving the channel in an indeterminate state and to prevent intrusion upon other assignment privacy. Nonstandard special I/O devices are assigned by channel, and the ADH makes no modifications to the function words for these devices.

The function buffer for magnetic tape or mass storage channels is limited to a word count of one word, except for search functions, in which case a second word, the identifier, and for UNISERVO III-A tape units, a third word, the mask, are allowed. For other than these cases, in a multiple-word EF buffer, each word is modified by the unit designation and subjected to the particular tests based on equipment type.

Word 2 and word 3 of the packet may be used to specify interrupt activities, one of which is executed when the corresponding interrupt occurs. Word 2 specifies the activity to be executed in case of an EI. The lower half of the word gives the activity starting address, and S3 is set to the activity identity if synchronization is necessary. The register save and priority are assumed to be X11 through A5, R1 through R3, and top priority, respectively. An EI activity must always be specified regardless of whether a monitor interrupt is to be used. The monitor activity is defined in words in the same format as the EI activity. If both a monitor and an EI occur, the EI activity is given control, and occurrence of the monitor interrupt can be determined by examining the access word. When control is given to the interrupt activity, register A0 is loaded with the packet address, and, for the EI activity, register A1 contains the EI status word.

Upon completion of an I/O operation by the ADH, a status code is stored in S1 of word 4 of the request packet denoting the conditions of the completion.

6.9.2. INITIATE ADH AND RETURN CONTROL IMMEDIATELY (IOARB$)

Purpose:

Initiates an arbitrary device I/O operation with control returned, in line, as soon as the request is either listed or the operations have been initiated. An interrupt activity is initiated when the request is completed.

Format:

```
    L       A0,pktaddr
    ER      IOARB$
```

Parameters:

pktaddr	Address of device I/O packet (see Figure 6–2).

6.9.3. INITIATE ADH, EXIT AT INTERRUPT (IOAXI$)

Purpose:

Initiates an arbitrary device I/O operation with the referenced activity simulating an exit function, and controls the return to the program at the appropriate interrupt activity specified in the request packet.

Format:

```
    L       A0,pktaddr
    ER      IOAXI$
```

Parameters:

pktaddr	Address of device I/O packet (see Figure 6–2).

Description:

The activity performing the IOAXI$ request does not actually exit, but saving and restoring registers is eliminated (except for register A0), and the register set is reduced to the minor set only. The continuation of the IOAXI$ activity at the interrupt point is with the same activity-id; hence, the value in the int-act-id field is ignored for the IOAXI$ request.

6.9.4. FREE FORMAT DISC HANDLER

This handler is designed to format, read, and write disc packs in other than the standard 1100 series executive formats. It is an adaptation of the ADH to control multi-interrupting, byte-oriented, command chain disc subsystems with the 1100 series executive format handler operating on other drives of the same subsystem.

The I/O packet format for free format disc is as described in 6.9.1 and illustrated in Figure 6–2 with the following exceptions.

Word 3

The monitor-interrupt-activity-addr field is unused. No monitor operations to free format disc are permitted. When the packet is checked for the function string (word 4), any monitor operation that is encountered causes reference to the error mode return point.

Word 4

The time-out field specifying the number of six-second intervals is unused. All free format disc I/O operations are confined to one six-second time interval by the executive system.

The free format device I/O is limited to one external function (EF) access control word per packet request where the function access word may be up to eleven words in length. The EF access words may contain the command parameters as well as the command string. For example an operation to do a read may be as follows:

Word 0		Set file mask command
	1	– Seek command
	2	– Search command
	3	– Read command
	4	– Jump command
	5	– Set file mask command
	6–7	– Seek parameters
	8–9	– Search parameters

When one function operation has been found and a second function operation is indicated, the program is considered in logic error and causes a reference to the error mode return point.

Once the mode string has been accepted and the I/O is to be started at initiation of the EF, a delay occurs if the next operation is to be an output operation. This delay is done to ensure that the EF chain and parameters access word has been sent before the output access control word is issued. Otherwise, the output access control word would overlay the EF access control word. If an input operation follows the issuance of the function chain, no delay occurs in opening input (LIC).

Once the I/O access control word has been initiated by either a Load Input Channel (LIC) or Load Output Channel (LOC), another I/O access control word (ACW) may follow until the limit of six modes is reached. This method is not advised because no delays are done after the ACW has been initiated, and thus the new ACW would overlay the previous ACW before it was completed. As a practical limit, it is recommended that a single I/O operation following the EF be used per packet request.

The EF command chain is checked to ensure that the M field (multiple function string) of the MSA/disc command is not set. If in the command chain the M field is found to be set, a program logic error is assumed and causes a reference to the error mode return point.

The device address field of the MSA/disc command is cleared by the executive and the file associated device address is inserted.

The external status received by the executive is returned at the normal mode return point in register A1 where the external status is found in H2 of register A1. If during the operation, an error condition (MSA error, unit check) is detected by the executive from the external status, the auxilary status or sense byte (byte 0 and byte 1) is requested from the MSA or control unit and returned with the original status at the normal return point. The external status occupies H2 of register A1 with the secondary status in H1 of register A1. The sense bytes are returned in the A Format, that is, Q1 = sense byte 0, Q2 = sense byte 1.

When the external status specifies a busy status, the executive waits for the control unit end external status and upon receiving it, causes reference to be made to the normal mode return point with the original busy external status in H2 of register A1.

When an external status specifies a channel end without an accompanying device end, the executive waits for the device end status and then causes reference to the normal mode return point with the device end external status in H2 of register A1.

6.10. STATUS CODES

Upon completing an I/O request, a status code is stored in S1 of word 3 of the I/O packet indicating the completion status. All status codes from 20_8 to 37_8 are considered error conditions where either the activity is terminated or the activity reentry point is reset to the error contingency point (previous reference to IALL$). If interrupt activity was specified by the executive request, the interrupt activity has the reentry point set to the contingency point. For status code 28_8 no status is stored in the packet. An illegal value for the packet address in register A0 (detected during the initial checking routine) results in an error type 4 and code 2. If detection is within executive request control, no interrupt activity is created and the main activity is reactivated at the contigency point.

Whenever a request for I/O is in error and the request causes an interrupt activity, it is the interrupt activity which is reset to the contingency entry point. The submitting activity is reset only when an illegal value in the control register is detected before the interrupt activity is created. When an illegal value is detected in register A0 during an I/O request, the interrupt activity exists and a status code of 23_8 is passed to the contingency routine.

The status codes are:

Octal Code — Description

0 The request has been completed normally. If data transfer is involved, the count is given in H2 of word 4.

1 EOF block detected on magnetic tape.

- Answer of E was given to an I/O error message.
- EOF block was detected on magnetic tape.
- Block read drum function was truncated by encountering an end-of-block word.
- Block search read function was truncated by encountering an end-of-block word before the specified number of words were transfered.

2 End-of-tape mark encountered on magnetic tape on a read backward from load point or on a write. No transfer takes place for the read backward. The write is done in the normal manner. Subsequent writes are performed in the same fashion and, barring other problems, results in returning the same status code.

3 No find was made on a mass storage device search. The search was terminated by an end-of-block, end-of-track, end-of-position, or expiration of sufficient time to pass over the entire area of concern depending upon the physical device and type of search.

4 A nonintegral block was read from magnetic tape. The number of data characters accepted from the last word is indicated by S3 of word 3 of the I/O packet and is explained in 6.4.2.4.

5 An attempt was made to initiate a mass storage search or read from an area which is wholly or partially unassigned. If the starting address is legal, the read is truncated as reflected by the word count in the substatus field.

10 The area of the FASTRAND mass storage file being unlocked by this write or unlock request timed out in the locking list or a subsequent request by the same activity had a packet format error detected between the time of submitting the request and the time of servicing. Other requests by other activities for the area may have been honored in the interim. If the function is write, the transfer is not performed.

11 A nonrecoverable error has occurred, the suppress recovery mode is set for magnetic tape or an answer of G was given to an error message. If the suppress recovery mode is set, the EI status code is stored in register A1 of the interrupt activity control register set. All suppress recovery operations come back with this status.

Octal Code — Description

12 A read, or write error on magnetic tape has resulted in loss of position on the unit. This code is returned for all outstanding requests at the time the answer of B was entered in response to the I/O error message. Any subsequent request is honored but the lost position is maintained and no further program check-points are valid. This condition can be cleared by requesting that the tape unit be rewound to load point.

13 The peripheral unit was declared down either by an unsolicited operator keyin or in response to an error message typed after the normal recovery failed to resolve a malfunction.

20–37 See Appendix C.

40 The request is either in the process of being executed or is listed on the request queue for the particular channel.

7. FILE CONTROL

7.1. INTRODUCTION

The file control routines exercise centralized control over all files in the system. The primary functions of these routines are:

- Maintain a directory (master file directory) of both the catalogued and temporary files
- Control mass storage allocation as new files are assigned and existing files are expanded
- Provide the interface between the user programs and mass storage I/O device handlers (maintains a record of the absolute addresses of the file granules)
- Prevent assignment of or access to any exclusively assigned file by any run other than the run to which the file has been exclusively assigned
- Automatically move files from mass storage to magnetic tape as available mass storage space becomes exhausted (known as rollout) and automatically retrieve these files when needed (called rollback)
- Provide the means which enable the user to obtain information on the assignment, contents, and facilities of a file

7.2. FILE ORGANIZATION

7.2.1. MASTER FILE DIRECTORY

For files which are to be retained beyond run termination, entries are constructed containing the identification and characteristics of each file and are maintained by the system in a master file directory. The process of entering a file in the master file directory is referred to as cataloging and effected by the @ASG control statements.

The information contained in each file entry includes the following:

- External name of the file including qualifiers
- Project identity from the @RUN control statement
- Account number from the @RUN control statement
- Date on which the file was catalogued
- Activity of the file including the date of last reference
- Usage authorization
- Recording mode (magnetic tape only)

- Granularity and number of granules assigned (mass storage only)
- Number of reels of tape and tape reel numbers (tape only)
- Linkage to the granule description (mass storage)

See Section 22 for a description of the master file directory structure and the means of retrieving information from it.

7.2.2. MASS STORAGE ALLOCATION

Mass storage is accessed by the executive in two ways:

(1) FASTRAND format – A FASTRAND–formatted file is accessed by sectors (28 words in length) which are allocated in granules of track and positions. A track is 64 addressable areas of 28 words each or 1792 words of storage. A position is 64 tracks (4096 addressable areas or 114,688 words).

(2) Word addressable format – A word addressable formatted file is accessed on a word basis and is allocated in the same manner as FASTRAND format.

When a mass storage file is initially assigned, only the number of granules initially requested in the @ASG control statement are allocated. After that, only those granules needed to service a given expansion request are automatically assigned. For example, if the initial request was three tracks and a one track write was requested starting at relative address 256, an assignment is made for tracks 1, 2, 3, and 5. Track 4 is not assigned until a reference is made to a relative address from 192 to 255.

The file supervision routines automatically effect the assignment of additional increments of mass storage space as required to satisfy the needs of the worker programs. The space availability function also handles the release of granules to the available status. Release of any part of a granule causes the release of the entire granule area.

Since files can be released a granule at a time, it is possible to end up with an empty file catalogued in the system with a master file directory item and no allocated space.

7.2.3. FILE ADDRESSING

As an extension to the master file directory, the executive maintains tables specifying the location of the various granules allocated to a given filename. These tables are stored in sector-sized areas of mass storage and are used by the device handlers to convert the relative location furnished in the request to absolute hardware locations. For example, a request to read at address 129 of a file with track granularity would refer to the second sector of the third track assigned to the file. This reference table allows voids and overlapping various types of mass storage within a file.

Unsolicited console input messages are available to control mass storage availability. This allows mass storage to be taken from and returned to the configuration without forcing an initial boot which would cause all user files to be deleted from the system. It allows users to reserve units for their own assignments and allows absolute addressing for I/O on those units.

Mass storage is defined to be in one of four states. They are:

UP – Up indicates that a mass storage unit is fully accessible by the operating system. This status allows the executive to read, write, and allocate on this unit.

SU – Suspend status indicates that a mass storage unit is accessible for I/O, however, the executive cannot allocate any space on this unit.

RV – Reserve status indicates that a mass storage unit is accessible only by absolute assignments and absolute I/O requests.

DN – Down status indicates that a mass storage unit is not to be used for any purpose by the system.

7.2.4. EXCLUSIVE USE OF FILES

The file supervisor routines allow assignment of mass storage files to any number of runs at one time providing the exclusive use option is not exercised on the @ASG control statement. This option delays the assignment of a file until no other run has that file assigned to it and ensures that other runs are delayed until a run releases the needed exclusively-assigned files.

All magnetic tape files are exclusively assigned regardless of the presence or absence of the option.

The read-and-lock and unlock functions are available at the handler level where logically contiguous areas (successive relative addresses) can be exclusively assigned to allow other runs simultaneous access.of all the unlocked portion of the file. The complete definition of the various functions involved and the timing limits to be considered are given in Section 6.

7.2.5. ROLLOUT AND ROLLBACK OF FILES

Depending upon the amount of available FASTRAND-formatted mass storage, the degree of use given to cataloguing files on mass storage, and the manner in which FASTRAND-formatted files are assigned, there may occur during normal operation the necessity to obtain additional space on FASTRAND mass storage by rolling out catalogued files to magnetic tape. This feature is provided automatically by the executive. The points at which rollout is turned on and off are expressed as system generation parameters.

Rollout to magnetic tape occurs when the request for allocation reduces the available mass storage below a system generation threshold.

The rollout routine utilizes the file activity as part of the criterion for file rollout eligibility.

A request to assign a rolled out FASTRAND-formatted file causes the executive to request mounting of the proper magnetic tape, unless already mounted, and it automatically retrieves the file back to FASTRAND-formatted mass storage. See Section 19 (SECURE processor actually performs the rollout and rollback).

7.2.6. RETRIEVING FACILITY ASSIGNMENT (FITEM$)

Purpose:

Provides a method to obtain a variable amount of information on file or facility assignments.

Format:

```
LA   A0,(pkt-lngth,pktaddr)
ER   FITEM$
```

Description:

A filename must be placed in the first two words of the information packet.

The remaining words of the packet are filled as a result of the FITEM$ request.

The minimum packet length is nine words; the maximum packet length is dependent upon the equipment type:

Equipment	Length
Word addressable mass storage, arbitrary devices, communications devices, whole unit mass storage	9
FASTRAND mass storage	10
Magnetic tape removable disc	13

If the pkt-length is 0377777_8, the maximum amount of information available for the equipment type is transfered to the packet. If the pkt-length given is less than 9 or greater than the maximum for the equipment type, only nine words are transfered to the packet and an error status is returned in register A0 (see following).

Rejection of the FITEM$ request occurs only if the relative packet address specified in the request packet is invalid; that is, the address falls outside the user's bounds, or the span of the FITEM$ packet violates the user's bounds.

The status codes applicable to FITEM$ requests are (returned in S1 of register A0):

1_8 – The requested packet length exceeded the allowable maximum.

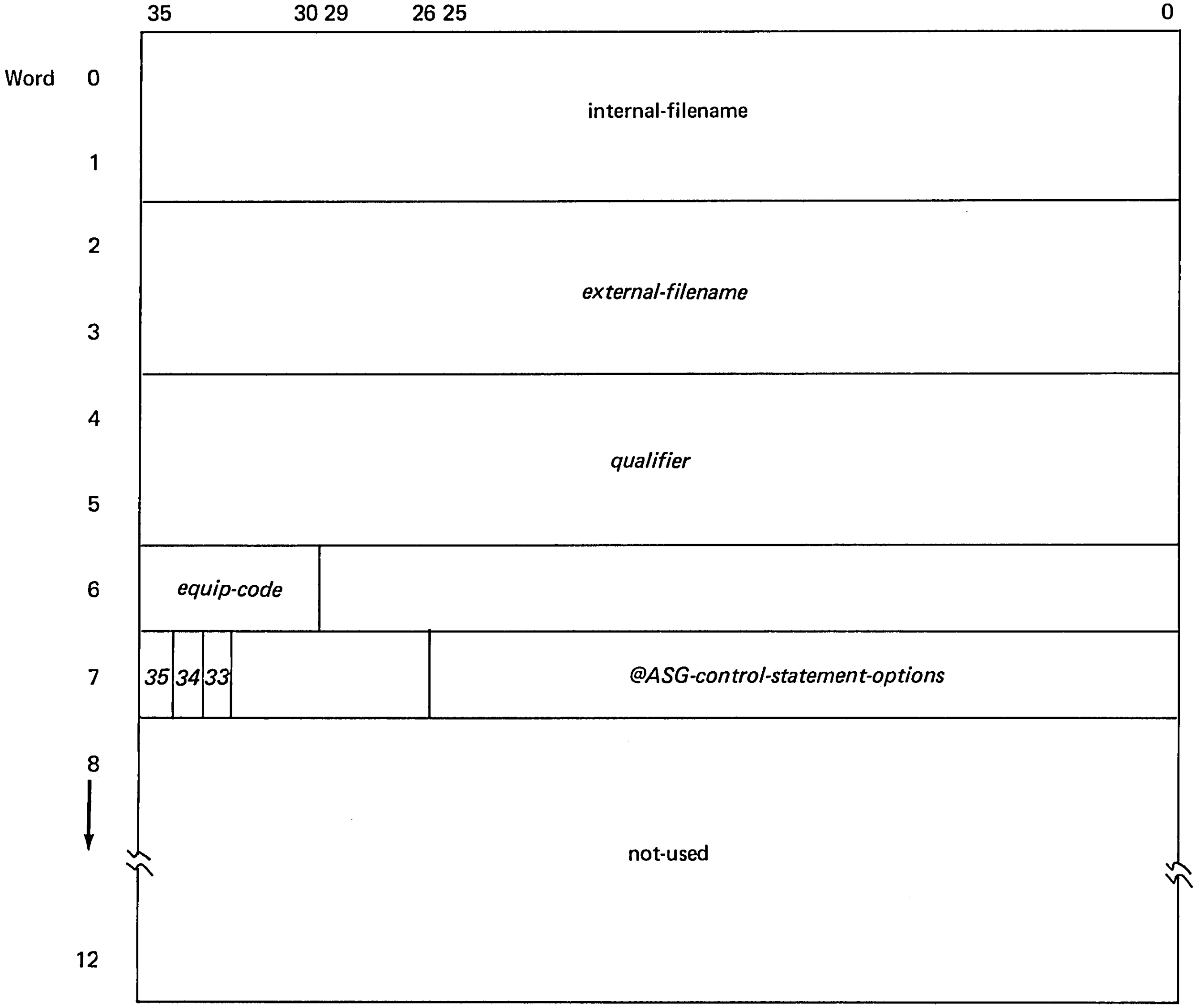
2_8 – The requested packet length was less than the allowable minimum.

Pktaddr is the address of a packet whose general format is:

Word	35 0
0 1	internal-filename
2 3	*external-filename*
4 5	*qualifier*
6 7 8 9 10 11 12	Device dependent See 7.2.6.1.through 7.2.6.6.

7.2.6.1. UNIT RECORD AND NONSTANDARD PERIPHERALS

The FITEM$ request packet format is:



Word 6	
equip-code	See Appendix E
Word 7	
bit 35	If set, system has label check
bit 34	If set, file assigned as a temporary file
bit 33	If set, assigned unit is in a downed state
@ASG-control-statement-options	Indicates the options specified on the @ASG control statement (see 3.7.1) that assigned the equipment. Master bit notation is used; that is, a 1 in bit 25 indicates the A option was specified, a 1 in bit 24 indicates the B option was specified, and so forth.

7.2.6.2. FASTRAND MASS STORAGE PERIPHERALS

The FITEM$ request packet format is:

Word	S1	S2	S3	S4	T3
0	internal-filename				
1					
2	*external-filename (file-portion)*				
3					
4	*qualifier*				
5					
6	*equip-code*	*file-mode*	*granularity*	*relative-F-cycle-nbr*	*absolute-F-cycle-nbr*
7	*35* | *34* | *33*	*@ASG-control-statement-options*			
8	*initial-granule-count*			*maximum-granule-count*	
9	*largest-track-reference*			*highest-granule-nbr*	
10	not-used				
11					
12					

Word 6

equip-code See Appendix E

file-mode

Bit 29 set — Exclusively assigned file
Bit 28 set — Read key is needed
Bit 27 set — Write key is needed
Bit 26 set — Read-only file
Bit 25 set — Write-only file
Bit 24 set — Word addressable drum

granularity — All zeros indicate position granularity; nonzero indicates track granularity

relative-F-cycle-nbr — Negative number is indicated by a 1 in bit 17.

Word 7

Same as word 7 in 7.2.6.1.

7.2.6.3. MAGNETIC TAPE PERIPHERALS

The FITEM$ request packet format is:

Word	S1	S2	S3	S4	S5	S6
0	internal-filename					
1						
2	*external-filename*					
3						
4	*qualifier*					
5						
6	*equip-code*	*file-mode*	*unit-count*	*relative-F-cycle-nbr*	*absolute-F-cycle-nbr*	
7	*35* *34* *33*		*@ASG-control-statement-options*			
8	*total-reel-count*	*logical-channel*	*noise-constant*	*mode-settings*		
9	*subsystem-1*		*unit-1*	*subsystem-2*		*unit-2*
10	*expiration-period*		*reel-index*	*files-extended*	*blocks-extended*	
11	*current-reel-nbr*					
12	*next-reel-nbr*					

Word 6

equip-code	See Appendix E
file-mode	Bit 29 set – Exclusively assigned file Bit 28 set – Read key is needed Bit 27 set – Write key is needed Bit 26 set – Read-only file Bit 25 set – Write-only file Bit 24 set – Word addressable drum
unit-count	Number of units assigned
relative-F-cycle-nbr	Negative number is indicated by a 1 in bit 17.

Word 7

Same as word 7 in 7.2.6.1.

Word 8

mode settings

The mode settings depend upon the equipment. For all tape units except UNISERVO 12/16 tape units, the modes are:

Bits Set	Description
17 –15	Not used; contents ignored
14	Even parity
13 and 12	High density
13 only	Medium density
12	Low density
11	Hardware translate
10	Software translate
9	User I/O recovery
8 – 6	Not used; must be zero

For seven-track UNISERVO 12/16 tape units, the mode settings are:

Bits Set	Description
17	Eight-bit packed MSA data transfer format
16	Six-bit packed MSA data transfer format NOTE: If bits 17 and 16 are both zero, it indicates quarter-word MSA transfer.
15	High density
14	Medium density NOTE: If bits 15 and 14 are both zero, it indicates low density.
13	Data converter (0-on; 1-off)
12	Parity (0-even; 1-odd)
11	control unit translator
10	Not used; must be zero
9 – 8	Must be set to mode set for hardware
7 – 6	Not used; must be zero

For nine-track UNISERVO 12/16 tape units, the mode settings are:

Bits Set	Description
17 – 16	Same as seven track
15 – 14	Must be set to mode set for hardware
13 – 12	Not used; must be zero
11	Density (0–1600 FPI; 1–800 FPI)
10	Not used; must be zero
9 – 8	Must be set to mode set for hardware
7 – 6	Not used; must be zero

Word 10

reel-index — Index to the current reel of a multireel file.

file-extended — Count of the number of hardware EOF marks encountered (applicable only to UNISERVO 12/16 tape units).

block-extended — UNISERVO 12/16: Count of the number of physical tape blocks encountered since load point or last EOF mark.

All other tape units: Count of the number of physical tape blocks encountered since load point.

7.2.6.4. MAGNETIC DRUM PERIPHERALS

The FITEM$ request packet format is:

Word	S1	S2	S3	S4	T3
0	internal-name				
1					
2	*external-name*				
3					
4	*qualifier*				
5					
6	*equip-code*	*file-mode*		*relative-F-cycle-nbr*	*absolute-F-cycle-nbr*
7	*35* | *34* | *33*		*@ASG-control-statement-options*		
8	*length-of-file-in-words*				
9	*maximum-file-length*				
10	not-used				
11					
12					

Word 6

Same as word 6 in 7.2.6.3. For an absolute drum, the format of word 6 is:

S1	S2	S3	S4	T3
equip-code	*file-mode*	*total-nbr-of-units*	*relative-F-cycle-nbr*	*absolute-F-cycle-nbr*

Word 7

Same as word 7 in 7.2.6.1.

7.2.6.5. COMMUNICATIONS PERIPHERALS

The FITEM$ request packet format is:

Word	S1	S2	S3	S4	S5	S6
0	internal-filename					
1						
2	*external-filename*					
3						
4	*qualifier*					
5						
6	*equip-code*	*execution-mode*	*carrier-type*	*line-speed-in-bits-per-second*		
7	*bits/char*	*CTM-code*	*EOM/ETX-character*		*CTM-options*	
8	*CTM-speed*	*line-status*				
9 → 12	not-used					

Word 6

equip-code — See Appendix E

execution-mode — CTMC only:
0_8 – Half word
1_8 – Quarter word

carrier-type — 0_8 – Leased line
1_8 – Telephone
2_8 – TELEX*

Trademark of Western Union Telegraph Co.

Word 7

bits

Number of bits per character (including parity)

CTM-code

Indicates type of CTM:
0_8 – Standard (monitor)
1_8 – NASA
2_8 – GSA3EI (external)
4_8 – CTM IV (external interrupt with status word)
6_8 – CTM VI (external interrupt with status word)
7_8 – CTM VII (external interrupt with status word)

CTM-options

Options are (indicated bit set):
Bit 10 – Block parity absent (0) or present (1)
9 – Continuously emitting modem
8 – Release function indicator (for ring interrupt feature)
7 – ECM/ETX character passed as status word
6 – Even (0 bit) or odd (1 bit) parity indicator
5 – Block parity character transferred to CPU
4 – Space-to-mark transition
3 – Block parity, character parity, or late input acknowledge
2 – Ring indicator
1 – Carrier off
0 – Send data with idle character

Word 8

CTM-speed

Units speed. Codes are:
0_8 – Low
1_8 – Medium
2_8 – High synchronized
3_8 – TELPAK*
4_8 – Parallel

line-status

Indicates status of line as follows:

Bit 20 = 0 – Line is not initialized for idle line monitor.
= 1 – Line is initialized for idle line monitor.

Bit 21 = 0 – Dial CLT is operable.
= 1 – Dial CLT is inoperable.

Bit 22 = 0 – Output CLT is operable.
= 1 – Output CLT is inoperable.

Bit 23 = 0 – Input CLT is operable.
= 1 – Input CLT is inoperable.

Bit 24 = 0 – CLT group (input, output, and dial) is not reserved.
= 1 – CLT group (input, output, and dial) is reserved.

Bit 25 = 0 – CLT group (input, output, and dial) is not assigned.
= 1 – CLT group (input, output, and dial) is assigned.

**Trademark of American Telephone and Telegraph Company.*

7.2.6.6. DISC PERIPHERALS

The FITEM$ request packet format is:

Word	S1	S2	S3	S4	S5	S6
0	internal-name					
1						
2	*external-name*					
3						
4	*qualifier*					
5						
6	*equip-code*	*file-mode*	*granularity*	*relative-F-cycle-nbr*	*absolute-F-cycle-nbr*	
7	*35* *34* *33*	*@ASG-control-statement-options*				
8	*initial-granule-reserve*			*maximum-granule-reserve*		
9	*largest-track-reference*			*highest-granule-reference*		
10				*total-pack-count*	*nbr-of-units*	
11	not-used					
12						

Word 6

Same as word 6 in 7.2.6.3.

Word 7

Same as word 7 in 7.2.6.1.

7.2.7. ALTERNATE METHODS OF RETRIEVING FACILITY ASSIGNMENT SYNOPSIS (FACIL$ AND FACIT$)

These linkages obtain a subset of the facility assignments previously defined in 7.2.6.

To obtain the first nine words of the FITEM$ packet, the following linkage is used:

```
L,U  A0,pktaddr
ER   FACIL$
```

To obtain the first ten words of the FITEM$ packet, the following linkage is used:

```
L,U  A0,pktaddr
ER   FACIT$
```

7.2.8. TAPE FILE INITIALIZATION (TINTL$)

Purpose:

Causes a specified tape file to be logically rewound so that a subsequent pass can be made from the load point of the first reel.

Format:

```
L    A0,(function,pktaddr)
ER   TINTL$
```

Description:

Pktaddr is the address of a two- (or three) word request packet whose format is:

	35 0
Word 0	filename
1	
2	for-use-on-a-function-1-or-2-request

Inclusion of a function specification (0–2) in H1 of the register A0 indicates the following:

Function	Description
0	Standard initialization of a tape file
1	Standard initialization of a tape file occurs and the reel number of the file's initial tape reel is placed in word 2 (the third word) of the request packet.
2	If there is no reel number in the initial reel position, the reel number specified in word 2 (the third word) of the request packet is placed therein, and normal tape file initialization occurs. If a function 2 request is made, however, and the initial reel position already contains a reel number, the run is placed in error mode.

Access to the reel currently in use is closed. If the initial tape of the file is mounted on the first unit, the tape is simply rewound. Otherwise, the tapes on the units involved are rewound, and a LOAD message is issued in order to have the initial reel mounted. If there are two units involved, they may be switched in order to ensure that the initial reel is mounted on the first unit.

7.2.9. TAPE SWAPPING (TSWAP$)

Purpose:

Closes access to the current reel of a tape file and requests loading of the next reel of the file (unless special action is requested).

Format:

Word	
0 1	filename
2	for-use-on-a-function-1-or-2-request

```
L     A0,(function,pktaddr)
ER    TSWAP$
```

Description:

Pktaddr is the address of a two- (or three-) word packet whose format is:

Inclusion of a function in H1 of the register A0 indicates the following:

Function	Description
0	Standard tape reel swap
1	Standard tape reel swap occurs and the reel number of the tape swapped to is placed in word 2 (the third word) of the request packet.
2	A nonstandard request. A request is made to mount the reel specified in word 2 (the third word) of the request packet. If this reel is not currently recorded as part of the file, it is added as the last reel.

Access to the reel currently in use is closed and the reel is rewound. A request is issued to mount the next reel of the file. If two units are involved in the assignment, the LOAD request specifies the unit other than that which was just in use.

7.3. TAPE LABELING

The executive provides a mechanism which:

(1) honors labeled tapes

(2) automatically creates a first file header label on a prelabeled blank (allocates the tape to the file being written)

(3) gives the user the ability to write ANSI-standard tape labels on a prelabeled tape.

To prelabel a tape the user writes a volume header and a skeleton format first file header at the begining of a tape reel. The volume header label and first file header are each 80 characters long and are described in Tables 7–1 and 7–2, respectively.

The reel number, if entered on the @ASG control statement, must agree with the reel number in the volume header. If they do not agree, an error message is added to the user's print file and the run is put in error mode.

After a reel is allocated, the qualifier and filename (or project-id and filename) on the @ASG control statement requesting the reel must agree with the qualifier and filename on the first file header. If they do not agree, the run is terminated.

If the file expands to a new reel, the retention period expiration-period parameter on @ASG control statement (see 3.7.1) on the new reel becomes one of the following:

(1) If the expansion occurs on the original assignment, the retention period is the same as for the other reels.

(2) If the expansion occurs on a subsequent assign with no retention field given, the new reel gets the system standard retention period.

(3) If the expansion occurs on a subsequent assign with a retention period stated, the retention period is the number of days stated on the @ASG control statement starting from the date the new reel was allocated.

Reels become logical blanks after the expiration date has elapsed.

The label blocks are protected from normal user I/O. A read or move backward function issued by the user does not allow him to read any of the header label fields. When the user issues a read or move backward from his first data block, he receives an 2_8 status in his I/O packet, informing him that he has reached load point.

A rewind function or @REWIND control statement issued by the user causes the tape reel to be rewound to load point. When the user I/O request is issued, the tape is moved forward until it is positioned on the user's first data block or the first block past the tape mark.

Field Contents	Field Length in Characters	Description
VOL	3	Recognition sentinel
1	1	Fixed by standard
reel number	6	Six alphanumeric characters identifying the physical reel
blank or nonblank	1	Nonblank indicates restricted access, as the tape reel is privately owned
blanks	26	Unrequired available space
account number	14	Account number of owner if this is a privately owned tape reel (UNIVAC uses a maximum of 12 characters left-justified, space filled)
blanks	28	Fixed by standard
1	1	Fixed by standard

Table 7–1. Volume Header Label Field Description for Table Labeling

Field Contents	Field Length in Characters	Description
HDR	3	Recognition sentinel
1	1	Fixed by standard
filename	17	Left-justified filename. The first 12 characters are referenced by the executive system
UNIVAC	6	Fixed as set identifier when referenced by system
0001	4	Reel sequence number within a file
0001	4	File sequence number within a reel which is fixed at 1
0001 00	4 2	Generation and version numbers which are fixed at 1 and 0
creation date	6	A blank followed by two characters for the year, followed by three characters for the day (001 through 366) within the year
expiration date	6	Same format as creation date field. The date after which this tape reel may be considered as available for reallocation
accessibility	1	A space indicates unlimited access to this reel and 15_8 – This reel is catalogued (on tape) 35_8 – This reel is catalogued, with read key 55_8 – This reel is catalogued, with write key 75_8 – This reel is catalogued, with read and write keys
block count	6	Fixed at zeros
qualifier	13	Used by the executive system (UNIVAC uses a maximum of 12 characters left-justified, space filled)
blanks	7	Fixed by standard

Table 7–2. First File Header Label Field Description for Table Labeling

The master file directory may indicate that tape reels have been allocated to a catalogued file before the HDR1 block on the reel has been updated to show that the reel is allocated or HDR1 blocks may have been written to indicate that a reel has been allocated to a catalogued file for which no master file directory items exist. The following examples illustrate how these situations may occur:

(1) Tape reel numbers specified on a @CAT control statement are entered in the master file directory, but the tape labels are not changed to show that they are allocated. Tapes must be assigned to a run before the labels may be changed, because tape labels are not filled in to show that the tape is allocated until the first I/O request to a reel is initiated. Using the @CAT control statement to catalogue tape files is not advised since tapes which are catalogued in the master file directory but not yet referenced on an I/O request are regarded as blank by the system.

(2) If a run containing an @ASG,C of a tape file terminates abnormally after the tapes have been referenced, the tape file is not catalogued but the HDR1 block shows the tape as allocated to a catalogued file which does not exist in the master file directory. These tape reels cannot be referenced by means of a temporary assignment. Master file directory links can be reestablished with any of the standard cataloguing control statements (@CAT, @ASG,C, @ASG,U).

(3) Using a @DELETE control statement for a tape file eliminates a catalogued file without updating the HDR1 block to show the reels as unallocated.

7.3.1. READING AND WRITING TAPE LABEL BLOCKS (LABEL$)

Purpose:

Enables the user to read or write any label block in the first label group on the volume except the VOL1 block.

Format:

```
L,U  A0,pktaddr
ER   LABEL$
```

Description:

Pktaddr is the address of packet whose format is:

		S1	S2	S3	H2
Word	0	ASCII-Fieldata translation	write-EOF-or label block		label-buffer-addr
	1	filename			
	2				

where:

ASCII-Fieldata-translation

If equal to 1, indicates ASCII format label block.

If equal to 0, the read packet is to be translated to Fieldata format or the write packet is to be translated from Fieldata format.

write-EOF-or-label-block

If equal to 2, write an EOF mark.

If equal to 1, write a label block.

If equal to 0, read a label block.

label-buffer-addr

Label buffer address where a 20-word buffer is required for read and write label requests; the buffer is not required for write EOF requests.

Control is returned to the user at the location immediately following the LABEL$ request after the request has been completely processed. Register A0 contains the following:

S1 If set to 0_8, indicates normal completion; if set to 40_8, indicates abnormal completion (check S2 and S3).

S2 Contains I/O status (if 0_8, all I/O has completed normally; if a nonzero value, see status for abnormal I/O in Appendix C).

S3 If 1_8, indicates invalid label buffer address.

If 2_8, indicates that a request was made to read a label following a request to write a label.

If 4_8, indicates invalid filename.

If 6_8, indicates an attempt was made to write on a tape file that was not available for writing or an attempt was made to write a label following a request to read a label.

If 10_8, indicates an invalid request (not 1_8, 2_8, or 4_8), or a write EOF request following a read of a label, or a write EOF request before HDR1 has been written.

H2 The label buffer address originally supplied by the user.

7.4. DISC LABELING

A label record is written on a disc pack when it is prepared by the executive (see 6.7) for use as mass storage. This record is only accessable to and used exclusively by the executive. It contains the pack identity and all the information necessary to establish executive control over the pack. The pack-id field of the label record is used at assign time to guarantee that the requested pack is received (see 3.7.1).

8. FILE UTILITY ROUTINES (FURPUR)

8.1. INTRODUCTION

In addition to the executive control statements, there is a set of control statements recognized by the executive as calls for the file utility routines (FURPUR). When the executive encounters a FURPUR control statement, it loads the FURPUR processor. FURPUR continues to process control statements until signalled by the executive that the next statement is not a FURPUR control statement.

Table 8–1 summarizes the function of each FURPUR control statement.

FURPUR Control Statements	Description
@CHG	Changes element name, filename, version name, read key, write key, and mode of a file. (See 8.2.15.)
@CLOSE	Writes two hardware EOF marks on a magnetic tape file and rewinds the tape. (See 8.2.10.)
@COPIN	Copies elements from an element file located on magnetic tape into a program file on FASTRAND-formatted mass storage. (See 8.2.2.)
@COPOUT	Copies a program file, or selected elements from a program file, located on FASTRAND-formatted mass storage onto a magnetic tape file in element file format. (See 8.2.3.)
@COPY	Copies a file or element from one file to another. (See 8.2.1.)
@CYCLE	Sets the maximum range of absolute F-cycle numbers to be retained for specified files which are listed in the master file directory or sets the maximum number of element cycles to be retained for specified symbolic element. (See 8.2.16.)
@DELETE	Drops catalogued files or marks elements in a program file as deleted. (See 8.2.7.)
@ENABLE	Clears the disable flags for catalogued files. (See 8.2.17.)
@ERS	Resets next write location to the first sector of the text area, clears the table of contents, and returns to the system all FASTRAND-formatted mass storage allocated to a program file. (See 8.2.6.)
@FIND	Locates an element in a magnetic tape file (file must be in element file format) and positions the file before the element's label block. (See 8.2.13.)
@MARK	Writes two hardware EOF marks on a magnetic tape file and positions the tape between the EOF marks. (See 8.2.9.)

Table 8–1. Summary of FURPUR Control Statements (Part 1 of 2)

FURPUR Control Statements	Description
@MOVE	Moves a magnetic tape file forwards or backwards over a specified number of EOF marks. (See 8.2.4.)
@PACK	Rewrites an entire program file, removing all elements marked as deleted from the program file. (See 8.2.14.)
@PCH	Punches program file elements into 80-column cards. (See 8.2.12.)
@PREP	Creates an entry point table from the preambles of the nondeleted elements of a program file. (See 8.2.11.)
@PRT	Provides a listing of the master file directory items for catalogued files, the table of contents of a program file, or the text of a symbolic element (see 8.2.5). Listings of absolute or relocatable elements may be obtained using the LIST processor (see 18.2.5).
@REWIND	Rewinds magnetic tape files back to the load point of the first reel. (See 8.2.8.)

Table 8–1. Summary of FURPUR Control Statements (Part 2 of 2)

8.1.1. COMMON INFORMATION

The operand fields may contain a filename (see 2.6.1), an element name (see 2.6.4), or a parameter value, depending on the control statement and its use.

If the filename in any parameter is identical to that specified in the immediately preceding parameter, coding a period in the parameter indicates to the FURPUR processor that the same filename is intended. Omitting the filename completely, including the period, indicates to the FURPUR processor that TPF$ is intended (only valid if the parameter normally specifies a file that resides on FASTRAND-formatted mass storage).

As with most other system processors, the FURPUR processor automatically assigns any catalogued file that was not assigned when the FURPUR control statement is encountered. In many cases, the FURPUR processor requires exclusive use of a file, and it places user files in the exclusive-use state as necessary to carry out the specified operation. After use, the FURPUR processor automatically returns all files to the assigned status the file has except when the function of the FURPUR control statement wsa to alter the file status. Temporary files, except TPF$, must be assigned by the user.

In most instances, the meanings of options used in FURPUR control statements vary with the statement. The meaning, however, of the following options is the same for all FURPUR control statements:

Option	Description
A	Process absolute elements
C	Do not exit through ERR$ if an error is encountered. The FURPUR processor would go on to process the next command or parameter field if more than two parameter fields are permitted as in the case of the @DELETE,C control statement. The C option can always be used, even when the discussion of the options specify 'no options'.
R	Process relocatable elements
S	Process symbolic elements

The FURPUR control statements are device dependent as well as file-type dependent. Program files exist only on FASTRAND-formatted mass storage, and element files exist only on magnetic tape. Thus, the statement 'the @PCH control statement is used to punch program file elements into 80-column cards' necessarily implies a mass-storage-to-card transfer. If the program file has been copied onto magnetic tape, the @PCH control statement cannot be used to punch elements into cards. The program file elements must be returned to FASTRAND-formatted mass storage priority to the attempt to execute the @PCH control statement.

8.1.2. SIMULTANEOUS USE OF FILES

The FURPUR processor, in common with other system processors, such as the collector, can directly access catalogued program files, even though they have not been assigned to the user's run. The mechanism which the FURPUR processor and the other processors use is the same; that is, a dynamic @ASG,AX (see 3.7.1) is done using a CSF$ request (see 4.8.1). These processors return each previously unassigned, catalogued file to its original status, using a dynamic @FREE control statement (see 3.7.4) with the appropriate options.

The X option in the @ASG,AX control statement means that the processor is requesting exclusive use of the file by the run which called the processor. This is done to make certain that no other runs currently in the system will add or delete elements, or otherwise tamper with the file, while the processor is attempting to determine the locations and contents of its various tables, pointers, and element texts.

If a dynamic @ASG,AX is attempted, and another run already has requested the program file assigned, the CSF$ request returns with status bit 18 set, which means: request on wait status for facilities. In this case, FURPUR prints a diagnostic indicating that the file cannot be assigned and terminates by means of ERR$.

8.1.3. MULTIREEL FILES

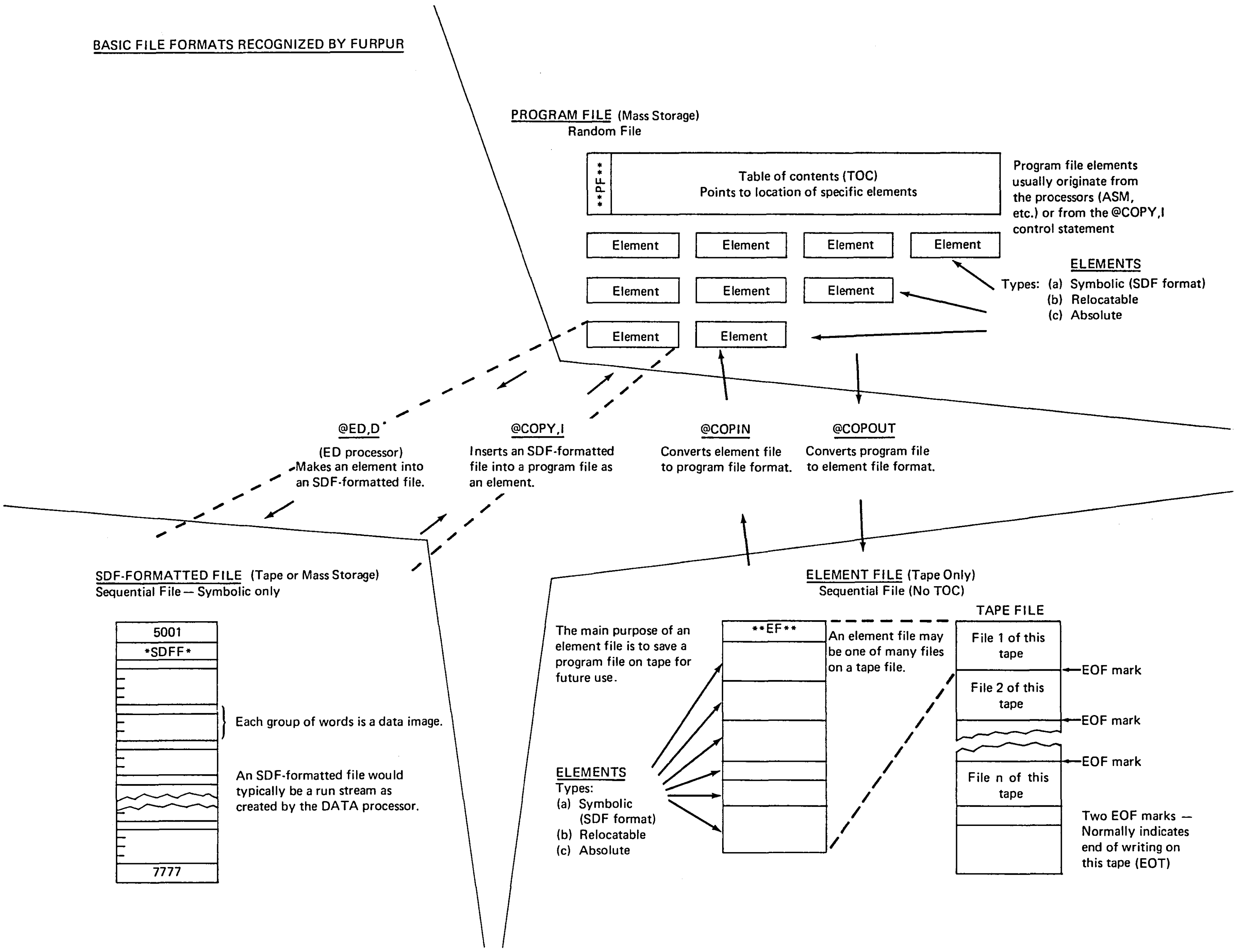
The FURPUR processor can handle multireel files. The @COPOUT control statement (see 8.2.3) automatically swaps reels when an end-of-reel condition is detected. The @COPOUT control statement writes a 14-word the end-of-reel sentinel which indicates to the @COPIN control statement (see 8.2.2) that the element being read extends onto a second reel.

The @REWIND control statement (see 8.2.8) returns the first reel of the file to the user when it returns control.

The @COPY control statement (see 8.2.1) also permits the reading and writing of multireel files.

8.1.4. BASIC FILE FORMATS

Figure 8–1 illustrates the relationships of files to each other. The exact formats have been simplified for clarity; for more detail see Section 24. The control statements illustrated are control statements that change the format of the files.



Program file elements usually originate from the processors (ASM, etc.) or from the @COPY,I control statement

Makes an element into an SDF-formatted file.

Inserts an SDF-formatted file into a program file as an element.

Converts element file to program file format.

Converts program file to element file format.

Each group of words is a data image.

An SDF-formatted file would typically be a run stream as created by the DATA processor.

The main purpose of an element file is to save a program file on tape for future use.

An element file may be one of many files on a tape file.

Two EOF marks — Normally indicates end of writing on this tape (EOT)

Figure 8–1. FURPUR Control Statements Used to Alter File Formats
]

8.2. FURPUR CONTROL STATEMENTS

Paragraphs 8.2.1 through 8.2.17 describe the various FURPUR control statements. The most frequently used control statements are presented first and the infrequently used control statements are presented last.

8.2.1. FILE COPYING (@COPY)

Purpose:

Copies a file or element to another file.

All parameters of the @COPY control statement are optional except name-1 and name-2.

Format:

```
@label:COPY,options   name-1,name-2,no.-of-files
```

Parameters:

options
See Table 8–2 for file options and Table 8–3 for element options. See 8.1.1 for additional information on the A, C, R, and S options.

name-1
Specifies the input file or element to be copied.

name-2
Specifies the output file into which the input file or element is to be copied.

no.-of-files
Used only for tape-to-tape copying of entire files. It specifies the number of input files to copy onto the output tape. If omitted, one file is copied onto the output tape. When an attempt is made to copy an empty file (two hardware EOF's), the copy operation is immediately terminated regardless of the contents of the no.-of-files parameter. The input tape remains positioned following the last EOF mark of the last file copied. The number of blocks in each file copied and the number of files copied are indicated in the output listing.

Option Character	Description
No option specified	FASTRAND-to-FASTRAND Copying – Overwrite one mass storage file (name-2) with the contents of another mass storage file (name-1), without regard to the files format. Tape-to-Tape Copying – Copies one or more files (depending on no.-of-files parameter) from the input tape to the output tape without regard to the file's format. No hardware EOF marks are written (see M option).
A, R, S	Copy the elements of the type specified from one program file and add them to another. Both program files must be located on FASTRAND-formatted mass storage. All elements of the type specified by the selected options are copied into the output file. Any combination of A, R, and S can be used.

Table 8–2. @COPY Control Statement, Options Applicable when Filenames are Specified (Part 1 of 3)

Option Character	Description
F	Copy the contents of one file into another file. Program and element files must not be copied using this option. The input file must be in SDF format. Reading of the input file is terminated by the SDF EOF mark. Block sizes for tape files must be 224 words. When the output file is a magnetic tape file, two hardware EOF marks are written following the file and the tape is positioned between the two EOF marks.
G	When used with the M option, copy a FASTRAND-formatted mass storage file to magnetic tape. When used without the M option, copy a magnetic tape file produced by the @COPY,GM control statement back onto mass storage. FASTRAND-to-Tape Copying – Each track of the file, beginning with relative track 0, is prefixed with its relative track number and written onto the tape. The @COPY operation is terminated after the highest track referenced in the file has been written to tape. The first block in the output file is a label block that indicates the file format (@COPY,G). The M option writes a EOF mark on the tape. Tape-to-FASTRAND Copying – The first two words of each tape block provides the relative track number into which the block (minus the first two words) is to be written. The @COPY operation is terminated when an EOF mark is encountered on the input file. Since track-size blocks of data are transferred on a @COPY,G operation without regard to format of the contents, the transfer is done relatively quickly and the files contents are not changed. The G option provides an efficient method of saving and recreating FASTRAND-formatted files.
I	Used to add an SDF-formatted file to a program file as a symbolic element. name-1 – Specifies the input file in SDF format. name-2 – Specifies the output file and element name. The SDF-formatted file being copied is entered into the program file (located on FASTRAND-formatted mass storage) as a symbolic element with an element cycle of 0. Reading of the input file (which may be either tape or FASTRAND-formatted mass storage) is terminated by an SDF EOF mark.
M	The option can be specified only when the output file is a magnetic tape file. FASTRAND-to-Tape Copying – used with the G option to copy a FASTRAND-formatted mass storage file to magnetic tape. Two hardware EOF marks are written on the tape following the file copied, and the tape is positioned between the two EOF marks. Tape-to-Tape Copying – Used without other options or with the N option for tape-to-tape copying of one or more files (depending on no.-of-files parameter). If more than one file is being copied, a hardware EOF mark is written on the tape following each file copied except the last, where two hardware EOF marks are written and the tape is positioned between the two.
N	Copy a magnetic tape file containing an abnormal frame count to another magnetic tape file or to a FASTRAND-formatted mass storage file. When the output file is tape, the M option may be used along with the N option to write hardware EOF marks.

Table 8–2. @COPY Control Statement, Options Applicable when Filenames are Specified (Part 2 of 3)

Option Character	Description
P	Used to copy all nondeleted elements from one program file and add them to another. Both program files must be located on FASTRAND-formatted mass storage files.
V	Copy one file into another file. The input and output files must not both be on magnetic tape or FASTRAND-formatted mass storage. FASTRAND-to-Tape Copying – Variable block size is assumed. The first word of the block (containing the block size) is stripped from the block before it is written into the tape file. Tape-to-FASTRAND Copying – A word containing the block size is prefixed to the block before it is written on FASTRAND-formatted mass storage. The input tape file must be terminated by a hardware EOF mark. A copy to or from FASTRAND-formatted mass storage always begins in sector 0 and each block starts in a new sector. If the block size is not divisible by 28, the excess words of the last sector contain random data.

Table 8–2. @COPY Control Statement, Options When Filenames Are Specified (Part 3 of 3)

Option Character	Description
A, R, or S	Copy the specified element in the input program file and add it to the output program file. The options represent the types of elements to be copied (one or more is needed). The element name can be changed by renaming it in name-2. Both input and output files must be on FASTRAND-formatted mass storage.

Table 8–3. @COPY Control Statement, Options Applicable When Element Names Are Specified

Description:

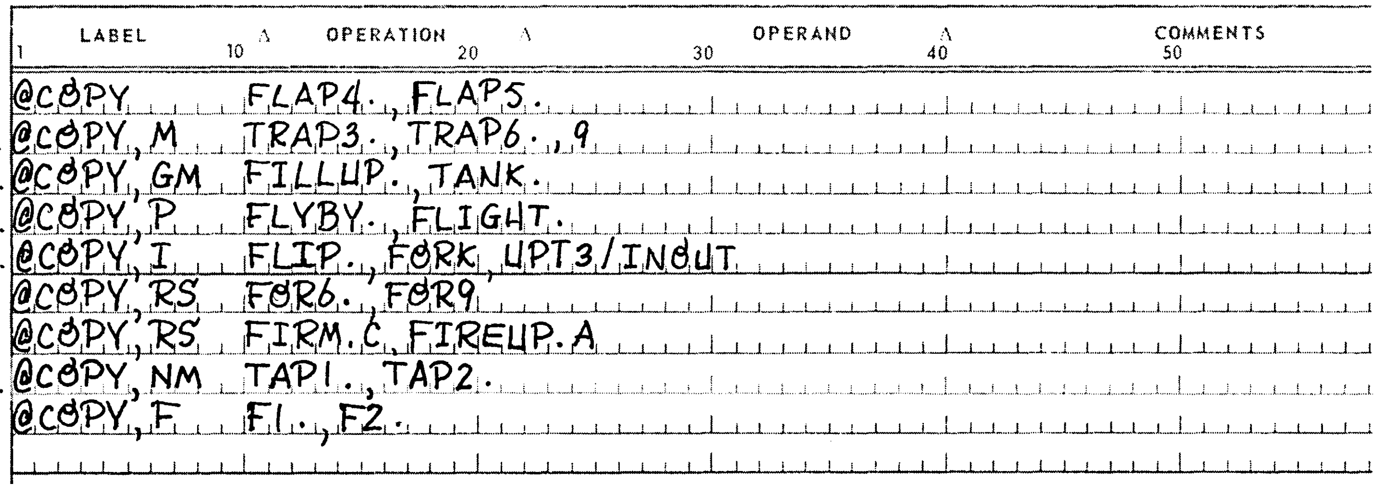
See 2.6.1 for additional information on specifying file names.

When any of the A, P, R, or S options are used, the procedure name entries are automatically added to the output file's procedure name table. If a relocatable element is copied, the output file's entry point table is destroyed and the @PREP control statement (see 8.2.11) must be used to recreate it.

Before doing any copy operation from tape, it may be necessary to execute a @MOVE control statement (see 8.2.4) to position the tape beyond some EOF mark. No @COPY operation will move backward or forward over an EOF mark prior to the start of the copy.

Examples:

In the following examples, tape filenames start with a T, and FASTRAND-formatted mass storage filenames start with an F.



```
   LABEL      ∧   OPERATION   ∧              OPERAND     ∧              COMMENTS
 1          10              20             30          40            50
1. @COPY        FLAP4.,FLAP5.
2. @COPY,M      TRAP3.,TRAP6.,9
3. @COPY,GM     FILLUP.,TANK.
4. @COPY,P      FLYBY.,FLIGHT.
5. @COPY,I      FLIP.,FORK,UPT3/INOUT
6. @COPY,RS     FOR6.,FOR9
7. @COPY,RS     FIRM.C,FIREUP.A
8. @COPY,NM     TAP1.,TAP2.
9. @COPY,F      F1.,F2.
```

1. The contents of FASTRAND-formatted mass storage file FLAP4 is copied into FASTRAND-formatted mass storage file FLAP5 over writing any previous contents of the FLAP5 file.

2. The nine files which form magnetic tape file TRAP3 are copied into magnetic tape file TRAP6. Each file in output file TRAP6 is separated by EOF marks as directed by the M option. The last file copied into the output file is followed by two EOF marks and the output file is positioned between the two final EOF marks.

3. Copy the contents of FASTRAND-formatted mass storage file FILLUP into magnetic tape file TANK. Two EOF marks are written at the end of the output file (TANK) and the tape is positioned between the two EOF marks (M option). Since file TANK is in @COPY,G format, the @FIND and @COPIN control statement cannot be used to access the file; however, @COPY,G format makes more economical use of time and space. The entire file, as it was before this operation was initiated, including all tables of contents and deleted elements, is reproduced when the file is returned to FASTRAND-formatted mass storage using a @COPY,G control statement. Do not attempt to merge two program files, each of which were saved on tape using the @COPY,GM control statement because the second file would overlay the first.

4. The nondeleted elements of program file FLYBY are copied into program file FLIGHT.

5. The contents of input file FLIP, which is in SDF format, are copied into output file FORK in program file format. Input file FLIP is entered in FORK as an element having UPT3 as its element name and INOUT as its version name. It is set at element cycle 0.

6. The nondeleted relocatable and symbolic elements (R and S options) located in program file FOR6 are copied into program file FOR9.

7. The nondeleted relocatable and symbolic elements with element name C (version name of spaces) in program file FIRM are copied into program file FIREUP as elements with element name A (version name of spaces).

8. Magnetic tape file TAP1 is copied onto magnetic tape TAP2. Two EOF marks are written on TAP2 and the tape is positioned between the two. File TAP1 may contain abnormal frame counts.

9. The contents of file F1 are copiedontofile F2. File F1 must be an SDF-formatted file.

8.2.2. COPYING FROM TAPE TO PROGRAM FILES (@COPIN)

Purpose:

Used to copy one or more elements from an element file located on magnetic tape into a program file located on FASTRAND-formatted mass storage.

All parameters of the @COPIN control statement are optional except name-1.

Format:

```
@label:COPIN,options   name-1,name-2
```

Parameters:

options — See Table 8–4 for file options and Table 8–5 for element options. See 8.1.1 for additional information on the A, C, R, and S options.

name-1 — Specifies the input element file or input element to be copied.

name-2 — Specifies the output program file, or output program file and element name. If name-1 is an element name and name-2 is a file name, the element will retain its name in the new file.

Description:

See 2.6.1 for additional information on specifying file and element names.

Procedure names are saved, but the entry points were discarded when the program file was converted to an element file with a @COPOUT control statement (see 8.2.3). When a relocatable element is added to a program file, any entry point table that may have existed for the file prior to the execution of the @COPIN control statement is destroyed. The @PREP control statement (see 8.2.11) may be used to recreate the entry point table. If a tape error occurs, only those elements transferred before the error occurred are entered in the program file's table of contents.

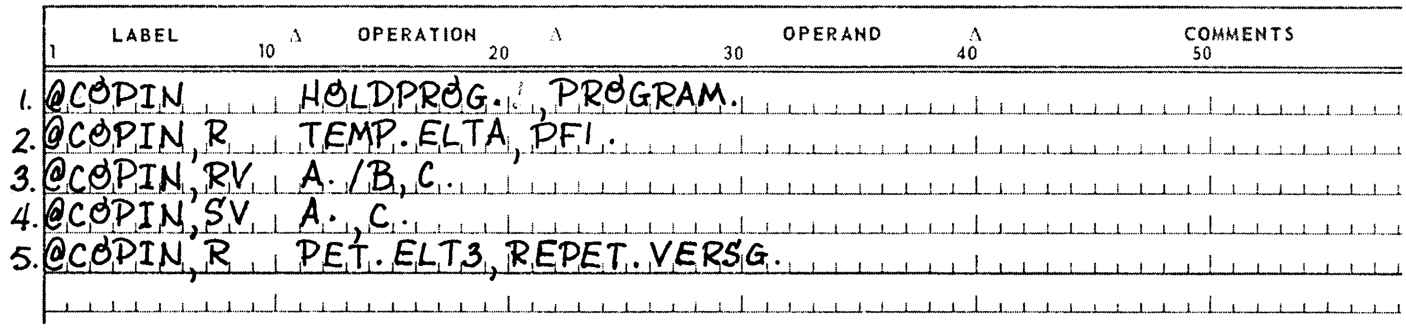
Option Character	Description
No option specified	Copies elements from the input magnetic tape file (in element file format) into the program file located on FASTRAND-formatted mass storage. The tape file must be positioned at the label block of the first element being copied (use @FIND control statement – see 8.2.13) and continues until a hardware EOF mark is encountered. The elements retain the element name they had in the element file.
A, R, S	Same operation as if no options were specified except that the A, R, and S options can be used to designate the type of elements to be copied. Any combination of A, R, and S can be used. All elements of the types specified are transferred. The elements retain the names they had in the element file.
V	Same as for elements (see Table 8–5).

Table 8–4. @COPIN Control Statements, Options Applicable When Filenames Are Specified

Option Character	Description
A,R,S	One element is copied and inserted into the output program file. The input tape must be positioned (with the @FIND control statement – see 8.2.13) to the correct element to be copied. The option and element specified must match the type and element to which the tape is positioned. Only one option may be specified. The element name remains the same unless renamed in name-2.
V	Used to copy elements having the same version name from an element file on magnetic tape into a program file on FASTRAND-formatted mass storage. The input file must be positioned at the label block of the first element to be copied and copying continues until a hardware EOF mark is encountered. name-1 – Specifies an input file, or input file and element version name. name-2 – Specifies an output file, or output file and element version name. When the version name is omitted from name-1, only those elements having a blank version name are copied into the output file. When the version is omitted from name-2, the elements retain the version names they had in the input file. The V option may be user with the A, R, and S options to select particular types of elements within version names for copying.

Table 8–5. @COPIN Control Statement, Options Applicable When Element Names Are Specified

Examples:


LABEL
OPERATION
OPERAND
COMMENTS
1
10
20
30
40
50

```
1. @COPIN     HOLDPROG.,PROGRAM.
2. @COPIN,R   TEMP.ELTA,PF1.
3. @COPIN,RV  A./B,C.
4. @COPIN,SV  A.,C.
5. @COPIN,R   PET.ELT3,REPET.VERSG.
```

1. Element file HOLDPROG located on magnetic tape is copied and reformatted into program file format and added to program file PROGRAM on FASTRAND-formatted mass storage.

2. Relocatable element ELTA in element file TEMP is copied into program file PF1. The tape must be prepositioned to the label block of relocatable element ELTA. The entry point table of file PF1 is not updated (a @PREP control statement is needed to update the entry point table – see 8.2.11).

3. All relocatable elements in element file A with the version name B are copied into program file C. They retain the version name B in the program file.

4. All symbolic elements in element file A with a blank version name are copied into program file C. These elements added to the program file have a blank version name.

5. The relocatable element in element file PET with the element name ELT3 and a blank version name is copied into program file REPET and given the element name VERS6 with a blank version name. The tape must be prepositioned to the label block of the relocatable element ELT3 with a blank version name (use a @FIND control statement — see 8.2.13).

8.2.3. COPYING PROGRAM FILES TO TAPE (@COPOUT)

Purpose:

Copies a program file, or selected elements from a program file, located on FASTRAND-formatted mass storage into a magnetic tape file in element file format.

All parameters of the @COPOUT control statement are optional except name-2.

Format:

```
@label:COPOUT,options   name-1,name-2
```

Parameters:

options — See Table 8–6 for file options and Table 8–7 for element options. See 8.1.1 for additional information on the A, C, R, and S options.

name-1 — Specifies the input program file or element to be copied

name-2 — Specifies the output element file, or output element file and element/version names.

Description:

See 8.1.1 for additional information on specifying file names.

Procedure name entries are saved but entry points are discarded. Tape files must be in element file format in order to use the @FIND and @COPIN control statements (see 8.2.13 and 8.2.2, respectively).

If either the A, R, S, or V option is specified the @COPOUT control statements, an EOF mark is not written automatically and a final @MARK control statement (see 8.2.9) may, therefore, be necessary.

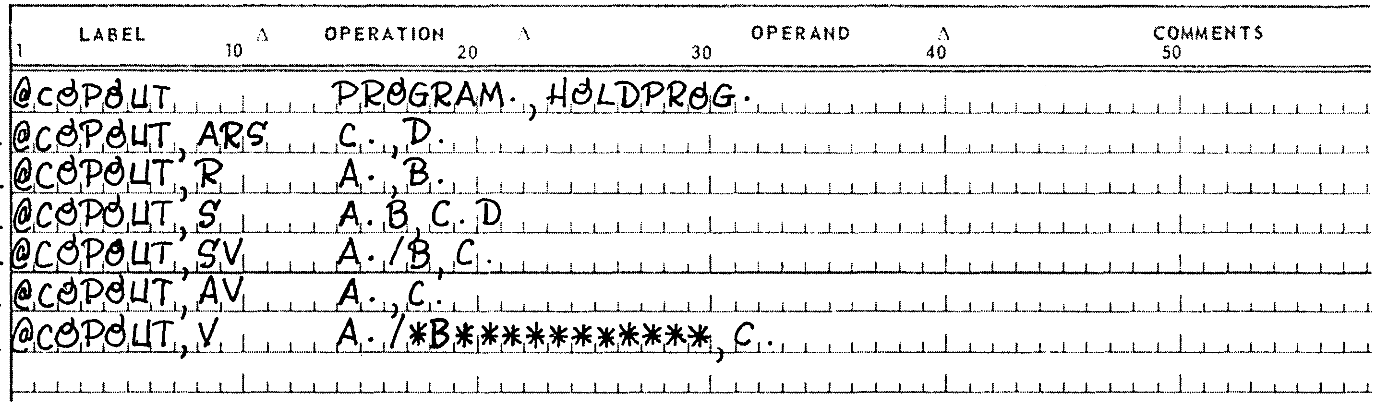
Option Character	Description
No option specified	All nondeleted elements are written onto the magnetic tape output file in element file format. Two EOF marks are written at the end of the file and the tape is backspaced one EOF mark. Elements retain the element name they had in the program file.
A, R, S	All nondeleted elements of the types specified by the options are written onto the magnetic tape output file in element file format. The elements retain the names they had in the program file. Any combination of A, R, and S can be used.
V	Same as for elements (see Table 8–7).

Table 8–6. @COPOUT Control Statements, Options Applicable When Filenames Are Specified

Option Character	Description
A, R, S	All specified element types for the element names given are written into the output magnetic tape file in element file format. Only nondeleted elements are transferred. If the name-2 element name is different than the name-1 element name, all elements copied have the new name. One or more options must be specified. No EOF mark is written.
V	All nondeleted elements, selected by version name and type, are written onto the magnetic tape output file in element file format. The V option may be used in combination with the A, R, and S option. When it is used alone, all element types are considered. name-1 – Specifies an input file, or an input file and element version name. File 1 must be on FASTRAND-formatted mass storage and be in program file format. name-2 – Specifies an output file, or an output file and element version name. If the version name is omitted from name-1, only those elements with a blank version name are considered for copying into the output file. When a version name is given in name-2, it replaces the original version name. When the version name is omitted from name-2, the elements written into the output file retain the names they had in the input file. Version Mask – An * in the version name in name-1 causes the character in the corresponding position in the version names of the elements in the input file to be ignored. For example: TAB./**D********* in name-1 would write all nondeleted elements in file TAB with a D as the third character in their version name into the output file. The V option must be specified when the version mask is used.

Table 8–7. @COPOUT Control Statement, Options Applicable When Element Names Are Specified

Example:


LABEL
OPERATION
OPERAND
COMMENTS
1
10
20
30
40
50

```
1. @COPOUT           PROGRAM.,HOLDPROG.
2. @COPOUT,ARS       C.,D.
3. @COPOUT,R         A.,B.
4. @COPOUT,S         A.B,C.D
5. @COPOUT,SV        A./B,C.
6. @COPOUT,AV        A.,C.
7. @COPOUT,V         A./*B**********,C.
```

1. The contents of program file PROGRAM located on FASTRAND-formatted mass storage are copied into magnetic tape file HOLDPROG and reformatted as an element file. Since no options are specified, two EOF marks are written following the output file, and the tape is positioned between the EOF marks.

2. The nondeleted elements of program file C located on FASTRAND-formatted mass storage are copied into magnetic tape file D and reformatted as an element file. No EOF marks are written following the file (option having been specified).

3. All nondeleted relocatable elements in program file A located on FASTRAND-formatted mass storage are copied into magnetic tape file B and reformatted as an element file. No EOF marks are written following the file.

4. Symbolic element B in program file A is copied into magnetic tape file C in element file format and given element name D.

5. All nondeleted symbolic elements in program file A with a version name of B are copied into magnetic tape file C and retain the version name B. File C is in element file format.

6. All nondeleted absolute elements in program file A with a blank version name are copied into magnetic tape file C. File C is in element file format.

7. All nondeleted elements in program file A with a version name containing a B as the second character are copied to magnetic tape file C. The version names are unchanged.

8.2.4. POSITIONING TAPE FILES (@MOVE)

Purpose:

Moves a magnetic tape file forward or backwards over a specified number of EOF marks.

All parameters in the @MOVE control statement are *required* except label and options.

Format:

```
label:MOVE,options    filename,n
```

Parameters:

options — The B option is the only valid option. If specified, tape movement is backward; if omitted tape movement is forward.

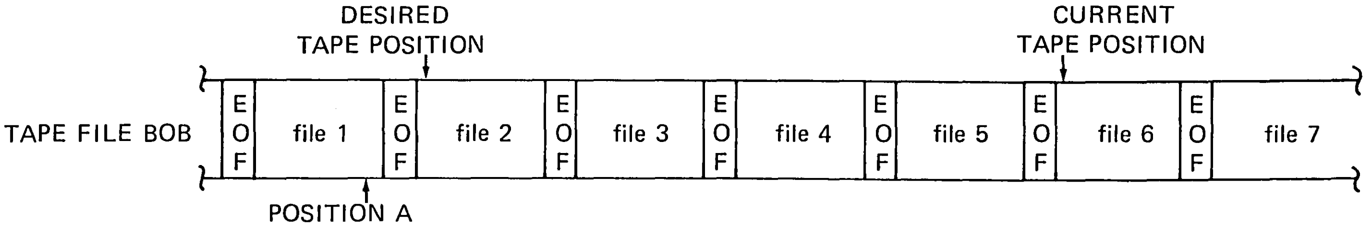
filename — Specifies the name of the tape file.

n — Specifies the number of EOF marks to be skipped.

Description:

Tape movement in the forward direction leaves the tape positioned at the start of the file on a multifile reel.

Care must be exercised when moving tape in the backward direction. Assume that tape file BOB is positioned at file 6:



To position the tape to the start of file 2, the following sequence must be executed:

```
@MOVE,B   BOB,5
@MOVE    BOB,1
```

Step 1 moves to tape to position A, and step 2 moves the tape to the start of file 2.

If a @MOVE,B control statement for a multireel tape file encounters the load point of the file it is currently on, a diagnostic message is given and the ERR$ exit is taken.

8.2.5. LISTING FILES AND MASTER FILE DIRECTORY (@PRT)

Purpose:

Obtains a listing of the text of a symbolic element, the table of contents of a program file, or the master file directory items (see Section 22) of catalogued files. The control statement does not list absolute or relocatable elements, as this may be done by the LIST processor (see 18.6).

All parameters in the @PRT control statement are optional.

Format:

```
@label:PRT,options   name-1,name-2...,name-n
```

Parameters:

options

See Table 8–8 for options (applicable to files, project-ids, and account numbers) and Table 8–9 for element options.

names

Specifies any of the following:

- the name of a catalogued file in any format
- the name of a program file
- the name of a symbolic element
- the name of an account number
- the name of a project-id

Option Character	Description
No option specified	When no parameters are specified, the entire master file directory is listed. The read/write keys are not included in the listing except when the SYS$*DLOC$ file (see 22.3) is assigned. The items are listed first by project-id, then by account number, and then by qualifier and filename.
F	List the information from the master file directory items for each catalogued file specified, subject to existing system security regulations. The read/write keys are not listed except when the SYS$*DLOC$ file (see 22.3) is assigned.
L	Used with the T option (in a demand run) to obtain a long rather than a short listing of a program file's table of contents. This option is meaningful only for demand runs.
N	List the information from the master file directory items for each account number specified, subject to existing system security regulations. The read/write keys are not listed except when the SYS$*DLOC$ file (see 22.3) is assigned. When no account number is specified, the entire master file directory is listed, first by account number, then by project-id, and then by qualifier and filename.
P	List information from the master file directory items for each project-id specified, subject to existing system security regulations. The read/write keys are not listed except when the SYS$*DLOC$ file (see 22.3) is assigned. When no project-id is specified, the entire master file directory is listed, first by project-id, then by account number, and then by qualifier and filename.
T	List the table of contents for each specified program file (located on FASTRAND-formatted mass storage).

Table 8–8. @PRT Control Statement, Options Applicable When Filenames, Account Numbers, or Project-ids Are Specified

Option Character	Description
No option specified	List the text of the specified symbolic elements.
S	Same as if no options were specified
T	List the table of contents for each element specified. All element types (A, S, R) with the specified element name are listed.

Table 8–9. @PRT Control Statement, Options Applicable When Elements Are Specified

Description:

When the @PRT control statement is used to obtain a listing of the master file directory, the read/write keys and the project-id's are replaced by slashes (//////), except for those files that have the same project-id as the run making the request. For more information, see 22.3.1.

The table of contents information output from the execution of a @PRT,T control statement contains heading information describing the contents of the table of contents. Some of the heading information is not self-explanatory. These are:

Element Table:

DELETE FLAG — An asterisk means entry deleted. No other symbol is used.

TYPE — If the element is symbolic, the processor which created the element is indicated.

DATE AND TIME — Time that element was created or, in some cases, when it was added to this file.

SEQUENCE NO. — The element sequence number is the position of the element in this file (this is sequentially issued) as elements are added to the file.

SIZE-PRE,TEXT — TEXT is the text size in sectors (a sector is 28 words). PRE is the preamble size in sectors (relocatable elements only).

CYCLE WORD — The first field is the maximum number of cycles (cycle limit) to be maintained for the unit (see 2.6.5). The second field is the most current cycle (absolute). The third field is the number of cycles currently being maintained.

PSRMODE — Blank if element does not contain third- or quarter-word mode operations.

QUARTR if element is quarter-word sensitive.

THIRD if element is third-word sensitive.

Location (Relative Sector No.) — Refers to the sector position of the start of the text.

Procedure Table (Assembler, COBOL, FORTRAN):

DELETE FLAG — An asterisk means entry deleted. No other symbol is used.

LOCATION — Refers to the word position relative to the start of the file.

LINK — The sequence number of the element that contains this procedure name.

Entry Point Table:

NAME — Name of externally defined symbol.

LINK — The sequence number of the element that contains this entry point.

The @PRT,TL control statement from a demand terminal results in the listing at the demand terminal of the table of contents in the format described above. When using the TL options; if an element name is given in addition to the filename, the table of contents is listed for the specified element only.

When the L option is omitted, the @PRT,T control statement from a demand terminal results in the following short-form table of contents format:

```
type   element-name/version(cycles)
```

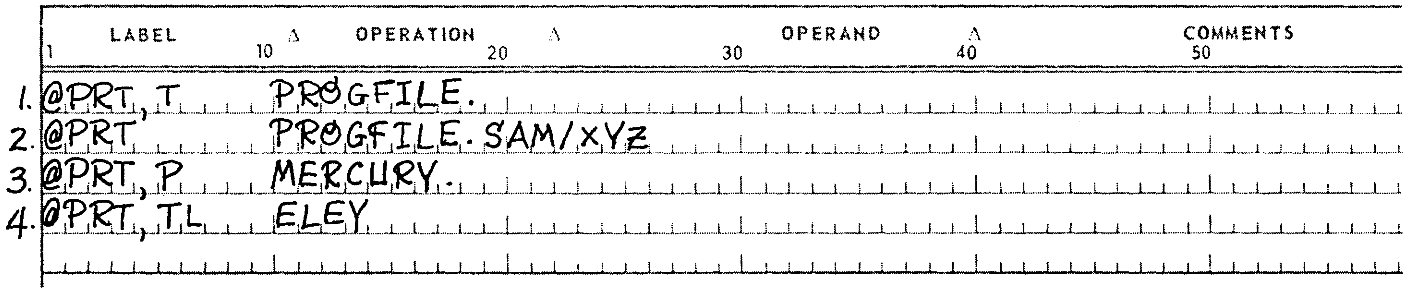
where:

type — Indicates the type of element. The element type codes are:

ABS	– Absolute element
ALG	– ALGOL symbolic element
ASM	– Assembler symbolic element
ASMP	– Assembler procedure
COB	– COBOL symbolic element
COBP	– COBOL procedure
DOC	– DOC processor symbolic element
ELT	– ELT processor symbolic element
FOR	– FORTRAN symbolic element
FORP	– FORTRAN procedure
MAP	– Collector (MAP processor) symbolic element
REL	– Relocatable element
SYM	– Symbolic element of unspecified type

cycles — Indicates the number of element cycles accumulated

Examples:



```
   LABEL    Δ  OPERATION  Δ        OPERAND    Δ        COMMENTS
 1         10          20        30          40        50
1. @PRT,T     PROGFILE.
2. @PRT       PROGFILE.SAM/XYZ
3. @PRT,P     MERCURY.
4. @PRT,TL    ELEY
```

1. The table of contents for program file PROGFILE is listed following the format given in Description. The period must follow the filename; otherwise, the specified name is considered to be an element name in TPF$.

2. The most recent cycle of symbolic element SAM, version XYZ, in program file PROGFILE is listed.

3. Information from the master file directory items for all catalogued files whose project-id is MERCURY is listed subject to current system security restrictions. This information is completely labeled to prevent any ambiguities as to the meaning of any entry in the listing. The project-id MERCURY must be the same as the run's project-id. If not, no listing is generated.

4. A complete table of contents is to be given for element ELEY in TPF$.

8.2.6. EMPTYING A PROGRAM FILE (@ERS)

Purpose:

Resets the next write location to the first sector of the text area, clears the table of contents, and returns to the system all FASTRAND-formatted mass storage granules allocated to a program file.

All parameters in the @ERS control statement are optional.

Format:

```
@label:ERS   filename-1,filename-2,...,filename-n
```

Parameters:

filenames	Specifies the program files to empty.

8.2.7. DELETING FILES AND ELEMENTS (@DELETE)

Purpose:

Drops catalogued files or marks elements in program files as deleted.

All parameters in the @DELETE control statement are optional except name-1.

Format:

```
@label:DELETE,options   name-1,name-2,...,name-n
```

Parameters:

names — Specifies the catalogued file or the element to be deleted.

options — See 8.1.1 for additional information on the A, C, R, and S options.

- No Options Apply When Deleting a Catalogued File

  Each catalogued file specified is marked as dropped. The filenames specified may be external or internal.

  When an external filename is specified, the F-cycle must be specified if it is not the latest F-cycle. If the file has read/write keys and is to be assigned to the run by the FURPUR processor, the read/write keys must be specified. The keys may be omitted if the file was assigned prior to calling the FURPUR processor.

  If an internal filename is used, it must have been attached to an external filename by means of a @USE control statement (see 3.7.5).

  The file is not actually dropped until all other runs that have the file assigned to them have freed the file. When the file is dropped, the master file directory items are updated. The older F-cycles have their relative F-cycle increased.

  See 2.6.3 for a discussion of file cycles.

- Deleting Elements

  When the A, R, and S options are specified, an element of that type in a program file is marked deleted. Each entry in the operand fields names the element and the program file that contains it. Any combination of A, R, and S options may be used, but at least one must be specified.

  Each element specified in the @DELETE control statement must exist as a nondeleted element before the @DELETE control statement is encountered. Including a cycle number for a symbolic element is illegal; all cycles of the element must be deleted. All procedure names associated with the deleted element are marked as deleted and the entry point table is destroyed. A @PACK control statement (see 8.2.14) may be used to physically eliminate the deleted elements.

Description:

The effect of a @DELETE control statement on a catalogued file is the same as the sequence:

```
@ASG,AYQ      FILEA
@FREE,D       FILEA
```

Examples:

```
     LABEL        Δ    OPERATION     Δ              OPERAND        Δ                COMMENTS
1                10              20              30              40              50
1. @DELETE,S   F.ELT1/VERS,F1.ELTY
2. @DELETE     FLAD.,TARE5.,ZEBRA4.,BAKER
```

1. The symbolic elements ELT1/VERS in program file F and ELTY in program F1 are marked as deleted. Any associated procedure names are also marked as deleted.

2. Relative F-cycle -0 of catalogued files FLAP, TARE5, ZEBRA4, and BAKER is dropped from the master file directory (the files are decatalogued).

8.2.8. REWINDING TAPE FILES (@REWIND)

Purpose:

Rewinds magnetic tape files back to the load point of the first reel.

All parameters in the @REWIND control statement are optional except filename-1.

Format:

```
@label:REWIND,options   filename-1,filename-2,...,filename-n
```

Parameters:

options
: The C (see 8.1.1) and I options are the only valid options. If the I option is specified, the tape file is rewound with interlock; if omitted, the tape file is rewound without interlock.

filenames
: Specifies the tape files to be rewound.

8.2.9. MARKING AN EOF ON TAPE (@MARK)

Purpose:

Writes two hardware EOF marks on a magnetic file and leaves the tape positioned between them. Some FURPUR control statements do an automatic @MARK or can be made to do a @MARK by specifying the M option.

All parameters in the @MARK control statement are optional except filename-1.

Format:

```
@label:MARK    filename-1,filename-2,...,filename-n
```

Parameters:

filenames	Specifies the tape files on which hardware EOF marks are to be written.

8.2.10. CLOSING TAPE FILES (@CLOSE)

Purpose:

Writes two hardware end-of-file (EOF) marks on a magnetic tape file and then rewinds it.

All parameters in the @CLOSE control statement are optional except @, CLOSE, and filename-1.

Format:

```
@label:CLOSE,options    filename-1,filename-2,...,filename-n
```

Parameters:

options	The C (see 8.1.1) and I options are the only valid options. If the I option is specified, the tape is rewound with interlock; if omitted, the tape is rewound without interlock.
filenames	Specifies the tape files to be closed. The tapes are then rewound.

8.2.11. ENTRY POINT TABLE CREATION (@PREP)

Purpose:

Creates an entry point table from the preambles of the nondeleted elements of a program file.

All parameters in the @PREP control statement are optional.

Format:

```
@label:PREP    filename-1,filename-2,...,filename-n
```

Parameters:

filenames	Specifies the program files for which entry point tables are to be created.

Description:

If a previous entry point table existed, it is destroyed prior to creating the new one. Note that whenever a relocatable element is added to or deleted from a file, any existing entry point table is deleted.

8.2.12. PUNCHING PROGRAM FILE ELEMENTS (@PCH)

Purpose:

Punches program file elements into 80-column cards.

All parameters in the @PCH control statement are optional except eltname.

Format:

```
@label:PCH,options   eltname,seq-char
```

Parameters:

options — See 8.1.1 for additional information on the A, C, R and S options. The function of the options are as follows:

A, R,S – Specifies the type of element to be punched. Any combination of A, R, or S may be used, but at least one must be given.

The following options may be used only in conjunction with the S option:

G – Produce punched card output containing compressed symbolic images.

H – Punch sequence number into columns 73–80 of each image. Seq-char must contain an alphabetic sequence of from one to three characters. The characters are left-adjusted and overlay columns 73–75.

J – Compress input images and sequence output cards in columns 73–80.

The G and J options may not both be specified in the same control statement.

eltname — Specifies element to be punched.

seq-char — Specifies alphabetic sequencing characters when H option is selected.

Description:

See 2.6.4 for information on how to specify element names.

The FURPUR processor ensures that the elements punched contain the control cards needed to reinsert them into the same program file, or a program file with the same name in a later run. The first card of a procedure element is a @PDP,I control card (see 3.6) otherwise, it is an @ELT,I card. The filename on the control card is the name of the file from which the element was punched.

Relocatable and absolute elements are automatically (without special option) sequenced in columns 79–80. Sequencing starts with AA and ends with ZZ (starts over with AA if necessary). If the H option is specified, symbolic images are sequenced in columns 76–80. The card sequence will be 100 apart and is preceded by the designated alphabetic string given in the seq-char field on the @PCH control statement.

The punched output is normally preceded by a properly formatted @ELT,I card. The @ELT,I card produced by @PCH has the same element name as the @PCH control statement. Thus, if the file containing the element that was punched is assigned to a subsequent run, all that is necessary to reintroduce the element is to include the @PCH-produced cards in the run stream.

Examples:

```
   LABEL      Δ   OPERATION    Δ          OPERAND      Δ             COMMENTS
 1          10              20          30            40             50
1. @PCH,S      UPDATE.RUNPRØG
2. @PCH,SRJH  A.B,XYZ
```

1. Symbolic element RUNPROG in program file UPDATE is punched onto 80-column cards, one image per card.

2. Symbolic element B of program file A is punched on 80-column cards. The input images are sequenced in columns 76–80. The identification sequence is punched in columns 73–75. The input images are compressed, and the output is sequenced in columns 73–80.

   Relocatable element B of program file A is also punched. The text has been previously sequenced, and the FURPUR processor sequences the preamble.

   See 9.4.3 for a discussion of compressed symbolic elements.

8.2.13. POSITIONING WITHIN ELEMENT FILES (@FIND)

Purpose:

Locates an element in a magnetic tape file (file must be in element file format) and positions the tape immediately preceding the element's label block.

All parameters in the @FIND control statements are *required* except label and options.

Format:

```
@label:FIND,options   eltname
```

Parameters:

options	Only one of the options A, R, S may be used to specify the type of element. See 8.1.1 for additional information on these options.
eltname	Specifies the file and the element to be located.

Description:

The search is made in the forward direction until either the element is found or an EOF mark is encountered. When the EOF mark is encountered, the tape is backspaced to the previous EOF mark (or load point, whichever is encountered first) and the search is repeated. If no find is made, the error exit is taken when the EOF mark is encountered.

Normally, the @FIND control statement is used just prior to a @COPIN control statement (see 8.2.2) requesting that the element just located (or all elements up to the EOF mark) be inserted into a program file located on FASTRAND-formatted mass storage.

Care must be exercised when doing a @FIND operation on other than the first reel of a multireel file. If an EOF is encountered prior to locating the desired element the reel is backspaced only to the load point, not to the EOF which is located on a previous reel.

8.2.14. REMOVAL OF DELETED ELEMENTS (@PACK)

Purpose:

Rewrites an entire program file, removing all elements marked as deleted. @COPOUT and @COPY,P control statements (see 8.2.3 and 8.2.1, respectively) have the same effect on output files since they do not copy deleted elements.

All parameters of the @PACK control statement are optional.

Format:

```
@label:PACK    filename-1,filename-2,...,filename-n
```

Parameters:

filenames — Specifies the program files to be rewritten.

8.2.15. CHANGING FILE ELEMENT, AND VERSION NAMES, AND FILE KEYS AND MODES (@CHG)

The @CHG control statement is described in 8.2.15.1, which discusses how to change catalogued file names, keys, and modes; and in 8.2.15.2, which discusses how to change program file element and version names. Some examples of @CHG control statements are given in 8.2.15.3.

8.2.15.1. CHANGING CATALOGUED FILE NAMES, KEYS, AND MODES

Purpose:

Changes catalogued mass storage file names, keys and modes.

All parameters in the @CHG control statement are optional except name-1.

Format:

```
@label:CHG,options    name-1,name-2
```

Parameters:

options — The options are:

P – Set public mode

Q – Set private mode

V – Set read-only mode, clear write-only mode

W – Set write-only mode, clear read-only mode

Z – Clear read only and/or write-only modes (must not be used in conjunction with V and W options).

name-1 — Specifies the file to be changed

name-2 — Specifies the new name of the file.

Description:

The contents of name-2 when different from the corresponding field in name-1 determine what other functions are to be performed. A blank field indicates no change of that field in name-1.

One F-cycle series exists for each set of files with the same file name. Each catalogued file belongs to only one F-cycle series. Read/write keys, if any, are the same for all members of the series. The master file directory contains a lead item for each F-cycle series that lists the read/write keys for the series and points to a main item for each member of the series. The read-only mode and write-only mode indicators are kept in the main item for that member.

The @CHG control statement may be used to perform the following functions related to catalogued files:

(1) Change the read/write keys for all files of a given F-cycle series.

(2) Remove or set read-only or write-only modes on a file.

(3) Remove a file from an F-cycle series and add it to another series as the latest F-cycle.

(4) Set public or private mode on a file.

If an F-cycle series contains only one member, (1) is equivalent to changing the keys for a file, and (3) is equivalent to changing the name of a file.

Although the functions performed by the @CHG control statement do not include reading or writing in text areas of the files named, read/write keys, if the files have any, are required in order for @CHG to modify their master file directory items. This means that the filename on the first @ASG control statement given to the executive must include the keys if an exeternal name is used. If an internal name is used, it must be associated by a @USE control statement still in effect that includes the keys. FURPUR performs the initial assignment, if the user has not assigned the file. In this case, the same rules apply to the name furnished on the @CHG control statement as for the @ASG control statement furnished by the user.

8.2.15.2. CHANGING PROGRAM FILE ELEMENT AND VERSION NAMES

Purpose:

Changes program file element and version names.

All parameters in the @CHG control statement are *required* except label and options.

Format:

```
    @label:CHG,options    eltname-1,eltname-2
```

Parameters:

options — The A, C, R, and S options are the only valid options for this statement (see 8.1.1).

eltname-1 — Specifies the program file element.

eltname-2 — Specifies the same program file and the new element and version names.

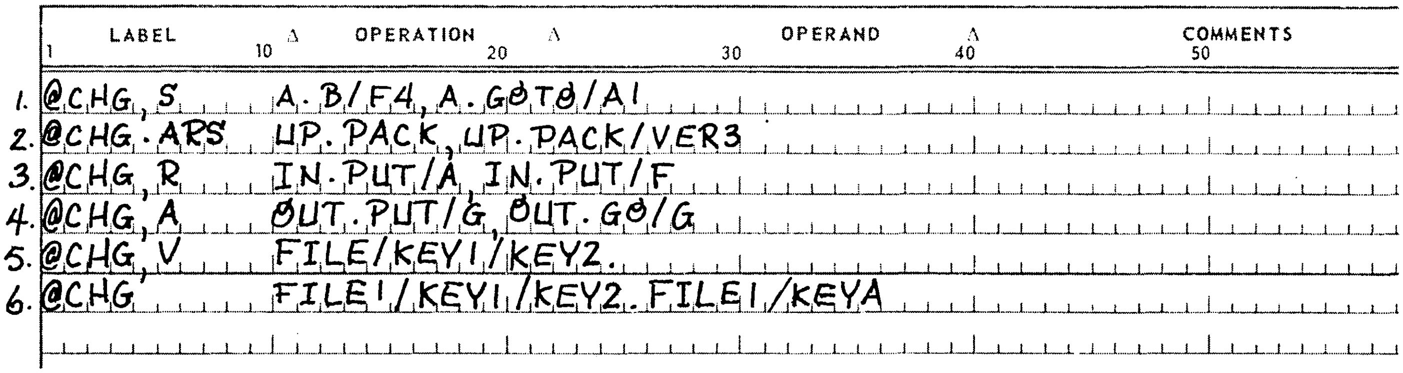
Description:

One or more of the A, R, and S options must be specified; only the elements of types specified by options will have their names changed. Element cycles may not be specified.

This operation can also be performed during a @COPIN (see 8.2.2), @COPOUT (see 8.2.3), or a @COPY,SRA (see 8.2.1) control statement.

8.2.15.3. EXAMPLES

The following six examples illustrate the operation of the @CHG control statement.



```
   LABEL      Δ  OPERATION   Δ         OPERAND    Δ      COMMENTS
1          10            20          30           40          50
1. @CHG,S      A.B/F4,A.GØTØ/A1
2. @CHG.ARS    UP.PACK,UP.PACK/VER3
3. @CHG,R      IN.PUT/A,IN.PUT/F
4. @CHG,A      ØUT.PUT/G,ØUT.GØ/G
5. @CHG,V      FILE/KEY1/KEY2.
6. @CHG        FILE1/KEY1/KEY2.FILE1/KEYA
```

1. Changes the element and version names of symbolic element B, version F4 of program file A to element name GOTO, version A1.

2. Assigns the version name VER3 to the absolute, relocatable, and symbolic elements named PACK in program file UP.

3. Changes the version name of relocatable element PUT in program file IN from A to F.

4. Changes the element name of the absolute element PUT in program file OUT to GO. The version name is not altered.

5. Changes the mode of catalogued file FILE from its present mode to read-only mode.

6. Change the read key of catalogued file FILE1 from KEY1 to KEYA and delete the write key.

8.2.16. ALTERING CYCLE RETENTION LIMIT (@CYCLE)

Purpose:

Sets the maximum range of absolute F-cycle numbers to be retained for a catalogued file (see 2.6.3) or the maximum number of element cycles for a program file symbolic element (see 2.6.5).

All parameters of the @CYCLE control statement are *required* except label.

Format:

@label:CYCLE name,n

FILES

Parameters:

name — Specifies a catalogued file whose cycle limit is to be changed.

n — Specifies the maximum range of F-cycles to be retained.

Description:

When a catalogued file is specified, the @CYCLE control statement sets the maximum number of F-cycle numbers. The filename specified must be in the master file directory. If n is 0, the file and all its cycles are marked as dropped. If n specifies a new maximum less than the current number of F-cycles being retained, the drop flag is set in enough F-cycles of the file to satisfy the new range (starting with the oldest cycle).

ELEMENTS

Parameters:

name — Specifies a program file symbolic element whose cycle limit is to be changed.

n — Specifies the maximum number of element cycles to be retained.

Description:

When a program file element is named, the @CYCLE control statement sets the maximum number of element cycles. If n specifies a new maximum less than the current number of element cycles being retained, a new element with the same name is created with as many of the oldest cycles marked deleted as needed to satisfy the new lower maximum.

Examples:

```
     LABEL        Λ   OPERATION    Λ                 OPERAND        Λ               COMMENTS
 1            10                20                30                40                50
1. @CYCLE     Q*A.B,2
2. @CYCLE     Q*D.,2
```

1. Assume that symbolic element B in program file Q*A consists of element cycles 5, 6, 7, and 8. Since the new limit is two cycles, a new element B is created consisting of cycles 7 and 8.

2. Assume that the master file directory entry for file Q*D indicates that four absolute F-cycles 18, 15, 14 and 12 of the file exist. Since the new limit is two, the drop flag for absolute F-cycles 15, 14, and 12 is set. The limit is considered to be the range starting from the highest current absolute F-cycle number.

8.2.17. ENABLING FILES DISABLED DUE TO MALFUNCTIONS (@ENABLE)

Purpose:

Resets (removes) the disable flag for catalogued files.

All parameters in the @ENABLE control statement are optional except filename-1.

Format:

```
    @label:ENABLE    filename-1,filename-2,...,filename-n
```

Parameters:

filenames	Specifies the catalogued files to be reactivated.

Description:

If the specified file is not disabled, a message to this affect is printed on the listing (normal exit is taken).

9. LANGUAGE PROCESSORS AND LIBRARIES

9.1. INTRODUCTION

This section describes the processor control statement and how it is used. Information is also given concerning the system library files (LIB$ and RLIB$) and the user's temporary program file (TPF$).

9.2. OPERATING SYSTEM LIBRARY FILES (LIB$, RLIB$)

Two separate library files are available during the processing of a user run: the absolute library file (SYS$*LIBR$) and the relocatable library file (SYS$*RLIB$).

The absolute library file (LIB$) contains the absolute elements of each standard processor included in the operating system. LIB$ may contain processors such as COBOL, PDP, FURPUR, and any other processor and executable program added by the installation.

The relocatable library file (RLIB$) contains the system-supplied relocatable elements and procedure elements which may be needed to assemble, compile, or collect the user program.

9.3. TEMPORARY PROGRAM FILE (TPF$)

A temporary program file (TPF$) is created by the executive for each run that is initiated. The qualifier for the filename is taken from the project-id field of the @RUN control statement. The file may be used as a scratch file for the user program's symbolic, relocatable, and absolute elements.

9.4. PROCESSOR CONTROL STATEMENTS

Processors form a special class of absolute elements which provide standard services for the user. The principal distinction is in regard to the manner in which the absolute element is invoked and the means by which information is made available to the processor. The general format of the processor control statement is:

```
@label:processor,option   param-1,param-2,param-3,...,param-n
```

The label field is as described in 3.2.1. The processor field is the name and file location (see 3.2.2) of the absolute element desired. The following is an example of a generalized processor control statement where the processor is located in a user-specified file rather than in the system library file LIB$:

```
   LABEL        ∧     OPERATION      ∧              OPERAND      ∧               COMMENTS
1          10                  20                  30          40                  50
1. @USER*FILE.PROG/ABS,P      FILE.IN, ELTOUT, FILE.OUT
```

The rules for locating the element in the processor field are slightly different from the standard rules for locating an element specified on an @XQT control statement (see 10.3.1):

(1) If a filename is specified, then that file is searched for the absolute element.

(2) If a filename is not specified but there is a leading period, then TPF$ is searched for the element.

(3) If a filename is not specified and there is no leading period, then the system library file SYS$*LIB$ is searched for the element; if there is no find, then TPF$ is searched. .The acronums for the standard processors (COB for COBOL, FOR for FORTRAN, and so forth) are the names of the respective absolute elements.

With one exception (the D option on the @ELT control statement – see 18.2) the option field has meaning only for the particular processor, though there are some options that have the same meaning for all processors. The format of the options parameter is described in 3.2.2.

The param-1, param-2,...param-n parameters contain information supplied to the processor. With the exception of the DATA processor (see 18.3), which works only with files and, therefore assumes filenames, the parameter fields are assumed to be in element name form although they need not represent element names. The meaning of the parameter fields is determined by the processor although the following rules are followed by the processors supplied by UNIVAC:

(1) If a field intended to contain the name of a program file is not specified, TPF$ is assumed.

(2) If a field is to contain an element name and the element name is specified but not the filename and there is no leading period, TPF$ is assumed. If there is a leading period, then the filename is taken from previous field provided that the field exists and was intended to name an element or a program file.

9.4.1. LANGUAGE PROCESSOR CONTROL STATEMENTS

The source language processors (ASM, COB, FOR, ALG, and so forth) have a common interpretation of several options as well as the first three parameters. The typical standard language processor control statement takes the form

```
    LABEL      Δ    OPERATION     Δ          OPERAND      Δ             COMMENTS
1             10                 20         30            40            50
@CØB     SI,RØ,SØ
```

where SI, RO, and SO represent eltname-1, eltname-2, and eltname-3.

The meanings of these parameters are:

SI (Source Input) — If a new symbolic element is being introduced from the run stream, this parameter specifies the file into which the new element is placed and the name which it is given. If an update is being performed, then this parameter specifies the element and the cycle of the element being updated.

RO (Relocatable Output) — This parameter specifies the name and the program file into which the element produced by the processor is placed. There is no restriction on the type of element being produced. For example, most of the processors produce relocatable binary elements; the collector produces either absolute or relocatable binary elements.

SO (Source Output) — This parameter specifies the name and the file for the updated symbolic element produced.

The source input routine options, I and U, assist in specifying the type of processing to be performed (see Table 9–3).

If no element name is specified or the parameter is left blank the following rules apply:

(1) If there is no file information and the parameter does not have a preceding period or if the parameter is void, then the file specified in the SI parameter is assumed.

(2) The element name from the SI parameter is assumed.

(3) If there is no version specified, then the version from the SI parameter is used.

Tables 9–1 and 9–2 describe the valid possibilities. The I and U options along with the SI parameters determine the interpretation of processor control statement. An error message is printed if there is any deviation from these rules. Table 9–1 is valid for the ASM, COB, FOR, ALG, and CFOR language processors (require SI, RO, and SO); Table 9–2 is valid for the PDP and LIST processors (requires only SI and SO).

I or U Option	SI Type	Element Type Produced	SI Notes	RO Notes	SO Notes
Neither	Not specified	New element	None	This parameter may or may not be specified. If is not specified, NAME$ is assumed. It is invalid to specify a cycle.	Illegal to use this parameter
Neither	Specified	Update	This parameter must be completely specified.	If this parameter is not completly specified, then the information from the SI parameter is used. It is invalid to specify a cycle.	If void, no source output is produced. If this parameter is not completely specified then information from the SI parameter is used. It is valid to specify a cycle.
I Only	Not specified	New element	None	This parameter must be completely specified but without a cycle.	Illegal to use this parameter.
I Only	Specified	New element	This parameter must be completely specified but without a cycle.	If not completely specified, the information from SI parameter is used. It is invalid to specify a cycle.	Illegal to use this parameter.
U Only	Specified	Update with cycling	This parameter must be completely specified.	If this parameter is not completely specified, then the information from the SI parameter is used. It is invalid to specify a cycle.	If this parameter is not completely specified, then the information from the SI parameter is used. It is invalid to specify a cycle.

Table 9–1. Processors Which Use the SI, SO, and RO Parameters

I or U Option	SI Type	Element Type Produced	SI Notes	SO Notes
Neither	Not specified	New element	An L option is assumed.	Illegal to use this parameter.
Neither	Specified	Update (no cycling)	This parameter must be completely specified	If this parameter is void, no source output is produced. It it is not completely specified, then the information from the SI parameter is used. Invalid to specify a cycle.
I only	Not specified	New element	None	Illegal to use this parameter.
I only	Specified	New element	This parameter must be completely specified but without the cycle.	Illegal to use this parameter.
U only	Specified	Update (with cycling)	This parameter must be completely specified.	If this parameter is not completely specified, then the information from the SI parameter is used. It is invalid to specify a cycle.

Table 9–2. Processors Which Require the SI and SO Parameters

9.4.2. SOURCE INPUT ROUTINE CONTROL OPTIONS

Table 9–3 contains a list of those options used by the source input routine (SIR – see 9.6) to control the input and output of the source language elements. Most UNIVAC supplied language processors (FOR, ASM, COB, ALG, and so forth) use the source input routine to obtain their input, therefore, the listed options are generally applicable to language processors.

Option Character	Description
G	Input is compressed symbolic in columns 1–80 of the card deck.
H	Input contains sequence numbers in columns 73–80 of the symbolic images.
I	Insert a new symbolic element into the program file.
J	Input contains compressed symbolic images in columns 1–72 of the cards and sequence numbers in columns 73–80. These sequence numbers are not checked by the K option.
K	Check sequence numbers in columns 73–80 of the symbolic images (valid only with H option).
P	Card image input, if any, is in Fieldata. Output symbolic element in Fieldata. (Compare with Q.)
Q	Output symbolic element in ASCII. Card image input, if any, is in ASCII. (If neither P nor Q is specified, code type of input element, if any, is used; otherwise, P is assumed.)
U	Update and produce a new cycle of the symbolic element.
W	List correction lines.

Table 9–3. Source Input Routine Options

9.4.3. COMPRESSED SYMBOLIC ELEMENTS

In order to minimize the number of cards required to contain a symbolic element, the FURPUR processor (see Section 8) can compress strings of blanks in symbolic images before punching the element. The source input routine can expand the compressed images on input.

A compressed symbolic card deck is produced when the appropriate options are used on the FURPUR @PCH control statement (see 8.2.12). The source input routine converts compressed card decks to SDF-formatted images (see 24.2.3) upon initial input when the appropriate options are used on the processor control statement (see 9.4.2).

The first card punched is an @ELT control statement (see 18.3) with the appropriate options. Following the @ELT control statement are the cards which contain the compressed symbolic images.

The compressed image consists of a stream of characters in the following format:

xccc...cyxccc...cz

where:

x Number of characters C ($1 \leqslant x \leqslant 37_8$)

y 40_8 + Number of blanks ($41_8 < y \leqslant 77_8$)

z 40_8 = End-of-image.

The number of characters in a string is limited to 37_8; the number of blanks is limited to 36_8. If either is larger, a new x or y is initiated.

In addition to the x, y, and z characters, two other special characters are used;

(1) 41_8 character — Indicates the end-of-images in this element.

(2) 0_8 character — A special character used in column 80 if a new x would begin in column 80. The x is moved to the next physical card and the 0_8 is placed in column 80.

The compressed images immediately follow one another on the physical card and continue to the next card when the end-of-card is reached. The punch routine begins each physical card with an x, y, or z by breaking an x string at the end of the card, and starting a new string on the next card. This guarantees a nonzero character in the first character position of the card. A compressed blank image would be represented by an 40_8 character. The physical card may contain compressed image characters in columns 1–80.

Compressed images are not retained in the program file. The source input routine expands the images, and stores them in the program file in SDF format.

9.5. MODIFYING SYMBOLIC ELEMENTS

9.5.1. LINE CORRECTION STATEMENT

On a symbolic listing, the successive lines of code are sequentially numbered. When altering the symbolic element, these numbers are used on the line correction statement to indicate where the correction lines are to be inserted. The format of the line correction statement is:

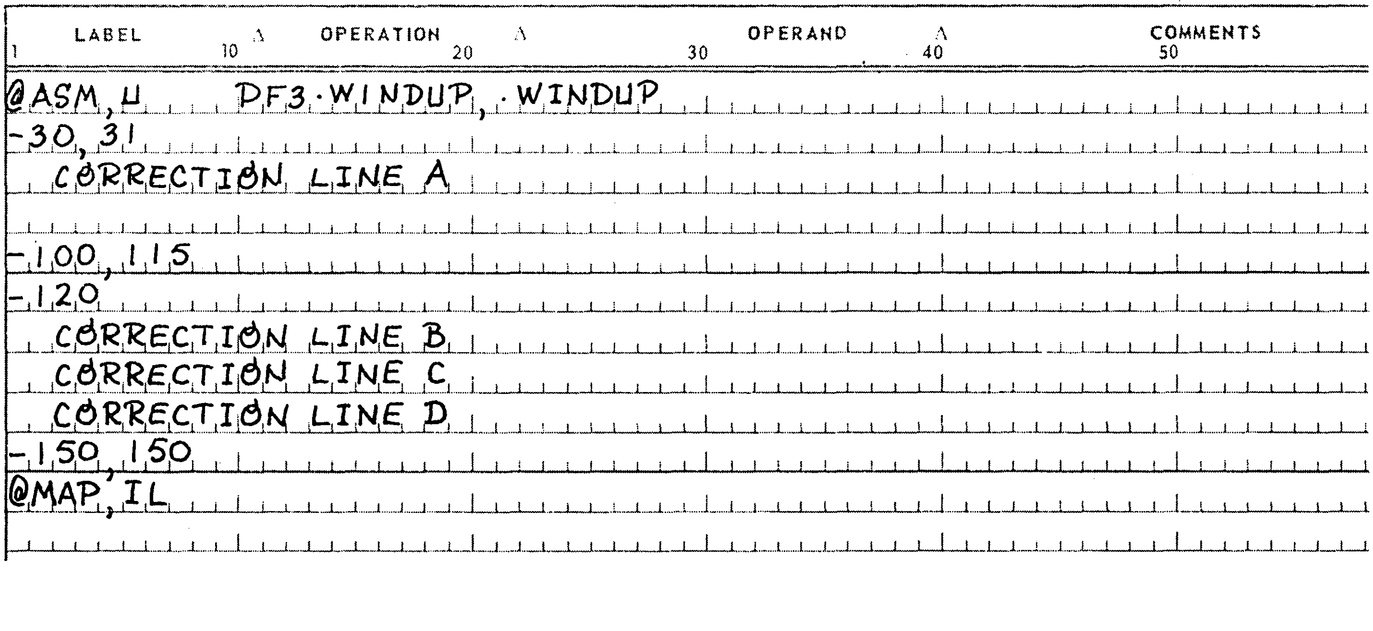
—n,m

The minus sign in column one is the correction indicator which specifies that symbolic code lines n through m are to be replaced by correction lines. The lines of code immediately following the —n,m construction are inserted until another correction indicator is read in column one or another correction statement is read. If no lines of code follow the —n,m line correction statement, lines n through m are deleted.

The construction —n with the correction indicator appearing in column one specifies that the succeeding lines of code are to be inserted in the symbolic element after line n.

If correction lines are to be inserted before the first line in the symbolic element, the correction lines are placed immediately after the processor control statement without specifying any insertion line numbers.

Examples:



```
   LABEL        OPERATION              OPERAND            COMMENTS
1          10            20          30          40          50
@ASM,U    PF3.WINDUP,.WINDUP
-30,31
  CORRECTION LINE A

-100,115
-120
  CORRECTION LINE B
  CORRECTION LINE C
  CORRECTION LINE D
-150,150
@MAP,IL
```

The U option on the @ASM control statement specifies that a new cycle of symbolic element WINDUP is to be produced. Lines 30 and 31 are replaced by correction line A. Lines 100 through 115 are to be deleted. Correction lines B, C, and D are inserted after line 120. Line 150 is deleted. Encountering the @MAP control statement indicates that there are no more correction lines to the symbolic element.

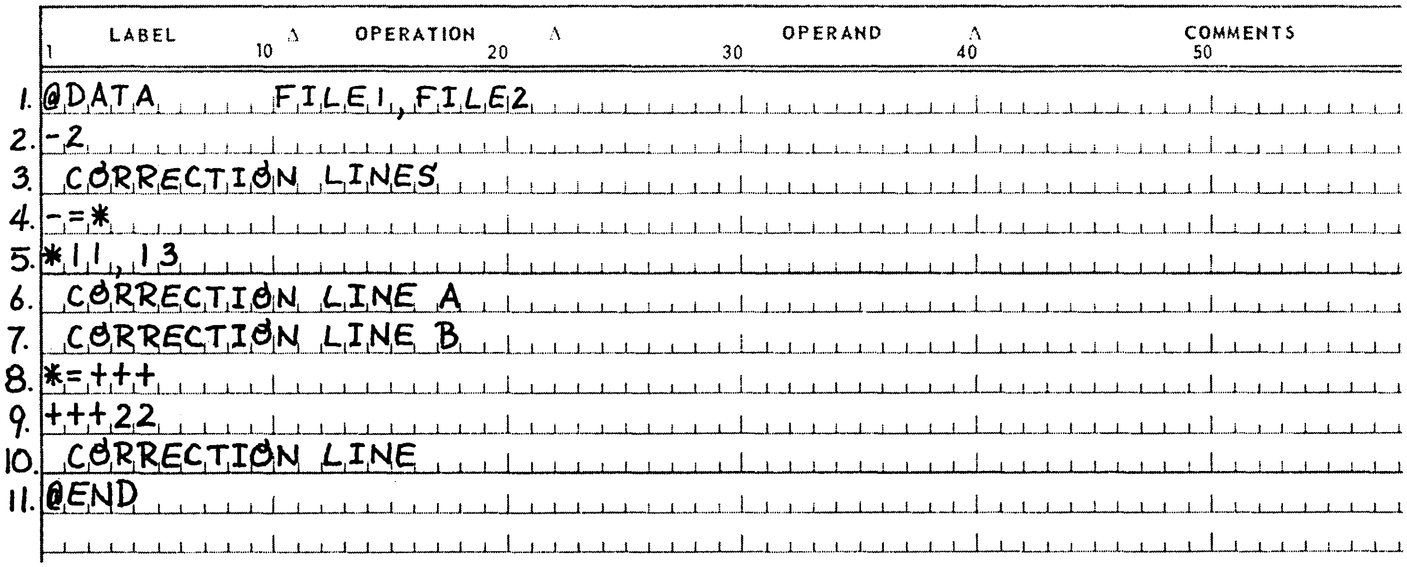
9.5.1.1. REDEFINITION OF THE CORRECTION INDICATOR

It is possible to redefine the correction indicator so that a symbol other than the minus sign may indicate the insertion of correction lines. Redefinition of the correction indicator makes it possible to insert correction lines that may be actual data preceded by a minus sign. The format for redefinition is:

```
–=x
```

x may be one to three special characters in length with no intervening blanks. The correction indicator may be redefined any number of times but only one symbol is recognized as the correction indicator at any time.

Examples:



```
    LABEL      Δ   OPERATION    Δ             OPERAND       Δ             COMMENTS
1          10            20              30            40             50
1.  @DATA     FILE1,FILE2
2.  -2
3.   CORRECTION LINES
4.  -=*
5.  *11,13
6.   CORRECTION LINE A
7.   CORRECTION LINE B
8.  *=+++
9.  +++22
10.  CORRECTION LINE
11. @END
```

Correction lines are to follow line 2 in the source program. Line 4 redefines the correction indicator to an asterisk (*). Lines 11, 12, and 13 are replaced with correction lines. Line 8 redefines the identifier to +++. Correction lines follow line 22 of the source program.

9.5.2. PARTIAL LINE CORRECTIONS

In addition to inserting entire symbolic correction lines, partial line corrections are also permitted. This is accomplished by using a range correction statement to define the number of code lines to be partially corrected followed by change correction statements which define the correction to be made.

In the formats given in 9.5.2.1 and 9.5.2.2 the slash (/) is used as a separator character. The separator character may be any character other than a digit, a comma if the change statement is type 2 or 3 (see 9.5.2.2), a blank, or a correction indicator. The first separator may be preceded by any number of blanks. The character chosen as a separator must not appear as a character in the old-data and new-data parameters of the change correction statement (see 9.5.2.2).

9:5.2.1. RANGE CORRECTION STATEMENT

The range correction statement format is:

```
–x,y–
```

The minus character immediately following the y is an edit indicator which must always be coded as a minus character. (The minus character that immediately precedes the x is a correction indicator that may be redefined by the user – see 9.5.1.1). The range correction statement must be followed by one or more change correction statements and there must be one change correction statement for each statement in the range –x,y–. For example, if the range correction statement is

–2,7–

then there must be six change correction statements immediately following the range correction statement. If the number of change correction statements following the range correction statement does not equal the number given on the range correction statement, a diagnostic is given.

The number specified in the x parameter must be greater than that given on any previous range, delete, or insert correction statement. The number specified in the y parameter must be equal to or greater than the x parameter.

9.5.2.2. CHANGE CORRECTION STATEMENTS

The change correction statements may be specified in any one of the following four formats.

Format 1: c/new-data

Format 2: c,d/new-data/

Format 3: /old-data/new-data

Format 4: /old-data/new-data/

Format 1 is used to replace the characters of an image from a specified column to the end of the image. The column number is specified in parameter c. Parameter new-data must contain the replacement characters. All of the data following the separator (except trailing blanks) is taken to be replacement characters.

Format 2 is used to replace a specified number of characters in an image. The column numbers entered in the c,d parameters specify the range of characters to be replaced. Parameter new-data contains the replacement characters. If the number of characters in the new-data parameter is greater than the range specified in the c,d parameters, then the characters following the column number specified in the d parameter are right shifted to make room; if it is less, the image is left shifted to close the image.

Format 3 operates similarly to format 1 except that the old-data parameter specifies one or more characters to be replaced. The coding line is scanned and when a find is made, the characters specified by the old-data parameter are replaced by the characters specified in the new-data parameter.

Format 4 operates similarly to format 2 except that the old-data parameter specifies one or more characters that are to be replaced by the characters specified in the new-data parameter.

Examples:

```
     LABEL         Δ      OPERATION      Δ              OPERAND        Δ              COMMENTS
1             10                  20                 30                  40                  50
@ASM,U      PF3.WINDUP,.WINDUP
-30,33-
73/SUBTØT6
42,48/SUBTØT7
/OVHEAD/CHARG7
/REFUND3/RETURND/
@MAP,IL
```

The U option on the @ASM control statement specifies an update to symbolic element WINDUP. Lines 30 through 33 are to be partially corrected. The characters in columns 73–80 in line 30 are replaced by SUBTOT6. The characters in columns 42–48 in line 31 are replaced by SUBTOT7. The characters OVHEAD in line 32 are replaced by CHARG7. The characters REFUND3 in line 33 are replaced by RETURND. Encountering the @MAP control statement indicates that there are no more corrections to the symbolic element.

9.5.2.3. PARTIAL LINE CORRECTION DIAGNOSTICS

When an error occurs, a diagnostic message is placed in the print file. The format of the diagnostic message is:

SIR EDIT ERR c r e

where:

c Indicates the cause of the error. Table 9–4 lists the possible errors.

r First four words of the range correction statement under whose control the error occurred.

e Specifies the change correction statement that caused the error.

c Contains the Words	Description
SEPARATOR	The separator used in the change correction statement is invalid or nonexistent.
COLUMN	The column number specified in a format 1 or 2 change correction statement is out of range; or c > d for a format 2 change correction statement.
NO FIND	The characters given in the old-data parameter of a format 3 or 4 change correction statement could not be found in the line being corrected. NOTE: Whenever one of the above errors occurs, the change correction statement is ignored and the line remains unchanged.
ASCII MODE	Indicates that symbolic input or output is in ASCII code, or that the user requested ASCII code. Since the source input routine cannot correct ASCII code, all range and change correction statements are ignored.
CARD COUNT <	Not enough change correction statements were provided. Those lines for which no change correction statement was provided remain unchanged.
CARD COUNT >	Too many change correction statements were provided. The excess change correction statements are ignored.

Table 9–4. Partial Coding Line Correction Diagnostics

9.6. PROCESSOR INTERFACE ROUTINES

A set of routines which are available in the system relocatable library file (SYS$*RLIB$) provides standard interfaces with the operating system for all language processors. These routines simplify the task of incorporating additional processors into the operating system.

In general, processors assign input and output files, obtain source and correction input, and generate source and relocatable output. When using the processor interface routines, the processor need only be concerned with requesting the next input image and outputting a relocatable word.

The various processor routines are described briefly in the following paragraphs. For a more detailed explanation see the *UNIVAC 1108 Operating System Technical Documentation.*

□ Preprocessor Routine

The preprocessor routine is designed to read the processor control statement, put it into standard format, dynamically assign the necessary files, obtain the options, obtain the next write location (for program files), and, if a tape is used, verify that the element read was the one called for.

The PREPRO routine is designed for use by processors which require source input, source output, and relocatable output parameters on the processor control statement. The PREPRM routine is designed for use by processor which require only source input and source output parameters.

□ Source Input Routine

The source input routine (SIR) is used by a processor to obtain the source language images from the run stream or from a symbolic element in a program or data file. The routine can automatically merge corrections, list the corrections, and produce an updated symbolic element. The symbolic element which contains the source input may be cycled; the desired cycle is specified in the processor control statement. The source input routine automatically passes to the processor only those images that pertain to the cycle requested.

□ Source Output Routine

Certain processors produce symbolic text as output rather than relocatable text. For these processors, the eltname-2 parameter of the processor control statement specifies the symbolic output element. The element produced by the symbolic output routine (DOR) is generated by the processor and may be distinctly different from the input or updated symbolic element.

The element produced (always cycle 0) is in SDF format and is entirely suitable for input to another processor by the source input routine.

□ Relocatable Output Routine

The relocatable output routine (ROR) is used to output into a program file a relocatable element that contains the relocatable text and preamble information produced by a processor in standard block and item sizes acceptable to the collector. The eltname-2 parameter of the processor control statement is used to specify the name of the relocatable element to be produced and the name of the program file into which it is placed.

□ Postprocessor Routine

The function of the postprocessor routine is to remove all changes in the assignment status of a program file which were made by a preprocessor routine. A processor working on a program file may require exclusive use of the file, and when finished, it should return the file to its original status. Because the preprocessor set up the file for the processor itself, the processor should call the postprocessing routine to restore the file to its original status.

□ INFOR Table Interface Routines

If a program is called for execution as a processor rather than by means of an @XQT control statement, its first READ$ request returns information from the operand fields of the processor control statement in a format called internal format (INFOR).

NOTE: When using the preprocessor routines, they must perform the first READ$ request and they must process the information from the processor control statement.

The INFOR table interface routine:

(1) Reads the INFOR table.

(2) Searches for a parameter subfield.

(3) Retrieves a complete parameter.

(4) Performs a dynamic @USE (see 3.7.5).

9.7. PROCEDURE DEFINITION PROCESSOR (PDP)

The procedure definition processor (PDP) accepts symbolic input defining assembler, FORTRAN, or COBOL procedures and builds an element in the user-defined program file. These procedures may subsequently be referenced in an assembly or compilation without definition.

One table is generated for each type of procedure in a program file. This table contains any labels that are defined externally to the procedure. For FORTRAN and COBOL procedures, these are labels on the procedure line. For assembler procedures, these are the labels defined externally (trailing asterisk after label) on the procedure and name lines of a first level procedure. For every label entered in a table, the location of the procedure or name line is noted. When a call is made for a procedure in a source program, the system automatically retrieves the procedure. If more than one procedure of the same type has the same label, an entry is made in the table for each procedure, but a call on that label will produce the last procedure entered for that label.

The PDP is called by the @PDP control statement.

All parameters in the @PDP control statement are optional except @, PDP, and eltname-1.

Format:

```
@label:PDP,options   eltname-1,eltname-2
```

Parameters:

options — See Table 9–5.

eltname-1 — Normally specifies the input element. However, when the I option is specified, eltname-1 specifies the new program file element.

eltname-2 — Specifies the output element.

Option Character	Description
A	Accept the results as correct even if errors are detected.
C	Indicates a COBOL procedure element.
F	Indicates a FORTRAN procedure element.
I	Insert a symbolic element into program file from the control stream.
L	Produce a complete listing of the output element with line numbers.
N	Suppress all listings.
S	Generate a single-spaced listing of the output element.
U	Generate a new cycle of the symbolic element.
W	List correction lines if corrections are provided.
X	Abort remainder of run if any errors are detected.

Note: *The source input routine options (see Table 4–3) also apply.*

Table 9–5. @PDP Control Statement, Options

Description:

When the F option is specified, PDP assumes that it is inserting or updating a FORTRAN procedure element. When the C option is specified, PDP assumes that it is inserting or updating a COBOL procedure element. When neither option is specified, PDP assumes that it is inserting or updating an assembler procedure element.

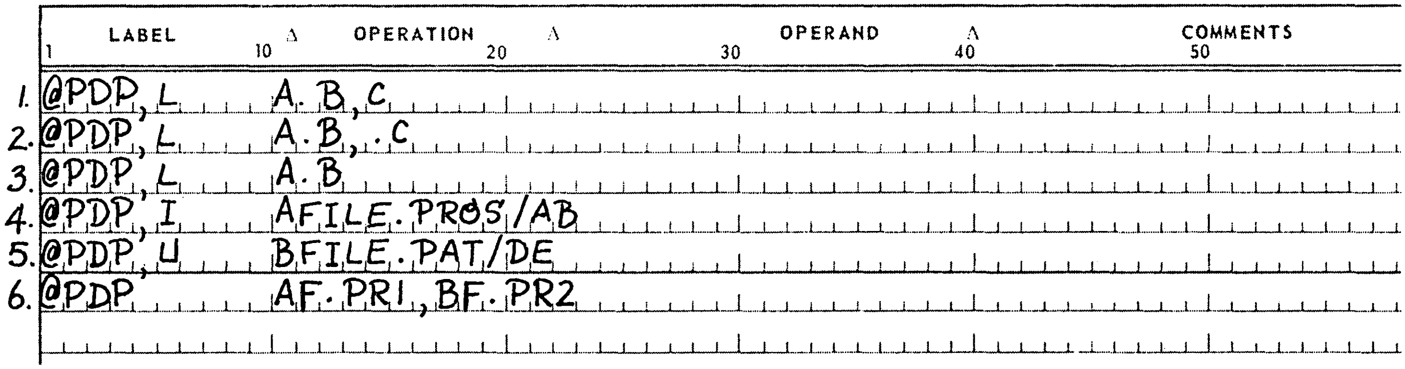
The I option is used only to introduce symbolic card images into a program file. When applying corrections to an element, the I option is not permitted.

PDP permits the processing of procedural elements from a tape file that is in element file format (see 24.2.2). Furthermore, corrections to this element are permitted if a symbolic element is produced in a program file. PDP does not attempt to interpret the names on a control statement, that is, it makes no effort to ensure uniqueness or avoid possible duplication of names in the eltname-1 and eltname-2 parameters.

Cycling of procedures is permitted. The cycle number may be increased if the U option is specified, only one cycle is retained, and when the procedure is collected, the latest cycle is supplied.

If PDP is processing elements from a tape file, the file must be positioned so that the label block is read in. If the name in the label block does not agree with eltname-1, PDP takes the error exit.

Examples:



```
    LABEL     △   OPERATION     △           OPERAND      △            COMMENTS
1         10                  20                 30                40              50
1. @PDP,L    A.B,C
2. @PDP,L    A.B,.C
3. @PDP,L    A.B
4. @PDP,I    AFILE.PRØS/AB
5. @PDP,U    BFILE.PAT/DE
6. @PDP      AF.PR1,BF.PR2
```

1. Generates a procedure element from file A, element B and places the new element in TPF$. Generates a complete listing.

2. Generates a procedure element from program file A, element B, calls it element C, and places it in program file A. Generates a complete listing. Eltname-2 must not name a tape file. PDP takes the error exit but does not abort (PDP will abort the run if the X option is specified).

3. Generates a complete listing of element B from file A.

4. Procedure definitions following this @PDP control statement are placed in file AFILE as element PROS, version AB, cycle 0.

5. Corrections are made to element PAT, version DE, latest cycle of file BFILE to generate an updated cycle of the same element in the same file.

6. Corrections following the @PDP control statement are merged with the most recent cycle of element PR1 in file AF to generate cycle 0 of element PR2 in file BF.

10. PROGRAM CONSTRUCTION AND EXECUTION

10.1. INTRODUCTION

The 1100 operating system provides the ability to combine the relocatable elements generated by a language processor into an executable (absolute) element. This combination or collection of relocatable elements is done by a system processor, the collector. The absolute element produced by the collector is structured such that the executive program loader can place it in execution. Once the absolute program has been created (that is, collected), it may be saved and reexecuted many times. The program need only be recollected when a modification to it is desired.

An absolute element (program) may be placed in execution through use of a program execution control statement (@XQT) within the control stream. When an @XQT control statement is encountered by the executive, the program is retrieved from its mass storage file, loaded into main storage, and executed. For a special class of programs (processors), the processor control statement initiates execution.

During exectuion, a program can determine which parts of the absolute elements are in main storage by requesting the executive to load previously-defined program overlay segments. In addition, the program has the capability of attaching to or linking to other previously defined absolute elements. The program has the ability during execution to dynamically determine the execution of other semi-independent absolute elements.

10.2. THE COLLECTOR

The collector is called by the @MAP control statement (See 10.2.1). It provides a straightforward means of collecting and interconnecting relocatable elements to produce a program in an executable form. This form is called an absolute element. Optionally, the collector can be used to create a single relocatable element from a collection of several relocatable elements. The three basic inputs to the collector are:

- The parameters supplied on the @MAP control statement
- The information supplied by the collector directives
- Relocatable elements taken from various sources, such as:
  - the temporary program file (TPF$)
  - user-created program files
  - the system's relocatable library (SYS$*RLIB$)

The three basic outputs of the collector are:

- An absolute or relocatable element
- A symbolic element
- A program listing

The primary output of the collector is the relocatable or absolute element which results from the collecting and linking of the various relocatable elements. This element is given a name and placed within a program file for subsequent use. Both the element name and the file in which the element is placed may be dictated by the user.

Usually the collector includes within an absolute element a set of tables for use by the diagnostic system. This output can be suppressed by the user (see Table 10-1).

10.2.1. COLLECTOR INITIATION (@MAP)

Purpose:

Specifies that the collector is to combine a set of relocatable elements into one absolute or relocatable element.

All parameters in the @MAP control statement are optional.

Format:

```
@label:MAP,options   eltname-1, eltname-2, eltname-3
```

Parameters:

options — See Table 10–1 and source input routine options (see Table 9–3).

eltname-1 — Specifies the input symbolic element which contains the source language (see 9.4.1).

eltname-2 — Specifies the absolute output element. When the R option is specified, eltname-2 names the relocatable output element (see 9.4.1).

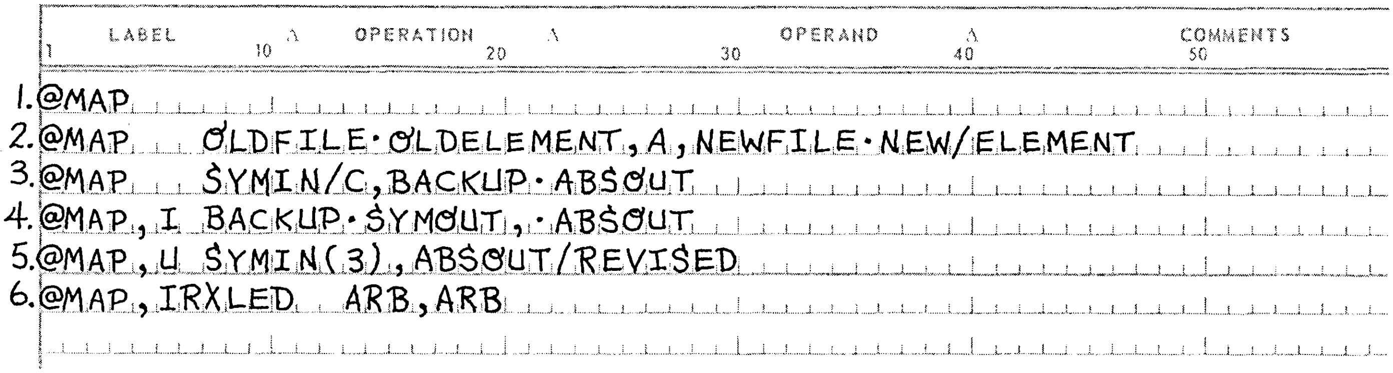
eltname-3 — Specifies the output source language element (see 9.4.1).

Option Character*	Description
A	Under no circumstance is the error exit (ERR$) to be taken during the collection, even if the collection is destroyed.
B	Mark the absolute element so that the program area is not cleared to zero prior to loading the program and any indirectly loaded segments (see 10.2.4.5.2).
D	Print a diagnostic message for all possible addresses over 65K (177777_8). Check for certain possible instruction format violations.
E	Allow program addresses to exceed 65K (177777_8). If this option is omitted and the program's D bank exceeds 65K, the D bank starting address is moved downward so that all (or as many as possible) of the over-65K addresses are forced below 65K.
F	Mark the output absolute or relocatable element as quarter-word sensitive.
L	Produce a complete listing which contains the following information concerning the program area: – main storage allocated to each element and segment – program address of all external definitions – the symbol '?' following any undefined entry point – the scale drawing of program segmentation – the external references of each element
N	Produce the most abbreviated print listing available.
R	Generate a relocatable element instead of an absolute element.
S	Produce a summary listing which includes a scale drawing of program segmentation. This option is assumed if neither the L nor N options are specified.
T	Do not mark the output element as quarter word sensitive. If neither the T nor F options are specified, the program sensitivity is determined as follows: – if only third-word sensitive elements are present, T is used – if only quarter-word sensitive elements are present, F is used – if both third- and quarter-word sensitive elements are present, the sensitivity of the element containing the program starting address is used.
V	Assign all addresses but strip off the D bank code (can be used to create a reentrant processor—see 10.2.3.4).
X	If an error is detected, terminate the collection and exit ERR$. When the X option is omitted, the results of the collection are accepted, even though there may be minor errors, as long as an absolute element is produced.
Y	Assign all addresses but strip off I bank code. Compare with V option.
Z	Suppress generation of diagnostic tables in the absolute element which are used by the PMD or other dump editors.

* *Also see source input routine (SIR) options, Table 9–3.*

Table 10–1. @MAP Control Statement, Options

Examples:



```
 LABEL      ∆   OPERATION   ∆          OPERAND      ∆          COMMENTS
1         10            20            30           40           50
1.@MAP
2.@MAP    OLDFILE·OLDELEMENT,A,NEWFILE·NEW/ELEMENT
3.@MAP    SYMIN/C,BACKUP·ABSOUT
4.@MAP,I  BACKUP·SYMOUT,·ABSOUT
5.@MAP,U  SYMIN(3),ABSOUT/REVISED
6.@MAP,IRXLED  ARB,ARB
```

1. This @MAP control statement produces the same results as the @MAP,I control statement. The names for the symbolic and absolute elements are automatically assigned by the collector. The printed output and internal table entries would appear as if the control statement had been: @MAP,I ,TPF$.NAME$. If no directives follow, the directive IN TPF$. is assumed.

2. Element OLDELEMENT is updated by any source language statements following the @MAP control statement. The output source language goes into file NEWFILE, element NEWELEMENT. The absolute element A goes into TPF$.

3. Version C (latest cycle) of element SYMIN from TPF$ is the input symbolic element. The absolute output element ABSOUT is written in file BACKUP. No source language output is produced; any successive correction lines are applied but not saved.

4. The source language statements (collector directives) following the @MAP control statement are inserted as the symbolic output element SYMOUT in file BACKUP. The absolute element ABSOUT is also put into file BACKUP.

5. Cycle 4 of element SYMIN located in TPF$ is produced. Any correction lines are saved. Version REVISED of absolute output element ABSOUT is also put in TPF$.

6. The source language statements following the control statement become the symbolic element ARB in the TPF$ file. The output element goes into TPF$ as relocatable element ARB. If errors are encountered during the collection, the run is terminated. A full listing is produced. D bank addresses are allowed to exceed 65K. Diagnostics are printed for addresses over 65K.

10.2.2. COLLECTOR DIRECTIVES

The collector directives enable the programmer to control the collection of his program. These directives:

- are free-form, hence, they may begin in any column of the source language image.
- may contain comments preceded by the blank-period-blank construction.
- follow the standard dropout rules (see 3.2.7) pertaining to filenames, element names, and so forth.

For the collection of complex programs which require relocatable input from many sources, construction of overlay segments, or the use of multiple libraries, the user must prepare a set of collector directives. These statements may follow the @MAP control statement or be contained in an element in a program file. The user has the same access and updating facilities for the (@MAP) symbolic element as for any other type of symbolic element.

10.2.2.1. ELEMENT INCLUSION (IN)

Purpose:

Allows the user to specifically include an element or an entire file in the collection of a program. An element may be preceded by a filename. The elements indicated on an IN directive are placed in the segment named by the preceding SEG directive (see 10.2.2.13).

All parameters in the IN directive are optional. If all parameters are omitted, IN TPF$ is assumed.

Format:

```
IN   name-1,name-2,name-3,...,name-n
```

Parameters:

name — Specifies the element or entire file to be included in the collection. See 2.6 for standard file and element notation.

Descriptions:

By starting 'filename' without a following element name, the user can specify the inclusion of all elements in a program. If some elements of a file have been explicitly named, these are included as specified and the 'IN filename' serves to bring in only the remaining elements in the file. When specifying an entire file for inclusion, a period must follow the filename.

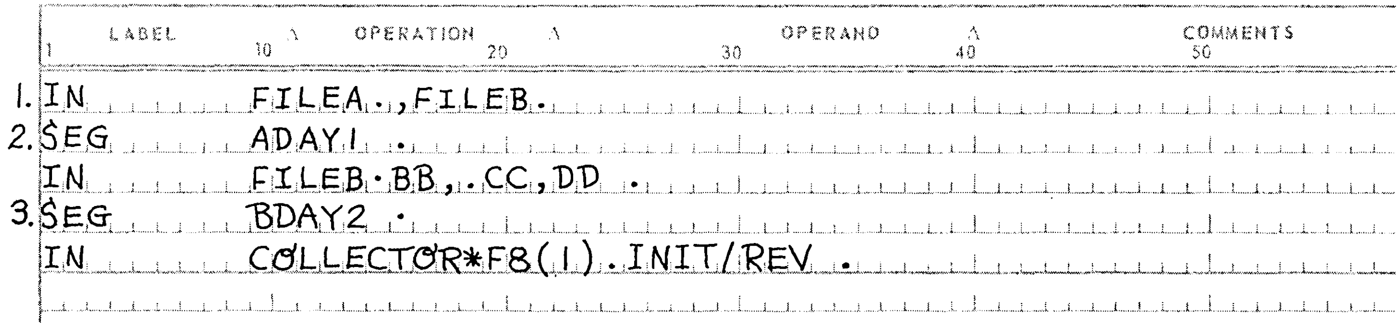
Elements named but not directly associated with a filename are searched for first in TPF$, then in any files named on LIB directives (see 10.2.2.3) and finally in the system relocatable library.

An element name may appear on only one IN directive and only once during a collection. A version name may be present, but the element name itself still may appear only once.

Common blocks may be named on IN directives but must not have an associated filename because they are embedded within other elements. When filename is present or implied by a period, the parameter is assumed to specify an element name. For inclusion of a common block in the collection see 10.2.4.6.

For the order of elements explicitly and implicitly included in the collection, see 10.2.3.2.

Examples:



```
   LABEL      OPERATION      OPERAND      COMMENTS
   1          10        20        30        40        50
1. IN         FILEA.,FILEB.
2. SEG        ADAY1 .
   IN         FILEB.BB,.CC,DD .
3. SEG        BDAY2 .
   IN         COLLECTOR*F8(1).INIT/REV .
```

1. All relocatable elements in FILEA and FILEB are included in the collection.

2. Elements BB and CC from file FILEB and element DD, whose filename is not indicated, are included in the collection of segment ADAY1.

3. The relocatable element INIT version REV in file COLLECTOR*F8 F-cycle 1 is included in the collection of segment BDAY2.

10.2.2.2. ELEMENT EXCLUSION (NOT)

Purpose:

Names the elements which are to be exclused from the collection.

All parameters in the NOT directive are optional. If all parameters are omitted, NOT TPF$ is assumed.

Format:

NOT name-1,name-2,...,name-n

Parameters:

names	Specifies the elements, with or without filename, and files to be excluded from the collection. If the version name or filename is omitted, all elements of the specified name are bypassed.

Description:

When all elements of a file are to be excluded, the entire file may be designated for exclusion with a NOT directive. A period must follow the filename to ensure that it is not interpreted as an element name.

If a filename with no following element names appears in a NOT directive, elements within the named file cannot be implicitly or explicitly included during the collection. The exception to this is TPF$ which even though designated for exclusion, individual elements may be explicitly included.

Examples:

```
    LABEL       OPERATION          OPERAND             COMMENTS
  1          10            20          30          40          50
1. @MAP,I   A,A
   NOT    CWW,LRR
2. @MAP,I   A,A
   IN     FILEA.
   NOT    FILEA.CL,.AB
3  @MAP,I   A,A
   IN   FL1.,FL2.
   NOT  FL1.XXX,.XX2,FL2.CAT,.CAT2
4. @MAP,I   A,A
   IN     FIL1.
   NOT    SYS$*RLIB$.
   NOT    TPF$.
```

1. Elements CWW and LRR are excluded from the collection; all other relocatable elements in TPF$ are included.

2. Elements CL and AB are excluded from the collection; all other relocatable elements in the file FILEA are included.

3. Elements XXX and XX2 from file FL1 and elements CAT and CAT2 from file FL2 are excluded from the collection; all other relocatable elements from files FL1 and FL2 are included.

4. Relocatable elements from SYS$*RLIB$ and TPF$ are excluded from the collection. All elements from FIL1 are included.

10.2.2.3. FILE SEARCH SEQUENCING (LIB)

Purpose:

Specifies which files (libraries) are to be searched by the collector prior to searching the system relocatable library SYS$*RLIB$.

All parameters in the LIB directive are optional except filename-1.

Format:

```
LIB   filename-1,filename-2,filename-3,...,filename-n
```

Parameters:

filenames — Specifies the files to be searched; named in the order in which they are to be searched. These files must have been @PREPed (see 8.2.11) prior to being specified on the LIB directive or the file is bypassed in the search sequence.

Description:

The specified files are searched for entry points which satisfy external references and for specified elements without filenames. TPF$. is always the first and SYS$*RLIB$. the last file searched unless they are named on a NOT directive.

The files are searched in the order in which they appear in the directive; a file may be searched more than once if the filename appears more than once on a LIB directive. A file is searched only once for each time it is specified on a LIB directive.

When several LIB directives are given, they have a cumulative effect. Thus, if file A has external references satisfied by elements in file B which in turn have external references satisfied by elements in file A and none of these elements are explicitly included by IN directives, the following directive is necessary to ensure the inclusion of all referenced elements:

```
LIB   A,B,A
```

Examples:

```
   LABEL      Δ  OPERATION   Δ              OPERAND     Δ              COMMENTS
  1          10            20             30            40             50
1. LIB   CHR1
2. LIB   USE1, USE2, USE3, USE1
```

1. File CHR1 is searched after TPF$. and before the system relocatable library.

2. Files USE1, USE2, USE3, and USE1 (a second time) are searched in that order after TPF$. and before the system relocatable library is searched.

10.2.2.4. EXTERNAL DEFINITION RETENTION (DEF)

Purpose:

Creates the ENTRY$ table. This table contains all the locations and names of the external definitions retained after the collection of the absolute or relocatable element.

NOTE:
The DEF and REF (see 10.2.2.5) directives are primarily useful in the collection of reentrant processors and with R-option collections.

All parameters in the DEF directive are optional.

Format:

DEF def-1,def-2,def-3,...,def-n

Parameters:

defs Specifies the external definitions to be retained.

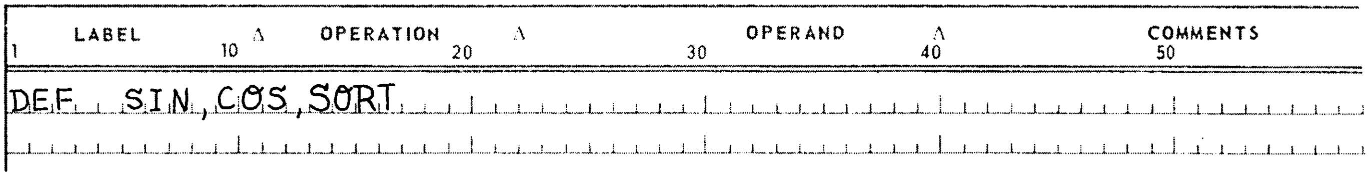
Description:

The DEF (and REF) directive cause the collector to build the COMMN$ table which defines the common blocks in a program. The user can address the two tables by the collector-defined names ENTRY$ and COMMN$, respectively (see 10.2.4.8).

If the R option was given on the @MAP control statement, the DEF directive must be used to specify those externalized labels which are to remain externalized in the merged relocatable output.

If no element explicitly named in an IN directive contains the named external definition, a search of the LIB files is made to find an element in which the symbol is defined.

Example:



```
     LABEL      Δ   OPERATION    Δ              OPERAND     Δ           COMMENTS
1           10               20             30             40             50
DEF  SIN,COS,SORT
```

The listed external definitions, SIN, COS, and SORT and their locations are retained after the collection in the ENTRY$ table of the resultant element.

10.2.2.5. EXTERNAL REFERENCE RETENTION (REF)

Purpose:

Creates a list of external references to be retained by the resulting absolute or relocatable element. No attempt is made to satisfy any references made to names indicated on REF directives. The table of retained external references is program addressable by the collector-defined symbol XREF$.

All parameters in the REF directive are optional.

Format:

REF ref-1,ref-2,ref-3,...,ref-n

Parameters:

refs Specifies the external references to be retained.

Description:

If an external definition that is identical to a REF name is encountered, a diagnostic is printed and the external definition is ignored.

Example:

LABEL	OPERATION	OPERAND	COMMENTS
1 10	20	30 40	50
REF ZIP,ZAP,DOD			

The listed external references (ZIP, ZAP, and DOD) are retained after the collection in XREF$ table of the resultant element. Any references to these symbols are satisfied with the address of the appropriate entry in the XREF$ table (see 10.2.4.8).

10.2.2.6. STARTING ADDRESS REDEFINITION (ENT)

Purpose:

Provides the user with the capability of overriding any start addresses provided by relocatable elements. Upon program initialization, control is transferred to the absolute address of the named symbol.

Format:

```
ENT   name
```

Parameter:

name — Must be an externally defined symbol which is the newly defined starting point.

Description:

In the absence of an ENT directive, the first start address encountered in processing the relocatable elements becomes the program start address. The start address must be contained in the main segment as it is the only segment initially loaded at execution time.

Example:

LABEL	OPERATION	OPERAND	COMMENTS
1 10	20	30 40	50
ENT GGYP			

Control is passed to the absolute address of the symbol GGYP in the main segment.

10.2.2.7. EXTERNAL REFERENCE DEFINITION (EQU)

Purpose:

Provides the means to assign, during the collection, a value to an undefined symbol or to change the value of an external definition.

All parameters in the EQU directive are optional except name-1/value-1.

Format:

```
EQU   name-1/value-1,name-2/value-2,...,name-n/value-n
```

Parameters:

names

Specifies the symbols to be defined.

values

The values to be assigned to the preceding name parameters.

The assigned values may be:

- octal integers (indicated by a leading zero)
- decimal integers
- a symbol
- a symbol with an offset (for example, BOB+4)

Description:

Any symbol used in the value parameter must be externally defined in one of the elements included in the collection. If an external definition which duplicates an EQU name is found, the external definition is ignored and a diagnostic is printed.

Examples:

```
    LABEL      Λ    OPERATION     Λ            OPERAND      Λ              COMMENTS
1          10                20               30           40              50
1. EQU  JOE/0200
2. EQU  AL/BOB+4
3. EQU  JOE/0200, ABE/SAM+10
```

1. The external reference JOE is defined as 200$_8$.

2. The external reference AL is defined as the value BOB+4$_{10}$.

3. The external references JOE and ABE are defined as 200$_8$ and SAM+10$_{10}$, respectively.

10.2.2.8. ELEMENT SELECTION DETERMINATION (CLASS)

Purpose:

Uniquely specifies one element version in a program file when more than one element has the same basic name but different version names. In the collection this occurs when:

- The version of the element was not specified on an IN directive and more than one relocatable element has that name;
- More than one relocatable element defines an external reference.
- A filename was not specified with the element on an IN directive and the element with different version names is present in more than one file.

Format:

CLASS string

Parameter:

string

Consists of 12 alphanumeric characters, asterisks, and blanks representing the versions of the elements. The string begins with the first nonblank character following the CLASS directive.

Description:

Successive CLASS directives have a cumulative effect and different ordering of CLASS directives may give different results.

Asterisks in a string represent characters in the version name to be ignored. Blanks in a string are valid.

When several elements qualify to be included in the collection, the collector compares the string parameter in the CLASS directive with the version names of the available elements. If the element version name is not identical to the string parameter, it is not included in the collection.

If, after the first comparison, more than one element qualifies, the string in the next CLASS directive is used in eliminating the remaining versions.

If all the CLASS directives have been used and there still remains more than one qualifying element, none of the remaining elements is used in the collection; a diagnostic message is given.

Examples:

```
     LABEL          OPERATION          OPERAND          COMMENTS
 1          10            20          30         40          50
1  @MAP,I  SAMP,ALPHA
   SEG   AARD
   IN  SIZE
   CLASS D***********
   CLASS ***B********
   CLASS *****4******
2. IN ELT1
   CLASS D*LA********
```

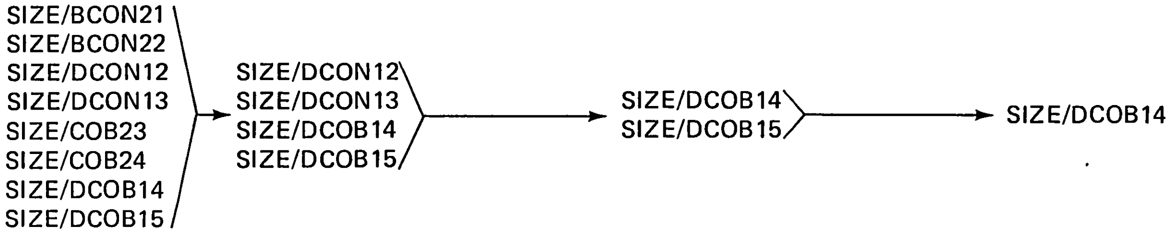
1. The IN directive does not specify which version of the element SIZE is to be used in the collection. The three CLASS directives specify that the version DCOB14 be used in the collection. Graphically this can be shown as follows:

The IN SIZE directive selects the following elements:

The CLASS D*********** directive selects the following elements:

The CLASS ***B******** directive selects the following elements:

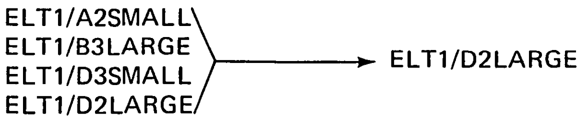
THE CLASS *****4******* directive makes the final element selection:



2. The CLASS D*LA******** STATEMENT SPECIFIES THAT VERSION D2LARGE of ELT1 element be used in the collection. Graphically this can be shown as follows:

The IN ELT1 directive selects the following elements:

The CLASS D*LA******** directives makes the final element selection:



10.2.2.9. CORRECTIONS FOR A RELOCATABLE ELEMENT (COR)

Purpose:

Specifies that a correction to a relocatable element is to be incorporated into the final absolute element with the original relocatable remaining unchanged. A COR statement may correct the following:

- an instruction word
- a data word
- a data word containing up to two symbols or values representing the upper and lower halves of the word.

Format:

```
    COR    eltname
```

The correction data images which follow the COR directive may be in any one of the following formats:

```
address,loccounter-1          f j a x h i u,loccounter-2,eltname-1

address,loccounter-1          dataword

address,loccounter-1          data,loccounter-2    data,loccounter-3
```

Parameters for the COR directive:

eltname — Specifies the element to which the corrections are to be made.

Parameters for the correction data images:

address,loccounter-1 — Specifies the relative address and location counter of the relocatable element to which the corrections are being made

f j a x h i u — Specifies field values in an instruction word that are to be inserted. The u field may be a:

- symbol
- symbol and offset
- octal number (a zero must precede the number)
- decimal number (no preceding zero)

dataword — Specifies a numeric correction

data — Specifies a symbol, symbol and offset, octal, or decimal correction

loccounter-2 — Specifies that the u and data fields are relative to the value of location counter LC2

eltname-1 — Specifies element in which loccounter-2 belongs if it is other than the element being corrected

Description:

The corrections contained in a COR directive are ignored if the R option was specified on the @MAP control statement.

Any number of correction statements may follow the COR directive.

COR statements cannot contain instructions which are jumps to any indirectly loaded segment.

A symbol must be an externalized entry point.

Examples:

```
   LABEL        OPERATION              OPERAND             COMMENTS
1           10           20          30          40          50
1. COR   ELT1
2. 000001,01   051 00 00 00 0 0 000011,01,ELT2
3. 000004,01   006 00 015 00 0 0 DUMMY+1
4. 000007,01   0000000000114
5. 000011,01   00000111 DUMMY+1
6. 000006,02   0000112,01    0000013
7. 000011,02   DUMMY+2
8. 000014,02   02,02  DUMMY+2
9. 000016,03   027 00 017 00 0 0  01176,02
```

1. Corrections to the relocatable element ELT1 are to be applied to the final absolute element.

2. The word being collected under element ELT1 at relative address 1_8, location counter 1, is modified to set function code to 51_8, and the j, a, x, h, and i fields to zero. The u field is given the value of 11_8 relative to location counter 1 of element ELT2.

3. The word, appearing at relative address 4_8 under location counter 1 of ELT1, is modified to set function code to 6_8, arithmetic register to 15_8; and u field to DUMMY+1. The j, x, h, and i fields are set to zero.

4. The word appearing at relative address 07_8 under location counter 1 is being changed to contain 114_8.

5. The word at relative address 11_8 under location counter 1 is changed to 111_8 in H1 and DUMMY+1 in H2.

6. The data at relative address 6_8 under location counter 2 contains 112_8, relative to location counter 1 of ELT1, in H1, and 13_8 in H2.

7. The data at relative address 11_8 under location counter 2 receives the value of symbol and offset ,DUMMY+2.

8. The data at relative address 14_8 under location counter 2 receives the data 2_8 relative to location counter 2 in H1, and the value of symbol DUMMY+2 in H2.

9. The instruction word appearing at relative address 16_8 under location counter 3 receives the function code 27_8; the A register 17_8; and a u field of 1176_8, relative to location counter 2 of ELT1. The j, x, h, and i fields are set to zero.

10.2.2.10. ADDING SNAPSHOT DUMPS (SNAP)

Purpose:

Specifies the elements in which a snapshot dump is to be taken. The snap data images immediately following the SNAP directive specify the address within the element where the SNAP is to be taken, length of the dump, and the frequency of the dump.

All parameters are optional except eltname.

SNAP Directive Format:

```
SNAP   eltname
```

Parameters:

eltname Specifies the element name where the dump is taken.

SNAP Data Image Format:

address-1,lc-1 address-2 length,reg times/frequency

All parameters are optional except address-1 and lc-1.

Parameters:

address-1,lc-1 Specifies the address and the location counter within the relocatable element of the instruction to be replaced with the dump request. This field may not contain a symbol.

address-2 Specifies the starting address of the program area to be dumped. This field may be in the form (all parameters optional) address,location-counter,element-name or symbol + offset.

length Specifies the length is words of the program area to be dumped.

reg Specifies which registers are to be dumped. The following codes are used:

0_8 – No registers dumped
1_8 – Only R registers dumped
2_8 – Only A registers dumped
3_8 – Both A and R registers dumped
4_8 – Only X registers dumped
5_8 – Both X and R registers dumped
6_8 – Both X and A registers dumped
7_8 – A, X, and R registers dumped

times Specifies the maximum number of times the snapshot dump is to be taken. If omitted, the value 100_{10} is assumed.

frequency Specifies at what frequency of reference the dump is to be taken. If omitted, the value 1 (which dumps every time the SNAP is encountered) is assumed.

Description:

No more than 16 snapshot dumps may be requested in any one collection. If more than one snapshot of the same element is to be taken, multiple specification statements may follow the SNAP directive.

When the dump request instruction SLJ SNAP$ is inserted at a specified address, the instruction appearing there is placed in a table in element SNAP$. After the dump is taken, the saved instruction is executed from within SNAP$ as if it had not been moved. If the saved instruction is a jump instruction, control transfers immediately to the location specified in the jump instruction; otherwise, control is transferred to the location following the location from which snap was called. Because the replaced instruction is executed from within SNAP$, the replaced instruction:

- Must not be altered during program execution.
- Must not be referenced as data or by indirect addressing.
- Must not be an SLJ instruction which specifies indirect addressing or indexing.
- Must not be an LMJ instruction which specifies indirect addressing or indexing.
- Must not be an EX instruction which references an LMJ or SLJ instruction.
- Must not be a test and skip instruction.
- Must not be used in reentrant code.

Example 1:

```
    LABEL        Λ   OPERATION     Λ              OPERAND      Λ               COMMENTS
 1            10               20               30              40               50
1. SNAP   LARK
2  010,1    012,1  020,07   0200,010
```

A snapshot dump is taken in element LARK. Line 2 gives the parameters for the dump. The instruction at address 10_8, under location counter 1 is the location where the snapshot request is placed. The address 12_8, under location counter 1 is the starting address of the dump. Sixteen locations in main storage are dumped along with the contents of the A, X, and R registers. The dump is to be taken 128 times at every eighth reference.

Example 2:

```
    LABEL        Λ   OPERATION     Λ              OPERAND      Λ               COMMENTS
 1            10               20               30              40               50
1. SNAP   JACK
2  0132,02   HAH+2    0400,04
```

A snapshot is to be taken in element JACK. Line 2 specifies that the instruction at location 132_8 under location counter 2 is the location where the snapshot request is placed. The address HAH+2 (where HAH must be externally defined) is the starting address of the dump. 256 or 400_8 main storage locations and the contents of the X registers are to be dumped. If the times and frequency parameters are not specified the system assumes a value of 100 for times and 1 for frequency.

10.2.2.11. END OF INPUT (END)

Purpose:

Specifies the end of the source language input for the collector. The END directive must precede any data statements which are unrelated to the @MAP control statement.

Format:

END

Example:

```
    LABEL        Λ   OPERATION     Λ              OPERAND      Λ               COMMENTS
 1            10               20               30              40               50
@MAP,IL
 series of
 collector directives
 END
```

10.2.2.12. ABSOLUTE ELEMENT OPTIMIZATION (MINGAP, MINSIZ)

Purpose:

Enables the user to modify the resultant absolute element so as to minimize the I/O transfer time when the program is loaded for execution.

Format:

```
MINGAP  value

MINSIZ  value
```

Parameter:

value	Specifies any positive integer

Description:

The program areas created by an assembler RES directive are unique in that at collection these areas do not contain meaningful data or instructions. The collector then has two alternatives when defining these RES areas within the generated absolute element:

(1) The area could be zero filled. This has the effect of increasing the size of the absolute element which affects the mass storage space requirements of the element as well as the number of words which must be transferred when the element is brought into main storage for execution.

(2) The area could be left void. This alternative decreases the size of the absolute element at the expense of increasing the number of access control words (ACW's) and hence the number of I/O operations needed to transfer the element to main storage for execution.

The collector uses a combination of these two alternatives depending upon the size of the RES area. Any area within the absolute element greater than or equal to MINGAP words is left void while those less than MINGAP words are zero filled. Each individual ACW required to transfer the element to main storage also controls a minimum of MINSIZ words. Both MINGAP and MINSIZ are initially set equal to 10.

While the value 10 is felt to be optimum in most cases and it generally does not need to be changed, there may be instances, depending upon type of mass storage and program application, where it is desirable to modify the parameters. For instance, increasing the number of words controlled by each ACW, and decreasing the number of I/O operations needed to transfer the program to main storage may reduce the time required to load the program.

Stressing the fact that the initial value of 10 should not be changed arbitrarily, the judicious modification of MINGAP and MINSIZ can produce an absolute element optimized for the particular situation.

10.2.2.13. PROGRAM SEGMENTATION (SEG)

Purpose:

Informs the collector of the beginning of a new program segment.

All parameters in the SEG directive are optional.

Format:

```
SEG  name-1,name-2
```

Parameters:

name-1	Specifies the name of the segment
name-2	Specifies the address relationship between the segment named in name-1 and the other program segments.

Description:

When name-1 is followed by an asterisk (*) the named segment is automatically loaded when referenced. The asterisk is allowed on all SEG directives, but is ignored if the directive defines the MAIN segment.

The name-2 parameter has several formats which determine the location of the segment named in name-1 as follows:

name-1	When name-2 is void, the starting address of the name-1 segment immediately follows the last address of the segment named on last preceding SEG directive.
name-1,name-2	Specifies that the starting address of the name-1 segment is the same as the starting address of the name-2 segment. These two segments can never exist in main storage at the same time.
name-1,(name-2)	Specifies that the starting address of the name-1 segment immediately follows the last address of the name-2 segment specified.
name-1,(name-2, name-3, . . . ,name-n)	Specifies that the starting address of the name-1 segment immediately follows the highest last address of the segments specified in name-2, name-3, name-4, and so forth.

Note that the highest last I bank address may be contained in a segment different than the one containing the highest last D bank address.

name-1,( )	Specifies that the starting address of name-1 segment immediately follows the last address of the longest of all segments previously named.

For additional information on the SEG directive, see 10.2.4.3.

10.2.2.14. RELOCATABLE SEGMENTS (RSEG)

Purpose:

Specifies the named segment as a relocatable segment. A relocatable segment (RSEG) is one whose location within the program is determined dynamically by the program during execution rather than at collection time. An RSEG may reference entry points within the program, but the RSEG itself may not contain definitions for references elsewhere in the program as the internal RSEG addressing is relocated at load time.

Format:

```
    RSEG   name
```

Parameter:

name	Specifies the relocatable segment.

Description:

For further information on relocatable segments, see 10.2.4.4.

10.2.2.15. DYNAMIC SEGMENT DEFINITION (DSEG)

Purpose:

To provide a mechanism by which the program area occupied by a segment will not be included in the initial program requirement.

Format:

```
DSEG   name-1,name-2
```

Parameters:

name-1	Specifies the name of the segment
name-2	Specifies the address relationship between the segment named in name-1 and the other program segments.

Description:

Dynamic segments are identical to normal overlay segments (defined by the SEG directives—see 10.2.2.13) in all aspects except one; the program area assigned exclusively to dynamic segments is not included when determining initial program size. It is the programmer's responsibility to guarantee that the program area is available by using the MCORE$ request (see 4.7.1) prior to requesting segment loading.

10.2.3. FUNCTIONAL ASPECTS OF THE COLLECTOR

After the collector has interpreted the parameters of the @MAP control statement (see 10.2.1) and the parameters of the collector directives (see 10.2.2), there remains the combining of the relocatable elements into a relocatable or absolute element and the insertion of the final element into the program file to complete the collection process.

10.2.3.1. COLLECTOR-PRODUCED RELOCATABLE ELEMENTS

Although the collector is generally used to produce an absolute element, a relocatable element can be produced by specifying the R option. All indicated relocatable elements are merged into a single element and only the external definitions specified in the DEF directive are retained. All other external definitions are submerged in the new relocatable element.

The R option is used most often when the user wants to include a relocatable element more than once in an absolute element. Initially, an R-option collection is performed. This combines the desired element with a specified set of relocatable elements. As long as no external definitions within the element are specified on the DEF directive, the desired element is submerged into the newly created relocatable element. The original relocatable element and the newly created relocatable element in which it has been submerged can both be collected in a single absolute element. Relocatable elements in SYS$*RLIB$ are not implicitly included in an R-option collection.

Only the following directives may be specified with the R option. All others produce a diagnostic message, and the collection continues.

DEF
ENT
IN
LIB
NOT
REF
CLASS
END

10.2.3.2. ELEMENT INCLUSION

Adding file elements to a collection is a two-part process:

(1) finding the files that were specified in the collector directives (see 10.2.2);

(2) finding within these files the elements that have been specifically named on IN directives (see 10.2.3.1) or that contain entry points which satisfy the undefined symbols.

Elements to be included in the collection may have been specified on IN directives (10.2.2.1) either with or without the filename in which those elements appear. For those elements with a filename present on the IN directive, the collector immediately references that file, finds the element, and processes the preamble of the element. After the preamble of a relocatable element has been processed, the text (instructions and data) of the relocatable element becomes part of the final output element.

After the elements with specified filenames are processed, the elements in TPF$. are processed.

If a @PREP (see 8.2.11) of TPF$. has not occurred, all relocatable elements in TPF$. are tentatively included in the collection unless specifically excluded (see NOT directive, (10.2.2.2). After all elements for a collection have been found, any nonreferenced element that was tentatively included from TPF$ is eliminated from the collection.

If a @PREP of TPF$. did occur, an element from TPF$. is included only if it is named on an IN directive or if one of its external definitions satisfies an undefined reference from another element included in the collection. When a @PREP of TPF$. has occurred, TPF$. is always the first file searched when attempting to locate elements named without a filename and elements with external definitions satisfying undefined references.

The situation may arise where the user wants to implicitly include elements from a file other than TPF$. which have entry point names or element names which duplicate entry point or element names in TPF$. Since TPF$. is searched prior to searching other files, the elements from TPF$. are included instead of the desired elements which are in other files.

Therefore, if duplicates of element names and external definitions are present and those in TPF$. are not wanted in the collection, a @PREP of TPF$. is needed to prevent automatic inclusion of the TPF$. elements. It is also necessary to specifically IN by filename and element name any elements from other files which have element names or external definitions duplicated by TPF$. elements.

For those elements without a file name on the IN directive, the collector searches files for these elements in the following order:

(1) The temporary program file (TPF$)

(2) User-defined files that were indicated by the LIB directive (see 10.2.2.3)

(3) The system relocatable library (SYS$*RLIB$)

In an attempt to satisfy all undefined references in the collection, the collector searches the specified files for elements that have entry point names that correspond to the symbol names appearing in the collector created UNDE table (see 10.2.3.5). The UNDE table contains all the symbols that are present in the undefined symbol tables of the processed element preambles. When an element is found with an entry point name corresponding to a symbol name in the UNDE table, the preamble of that element is processed and the now defined symbol is removed from the UNDE table. The order of search for undefined symbols is:

(1) The temporary program file (TPF$)

(2) User-defined files defined by the LIB directive (see 10.2.2.3) and previously prepared by the @PREP control statement (see 8.2.11).

(3) The system relocatable library (SYS$*RLIB$).

The included elements are placed in the instruction and data areas of the final absolute element. Odd numbered location counters of an element are assigned to the instruction area. Even numbered location counters and common blocks are assigned to the data area. See 10.2.2.1 and 10.2.4.6 for information on specific placement of common blocks.

The most efficient collection results when every element desired in the collection is explicitly named, including filenames; this eliminates @PREP requirements and library searches.

10.2.3.3. SEGMENTED VERSUS NONSEGMENTED PROGRAMS

The absolute element resulting from collecting of various relocatable elements may or may not be segmented (see 10.2.4). However, a nonsegmented program can be functionally considered a segmented program with only a main segment.

10.2.3.4. COLLECTING REENTRANT PROCESSORS

In creating reentrant processors it is not only more efficient to explicitly name all elements including their filenames but it is also extremely advisable. The nature of reentrant processors dictates that from one collection to another all elements be located in the same relative position within the absolute element. This can only be ensured by explictly incuding all elements in every collection of the absolute element. (See 10.5 for reentrant processor preparation.)

10.2.3.5. PROCESSING ELEMENT PREAMBLES

An element preamble is attached to every relocatable element created by the language processors and the collector. The element preamble provides information which is needed when collecting relocatable elements together to form an absolute or single relocatable element.

- The definition and location of each externalized entry point in the element.
- The length in words, under each location counter in the element.
- A table of the undefined symbols appearing in the element.
- Common blocks in the element.

When the preamble is processed:

- The entry points in the element are added to the collector entry point table (EP table).
- Undefined symbols appearing in the element which have no corresponding entry in the EP table are listed in the UNDE table.
- Undefined symbols in the element which have a corresponding name to an entry point in another element are linked to the EP table.

Symbols are removed from the UNDE table as corresponding entry point names are found. Newly encountered undefined symbol names are added at the end of the UNDE table.

10.2.4. PROGRAM SEGMENTATION

When an absolute program is being executed, it must reside in main storage. There may not be, however, enough available area in main storage to contain the complete program. Therefore, the program may be segmented so that the various segments can be loaded into main storage as the program is being executed.

Even when the total program size may fit into main storage, many times it is advantageous to subdivide the program into functionally independent units (segments) which are loaded into main storage only when needed. This reduction in the program's main storage requirements reduces main storage impact while allowing increased storage utilization.

A segmented program consists of:

- one main segment which resides in main storage throughout the execution of the program
- subordinate segments which are loaded into main storage as they are needed

As each subordinate segment is loaded into main storage, it may overlay all or part of a previously loaded overlay segment. The area overlayed is equal in size to the size of the new overlay segment. The main segment is never allowed to be overlayed except by a relocatable segment (RSEG).

10.2.4.1. SEGMENTATION DIRECTIVES

The directives needed to specify program segmentation are as follows:

- SEG – Informs the collector of the beginning of a segment (see 10.2.2.13).
- RSEG– Informs the collector of the beginning of a relocatable segment (see 10.2.2.14).
- DSEG– Informs the collector of the beginning of a dynamic segment (see 10.2.2.15).
- IN – Specifies the elements to be included in the segment (see 10.2.2.1).
- NOT – Specifies which elements are to be excluded (see 10.2.2.2).

Segments may be loaded and executed independently; however, elements common to several segments must be in main storage when the referencing segments are executed.

Since each segment has a path leading back to the main segment, elements which are implicitly included and which are referenced by two or more segments are attached to the segment which is located at the common point in the paths to the main segment of all referencing segments. Elements specified on IN directives are never moved from the segment in which they were specifically placed.

10.2.4.2. INSTRUCTION AND DATA AREAS

Every program containing segments in addition to the main segment always has a D bank. If there is no program data in the D bank, then it contains at least a segment load table. The segment load table contains an entry with information about every segment of the program. It is located preceding the user's main segment D bank. Since the segment load table has no main storage protection, special care has to be taken not to destroy its information.

The program's data, when it exists, is located after the segment load table and any other collector-produced tables, such as the ENTRY$, COMMN$, XREF$, and indirect load table.

The first address of the I bank (instruction area) is assigned 1000_8. The starting address of the D bank (data area) is dependent upon the size of the I bank, the possible use of the assembler SETMIN directive, the total program size, and the options specified on the @MAP control statement. The first address of the D bank, however, is always a multiple of 1000_8 and is usually given the value 40000_8.

Odd numbered location counters are assigned to the I bank; even numbered location counters and common blocks are assigned to the D bank.

The user program can reference the first and last I bank and the first and last D bank addresses by the symbols: FRSTI$, LASTI$, FRSTD$, and LASTD$, respectively. The collector replaces these symbols with the actual assigned address values.

An unnamed blank common block, if required in the program, is attached to the main segment under the name BLANK$COMMON. It may be positioned in another segment by an IN BLANK$COMMON directive.

Named common blocks are attached to the segment (if not named in an IN directive) which is located at the common point in the paths to the main segment of all segments referencing it.

10.2.4.3. SEG DIRECTIVE CONSIDERATIONS

The IN directive specifies the files and elements to be included in a segment. If no SEG directive is encountered prior to the first IN directive following the @MAP control statement, a SEG MAIN directive is assumed by the collector and it applies to all following directives until a SEG directive is encountered.

The segment name is 1 to 12 alphanumeric characters (with the $ and – allowed) in length. The segment name must not be the same as any entry point name in the collection, and must contain at least one alpha character.

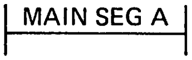
Within a segment, any elements included to satisfy undefined symbols are located at the beginning of the segment in the inverse order of their inclusion, that is, the last included element is the first element in the segment. Following any implicitly included elements are those named on IN directives in the exact order they were named. When all elements within a file are included in a segment by specifying only the filename on the IN directive, the ordering of the file's elements is random.

Example 1:

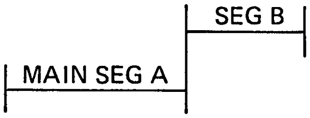
```
   LABEL          OPERATION          OPERAND          COMMENTS
1         10               20        30         40          50
1. SEG A
2. SEG B
3. SEG C, (A)
```

The program is to be divided into three segments.

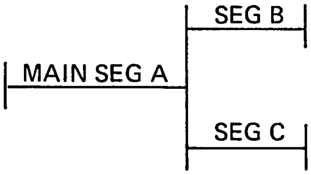
1. Specifies segment A as the main segment.



2. Specifies that the starting address of segment B is immediately after the last address of main segment A.



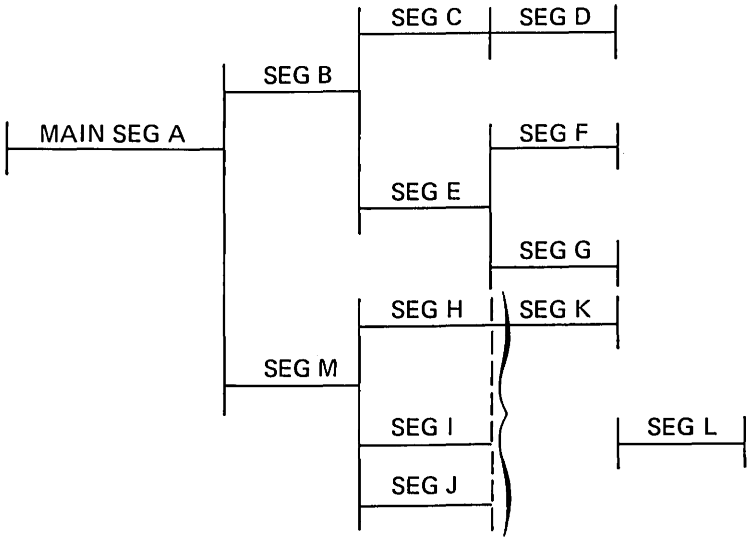
3. Specifies that the starting address of segment C is immeidately after the last address of main segment A. Segments B and C can never exist in main storage at the same time.



Example 2:

```
   LABEL          OPERATION          OPERAND          COMMENTS
1         10               20        30         40          50
1.  SEG A
2.  SEG B
3.  SEG C
4.  SEG D, (C)
5.  SEG E, C
6.  SEG F, (E)
7.  SEG G, F
8.  SEG M, B
9.  SEG H, (M)
10. SEG I, H
11. SEG J, H
12. SEG K, (H, I, J)
13. SEG L, (I)
```

The program is to be divided into thirteen segments.



1. Specifies segment A as the main segment.
2. Specifies that the starting address of segment B is immediately after the last address of main segment A.
3. Specifies that the starting address of segment C is immediately after the last address of segment B.
4. Specifies that the starting address of segment D is immediately after the last address of segment C.
5. Specifies that the starting address of segment E is the same as the starting address of segment C.
6. Specifies that the starting address of segment F is immediately after the last address of segment E.
7. Specifies that the starting address of segment G is the same as the starting address of segment F.
8. Specifies that the starting address of segment M is the same as the starting address of segment B.
9. Specifies that the starting address of segment H is immediately after the last address of segment M.
10. Specifies that the starting address of segment I is the same as the starting address of segment H.
11. Specifies that the starting address of segment J is the same as the starting address of segment H.
12. Specifies that the starting address of segment K is immediately after the last address of either segment H, I, or J, whichever segment is longest.
13. Specifies that the starting address of segment L is immediately after the last address of the longest segment in the set: A, B, C, D, E, F, G, M, H, I, J, and K.

10.2.4.4. RSEG DIRECTIVE CONSIDERATIONS

The elements included in the relocatable segment should be explicitly named on IN directives (see 10.2.2.1). When an element which is referenced by more than one segment is implicitly included it is placed in a segment other than the RSEG. Generally, it is advisable if an element is referenced by more than one segment (one of which is an RSEG), that the element be explicitly included in the main segment.

Relocatable segments may not be indirectly loaded. See 10.2.4.5.2 for the direct method of loading segments.

The starting address of a relocatable segment has no relationship to other segments in the collection. It may be loaded at whatever starting address within the program limits is given in register A2 during the LOAD$ calling sequence (see 10.2.4.5.1). The LOAD$ request adds the value in register A2 to all relative address references internal to the named relocatable segment. Any references to RSEG labels from outside the relocatable segment must be user modified by the value in register A2.

All instructions and data in a relocatable segment are collected together with all odd location counters followed by all even location counters.

A relocatable segment may be loaded into either the I or D bank of the program.

Example:

```
LABEL         OPERATION            OPERAND              COMMENTS
1         10          20         30          40          50
RSEG   ORSON
IN     ORSON
```

Element ORSON is specified for inclusion in the relocatable segment ORSON.

10.2.4.5. LOADING PROGRAM SEGMENTS

When a segmented program is called by a @XQT control statement (see 10.3.1), only the main segment is initially loaded. The subordinate segments are loaded by either:

- the direct method, or
- the indirect method.

10.2.4.5.1. DIRECT METHOD (L$OAD AND LOAD$)

When using the direct method of loading, use either:

- the L$OAD procedure, or
- the executive request LOAD$.

Format of the L$OAD procedure:

```
L$OAD       name,jump,clear,rseg-addr
```

Parameters:

name
Specifies the name of the segment to be loaded.

jump
Specifies the location where control is to be transferred after the segment is loaded; if omitted, control passes to the location following the LOAD$ request.

clear
If greater than zero, the program area containing the segment to be loaded is not zero filled prior to segment load. If zero, the area to be occupied by the segment is zero filled prior to segment load.

rseg-addr — If the segment was specified on the @MAP control statement (see 10.2.1) as relocatable, this parameter specifies the starting address for the relocatable segment. If omitted when loading a relocatable segment, the address must be in register A2 before the call is made:

```
L,U       A2,rseg-address
```

The address may be defined as an octal value or a tag not contained in the RSEG.

Description

The L$OAD procedure produces a sequence of code which loads: register A0 with the segment name; register A1 with return control address; and generates the LOAD$ request. The LOAD$ request takes the form:

```
L,U       A0,segname
L,U       A1,addr
ER        LOAD$
```

Segname is the same as the contents of the name-1 parameter in the SEG directive (see 10.2.2.13).

When bit 35 of register A0 is set, the segment loader does not clear the main storage area to be occupied by the segment. This decreases the time required to load the segment, but as a result, any areas within the segment created by RES cannot be assumed to be zero.

Examples:

```
     LABEL     Δ    OPERATION     Δ              OPERAND       Δ            COMMENTS
 1          10               20            30              40              50
1. L$OAD  NEW,ORG1
2. L$OAD  CAP,YELL,0,01350
```

1. After segment NEW is loaded, transfer control to location ORG1. The area occupied by segment NEW is zero filled prior to loading. The L$OAD procedure produces the same effect as the code:

```
     LABEL     Δ    OPERATION     Δ              OPERAND       Δ            COMMENTS
 1          10               20            30              40              50
       L,U     A1,ORG1    .
       L,U     A0,NEW     .
       ER      LOAD$
```

2. The area to be occupied by relocatable segment CAP is zero filled prior to loading. The starting address for segment CAP is 1350_8. After the segment is loaded, control is passed to YELL. The L$OAD procedure produces the same effect as the code:

```
     LABEL     Δ    OPERATION     Δ              OPERAND       Δ            COMMENTS
 1          10               20            30              40              50
     L,U       A2,01350
     L,U       A1,YELL
     L,U       A0,CAP
     ER        LOAD$
```

10.2.4.5.2. INDIRECT METHOD

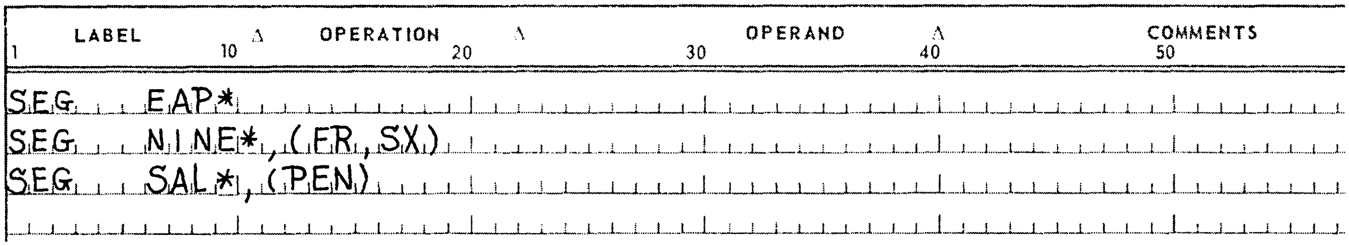
Whenever a segment that is marked for indirect loading is referenced by any jump instruction that passes control to the segment's I bank area, the segment is automatically loaded if it is not already in main storage. Segments to be loaded by the indirect method must be so marked on the SEG directive. The mechanics for such loading are set up by the collector and carried out by the segment loader. The collector replaces the address portion of the jump command with the address of an indirect load table entry. The indirect load table performs an SLJ instruction to the indirect load routine which, in turn, performs an ER to the segment loader (if the segment is not already loaded) and jumps to the location of the externally defined symbol. All registers are preserved by the process. The indirect load table is assigned to the data area of the main segment.

If indirect loading is used, the reference may not be made to an external symbol with an offset.

If the B option was specified on the @MAP control statement, the indirect load routine indicates that the segment's main storage area need not be zero filled.

Segments marked for indirect loading may also be loaded by the direct method.

Example:



LABEL Δ OPERATION Δ OPERAND Δ COMMENTS
1 10 20 30 40 50

```
SEG   EAP*
SEG   NINE*,(ER,SX)
SEG   SAL*,(PEN)
```

Segments EAP, NINE, and SAL are automatically loaded when any externalized I-bank entry point is referenced.

10.2.4.5.3. RELOADING THE MAIN SEGMENT

It may be desirable to reestablish the main segment of a program for either error recovery or reinitialization. This is done by the LOAD$ request. The LOAD$ request reloads the entire main segment including all collector produced tables. The main storage requirements remain unchanged. The calling sequence is:

```
L     A0,(CLEAR,0400000)
L,U   A1,REENTRY·ADDR
ER    LOAD$
```

The first coding line loads register A0 with the segment-id of 400000_8. The CLEAR parameter functions as follows:

- If clear is less than 0, the main segment area is not cleared before loading.
- If clear is greater than or equal to 0, the main segment area is cleared.

The second coding line loads register A1 with the reentry-address according to the following:

- If the reentry-addr is equal to 0, control is returned to the instruction following the LOAD$ request.
- If the reentry-addr is not zero, control is passed to that address.

10.2.4.6. USE OF COMMON BLOCKS

The collector produces, in the absolute element, load control specifications for the LOAD$ routine. These specifications indicate which text words (data and instructions) are to be put at which locations in main storage when the segment is loaded.

If a common block is given initial values (filled with text, rather than simply set aside as a reserved area), the collector produces specifications to put in these values when the segment containing the element which defines the initial values is loaded.

For example, if five different elements define five different initial values for the same common block and each of these five elements was in a different segment, the same common block located in a segment common to all referencing segments would be reinitialized each time one of the five different segments was loaded. This occurs regardless of where the referenced common block is located within the user's program area.

Any areas of the common blocks in which text is not loaded upon reinitialization are not changed as long as the reinitialization is caused by the loading of a segment other than the one in which the common block resides.

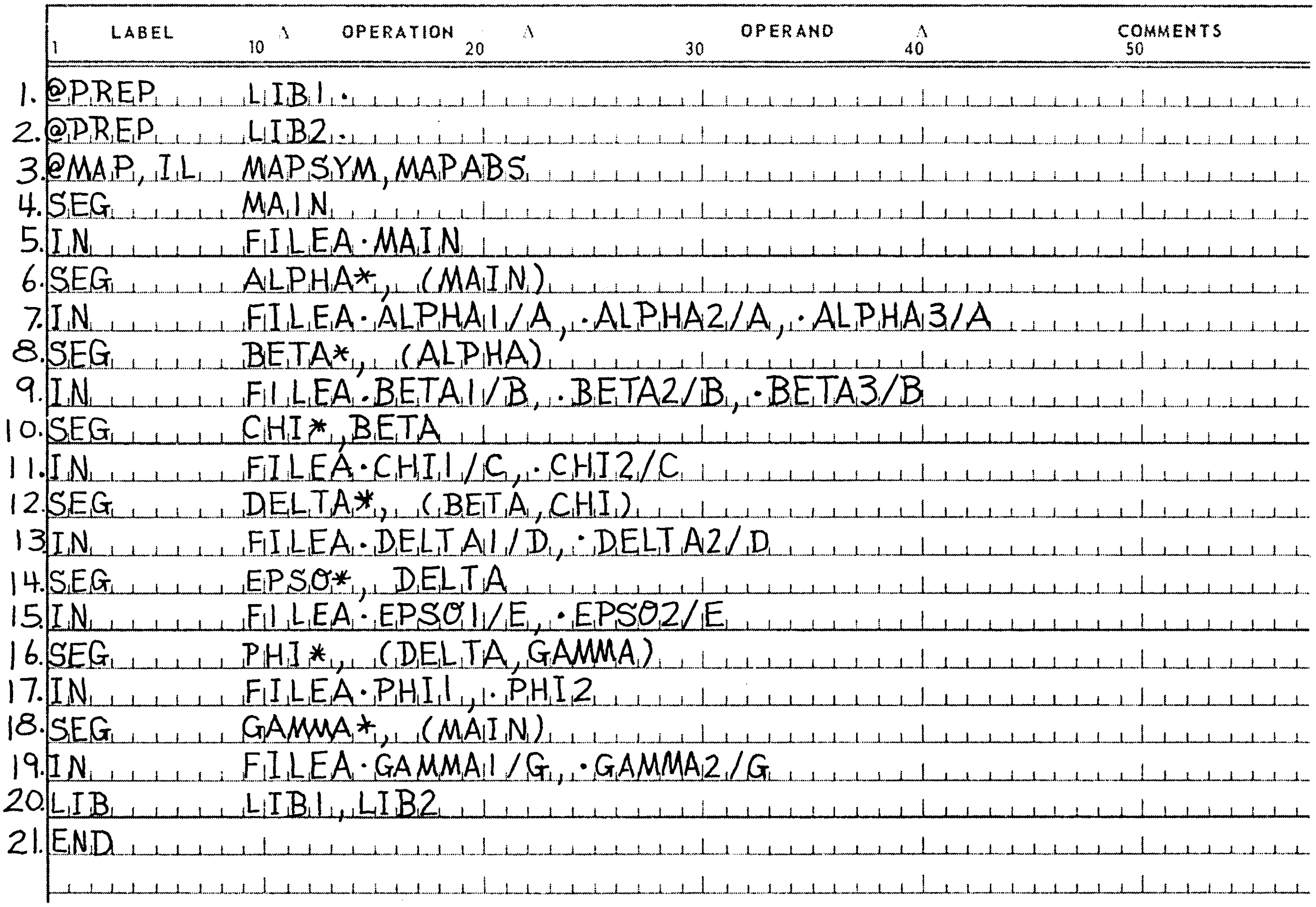
On IN directives (see 10.2.2.1), common blocks are always specified without a file name. A common block name must not be identical to an element name.

10.2.4.7. SEGMENTATION EXAMPLE

The following is an example of a segmented program. The elements in file FILEA and their required outside references are shown below:

FILE A	Elements in Which FILE A References are Defined
MAIN	FILE A . ALPHA 1, BETA 1, PHI1
ALPHA1/A	
ALPHA2/A	LIB 1 . SIN/X1
ALPHA3/A	LIB 2 . COS/X2
BETA1/B	LIB 1 . SQRT/X1
BETA2/B	
BETA3/B	
CHI1/C	LIB 1 . SQRT/X1
CHI2/C	
CHI3/C	
DELTA1/D	LIB 2 . CAT/Y5
DELTA2/D	
EPSO1/E	LIB 2 . CAT/Y5
EPSO2/E	
PHI1	
PHI2	
GAMMA1/G	LIB 1 . SIN/X1
GAMMA2/G	LIB 2 . COS/X2

The following coding is used to produce the segmented program.


LABEL
OPERATION
OPERAND
COMMENTS
1
10
20
30
40
50

```
 1. @PREP    LIB1.
 2. @PREP    LIB2.
 3. @MAP,IL  MAPSYM,MAPABS
 4. SEG      MAIN
 5. IN       FILEA.MAIN
 6. SEG      ALPHA*,(MAIN)
 7. IN       FILEA.ALPHA1/A,.ALPHA2/A,.ALPHA3/A
 8. SEG      BETA*,(ALPHA)
 9. IN       FILEA.BETA1/B,.BETA2/B,.BETA3/B
10. SEG      CHI*,BETA
11. IN       FILEA.CHI1/C,.CHI2/C
12. SEG      DELTA*,(BETA,CHI)
13. IN       FILEA.DELTA1/D,.DELTA2/D
14. SEG      EPS0*,DELTA
15. IN       FILEA.EPS01/E,.EPS02/E
16. SEG      PHI*,(DELTA,GAMMA)
17. IN       FILEA.PHI1,.PHI2
18. SEG      GAMMA*,(MAIN)
19. IN       FILEA.GAMMA1/G,.GAMMA2/G
20. LIB      LIB1,LIB2
21. END
```

1,2. Entry point tables are prepared for files LIB1 and LIB2.

3. Calls the collector. The I option specifies that symbolic element MAPSYM is introduced from the run stream. The L option specifies that a complete listing is to be produced. The absolute output element is called MAPABS. Both MAPSYM and MAPABS are placed in TPF$.

4. Segment MAIN is the program's main segment.

5. Element MAIN is found in FILEA file.

6. Segment ALPHA is marked for indirect loading. The starting address of ALPHA follows the last address of segment MAIN.

7. Elements ALPHA1/A, ALPHA2/A, and ALPHA3/A are found in FILEA file and are included in the collection of ALPHA segment.

8. Segment BETA is marked for indirect loading. The starting address of BETA follows the last address of segment ALPHA.

9. Elements BETA1/B, BETA2/B, and BETA3/B are found in FILEA file and are included in the collection of BETA segment.

10. Segment CHI is marked for indirect loading. The starting address of CHI segment is the same as the starting address of segment BETA.

11. Elements CHI1/C and CHI2/C are in FILEA and are included in the collection of the CHI segment.

12. Segment DELTA is marked for indirect loading. The starting address of DELTA segment follows the last address of either segment BETA or segment CHI, whichever is longer.

13. Elements DELTA1/D and DELTA2/D are in FILEA file and are included in the collection of segment DELTA.

14. Segment EPSO is marked for indirect loading. The starting address of EPSO segment is the same as the starting address of segment DELTA.

15. Elements EPSO1/E and EPSO2/E are in FILEA file and are included in the collection for segment EPSO.

16. Segment PHI is marked for indirect loading. The starting address of PHI segment follows the last address of either segment DELTA or segment GAMMA, whichever is longer.

17. Elements PHI1 and PHI2 in FILEA file are included in the collection of PHI segment.

18. GAMMA segment is marked for indirect loading. The starting address of GAMMA segment follows the last address of segment MAIN.

19. Elements GAMMA1/G and GAMMA2/G are in FILEA file and are included in the collection of segment GAMMA.

20. Files LIB1 and LIB2 are searched prior to searching the system library for the collection elements.

21. End of the collector directives.

Figures 10–1 and 10–2 show the instruction and data areas of main storage for the preceding example.

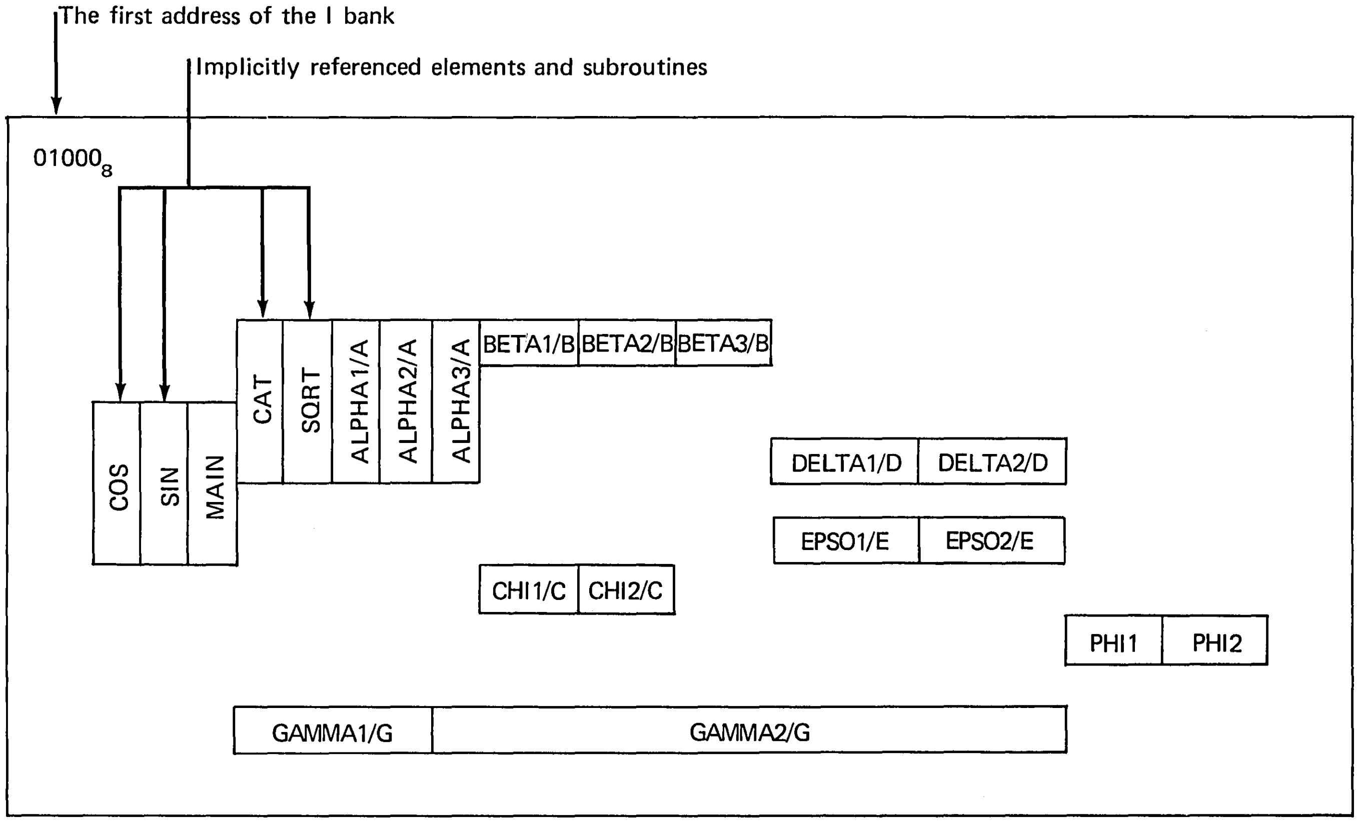


Figure 10–1. Instruction Area (I Bank) Main Storage Map for the Segmented FILEA

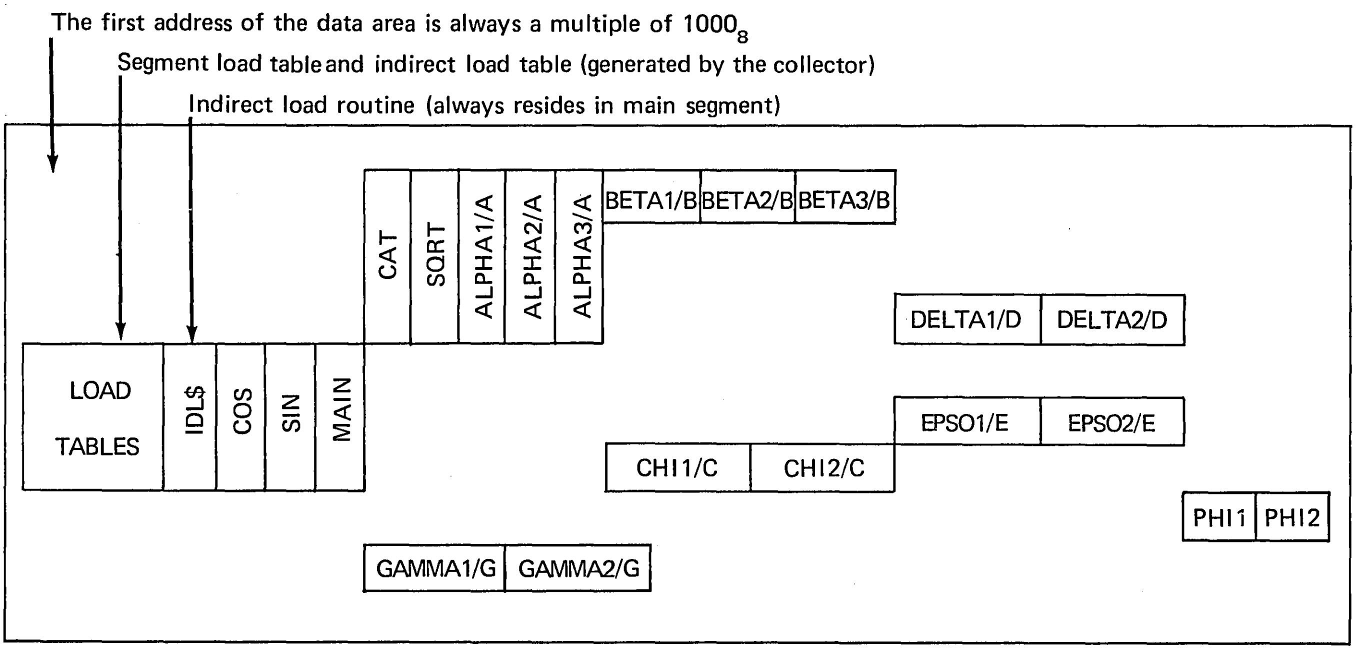

The first address of the data area is always a multiple of 1000_8
Segment load table and indirect load table (generated by the collector)
Indirect load routine (always resides in main segment)
LOAD TABLES
IDL$
COS
SIN
MAIN
CAT
SQRT
ALPHA1/A
ALPHA2/A
ALPHA3/A
BETA1/B
BETA2/B
BETA3/B
DELTA1/D
DELTA2/D
EPSO1/E
EPSO2/E
CHI1/C
CHI2/C
PHI1
PHI2
GAMMA1/G
GAMMA2/G

Figure 10–2. Data Area (D Bank) Main Storage Map for the Segmented FILEA

10.2.4.8. COLLECTOR GENERATED TABLES

Entry Point Table (ENTRY$):

Word		
0	000000	nbr-of-entries
1	entry-point-name	
2		
3	value-of-entry-point-program-addr	

The value of the entry point is the address of the reference vector entry of the entry point if in a segment designated for indirect loading.

Common Block Table (COMMN$):

Word		
0	000000	nbr-of-entries
1	common-block-name BLANK$COMMON for-blank-common	
2		
3	length-of-common-block	addr-of-common-block

The entry point table includes only those entry points specified on the DEF statement. When the DEF statement is present, the common block table is included in the absolute program.

External Reference Table (XREF$):

Word		
0	000000	nbr-of-entries
1	external-reference-name	
2		
3	ER ERR$	

The external reference is assigned the address of the third word of its entry in the external reference table entry.

NOTE:

The first addresses of the entry point table, common block table, and external reference table, are assigned, respectively, to the following external definitions (which may be directly referenced in a user program): ENTRY$, COMMN$, and XREF$. If no table exists, this value is zero.

10.3. PROGRAM EXECUTION

10.3.1. INITIATING EXECUTION (@XQT)

Purpose:

Used to initiate the execution of an absolute program prepared by the collector.

All parameters in the @XQT control statements are optional except @ and XQT.

Format:

@label:XQT,options   eltname

Parameters:

options — Options are user specified and are presented to the program initially in a register or may be retrieved by an OPT$ request (see 4.8.2).

eltname — Specifies the absolute element to be executed. If no filename is specified, TPF$ is assumed. If no eltname parameter is specified, the most recent absolute element placed in TPF$ is assumed. When no absolute element exists in TPF$, all relocatable elements in TPF$ are collected and the resultant absolute element is executed. The newly created absolute element is given the name NAME$ and placed in TPF$.

Examples:

```
     LABEL     Δ  OPERATION   Δ          OPERAND      Δ             COMMENTS
   1          10            20          30            40            50
1. @XQT
2. @XQT   FILEA.XYZ
3. @XQT  ABC
```

1. Execute the most recent absolute element placed in TPF$. If no absolute element is present, the collector creates an absolute element (NAME$) from all relocatable elements in TPF$.

2. Execute absolute element XYZ in file FILEA.

3. Execute absolute element ABC in file TPF$.

10.3.1.1. INITIAL EXECUTION STATUS

When an absolute program is loaded into main storage for execution as the result of an @XQT control statement or processor call statement, only the main segment of that program is loaded. Other program segments, if they exist are loaded only at the request of the executing program by directly or indirectly referencing the LOAD$ request.

The program begins execution at the collector-defined start address with one activity. This activity possesses the major register set.

Certain of the initial activity's registers are preloaded by the executive with generally useful information. These registers and the contents are:

Register	Contents	
A4	program type: 4 Demand 5 Deadline batch 6 Batch	
A5	Options from @XQTor processor call statement A Bit 25, B = bit 24,...., Z = bit 0	
R1	Date (in Fieldata DATE$ format)	
	Bits 35–24	Month (01 = Jan. 02 = Feb,etc.)
	Bits 23–12	Day of the month (01–31)
	Bits 11–0	Year (last two digits of the year)
R2	Current time (TDATE$) format)	
	Bits 35–30	Month
	Bits 29–24	Day
	Bits 23–16	Year (modulo 64)
	Bits 17–0	Time in seconds from midnight
R3	Accumulated CPU time for the run (in 200 microsecond increments)	

10.3.2. PROGRAM DATA SEPARATION (@EOF)

Data images to be read by the program may follow the @XQT control statement. The program uses the system reference READ$ (or equivalent) in gaining access to all data images. When an executive control statement other than an @EOF is detected by READ$, further reading by the program is inhibited and an end-of-data return is given. Those data images not read by the program are bypassed when the program is finished (a message denoting this is placed in the run's print file).

Purpose:

Used as a file divider (general sentinel) within the data stream which follows the @XQT control statement (or processor control statement) that can be bypassed (read) by a user program.

Format:

```
@EOF   s
```

Description:

s is a one-character, user-defined sentinel in column six of the control statement. This sentinel is passed to the requesting activity when the @EOF control statement is encountered. When the @EOF control statement is detected by READ$ request (see 5.2.1) an abnormal return is made to the requesting activity and the s parameter is placed in bits 5–0 of the activity's A0 register. A subsequent READ$ request causes the next image to be transmitted. The @EOF portion of the @EOF control statement is not transmitted to the user.

For the additional information on using @EOF statments and detecting end-of-data (see 5.2.1).

An example where the @EOF control statement is used is:

```
@XQT PROGX

......

data of part 1

......

@EOF A

......

data of part 2

......

@XQT PROGY
```

All data between the two @XQT statements are to be read by PROGX. The @EOF control statement serves as a marker between the two logical data sets.

10.4 REENTRANT PROCESSOR EXECUTION

10.4.1. GENERAL

A reentrant processor (REP) is an executable reentrant routine referenced from a user's program by the LINK$ or RLINK$ executive requests. A reentrant processor consists of only I bank addresses and resides as an absolute element in the system library (SYS$*LIB$) or a user file. A REP may be referenced many times during a user run without being reloaded and may access the D bank of the calling program. For purposes of debugging, a reentrant processor may reside on mass storage in a user-specified file. There are two types of reentrant processors:

- System standard reentrant processors listed at systems generation time.
- User-specified reentrant processors listed by the RLIST$ executive request (see 10.4.3).

10.4.2. SEARCHING THE REENTRANT PROCESSOR LISTS

A reentrant processor may exist in one of three lists which are searched each time a reentrant processor is called. They are:

- A permanent main storage list of SYS$*LIB$ reentrant processors created at systems generation time.
- A dynamic list of reentrant processors in SYS$*LIB$ not in the permanent list but created as the processors are referenced.
- A list of reentrant processors created by the RLIST$ request (see 10.4.3).

When a reentrant processor is called by the LINK$ or RLINK$ request, the order of search is:

(1) The list of reentrant processors created by the RLIST$ request.

(2) The dynamic list of reentrant processors in SYS$*LIB$.

(3) The permanent list of reentrant processors in SYS$*LIB$.

(4) The elements residing in SYS$*LIB$.

10.4.3. ENTERING A LIST OF USER-CREATED REENTRANT PROCESSORS (RLIST$)

Purpose:

Enters a list of user-created reentrant processors (list is in a file assigned to the run).

Format:

```
L       A0,(nbr-entries,pktaddr)
ER      RLIST$
```

Description:

H2 of register A0 is loaded with the address of the packet (pktaddr) containing the names of the reentrant processors. The packet format is:

Word	35 … 0
0 1	filename
2	reentrant-processor-name-1
3	reentrant-processor-name-2
4	reentrant-processor-name-3
5 ↓	
21	reentrant-processor-name-20

H1 of register A0 is loaded with the number of reentrant processor names (nbr-entries) in the packet.

The reentrant processor names are absolute element names consisting of six or less characters with a version name consisting of all blanks.

Once a list has been entered by the RLIST$ request, the user may access any processor on the list for the remainder of the run. The RLIST$ request permits new reentrant processors to be tested before being placed in SYS$*LIB$.

Each RLIST$ request replaces the current list, if any, with a new list. An RLIST$ request with the nbr-entries parameter equal to zero deletes all entries.

A maximum of 20 reentrant processors can be named in a RLIST$ request.

10.4.4. REFERENCING A REENTRANT PROCESSOR (LINK$ AND RLINK$)

10.4.4.1. LINK$ EXECUTIVE REQUEST

Purpose:

Transfers control to any reentrant processor in SYS$*LIB$ or the RLIST$ list.

Format:

```
L    A0,('repname')
ER   LINK$
```

Parameters:

repname	Specifies a six-character reentrant processor name.

Description:

After the reentrant processor is loaded (see 10.4.7), control is returned to the starting address of the reentrant processor.

All scheduling facilities, accounting, and other pertinent information of the calling activity including register contents are available to the reentrant processor during its execution.

After a reentrant processor is loaded, the D bank of the main program and the I bank of the reentrant processor are used for determining the main storage address limits for the reentrant processor.

10.4.4.2. RLINK$ EXECUTIVE REQUEST

Purpose:

References any reentrant processor in the SYS$*LIB$ list or the RLIST$ list.

Format:

```
L    A0,.('repname')
ER   RLINK$
```

Parameters:

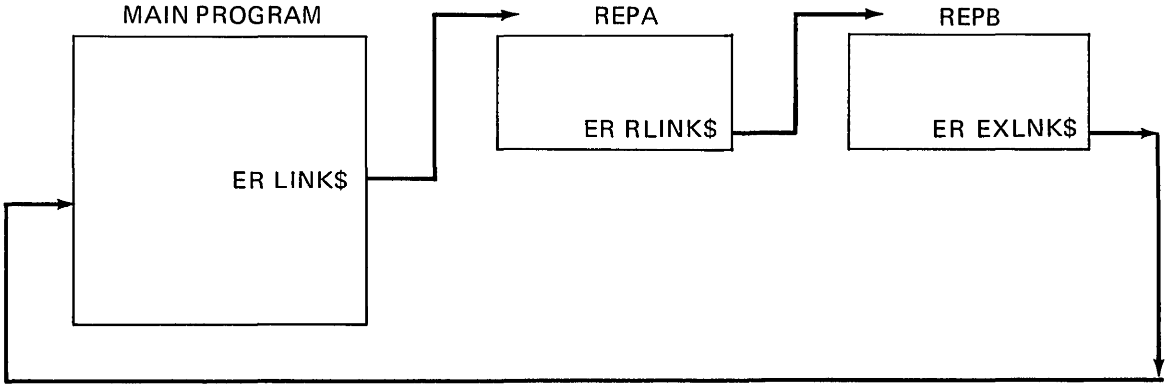
repname	Specifies a six-character reentrant processor name.

Description:

This request is the same as LINK$ (see 10.4.4.1) except that when it is used from within one reentrant processor to link to another reentrant processor, the return point saved is not back to the calling reentrant processor, but to the return point that the calling reentrant processor itself would return to.

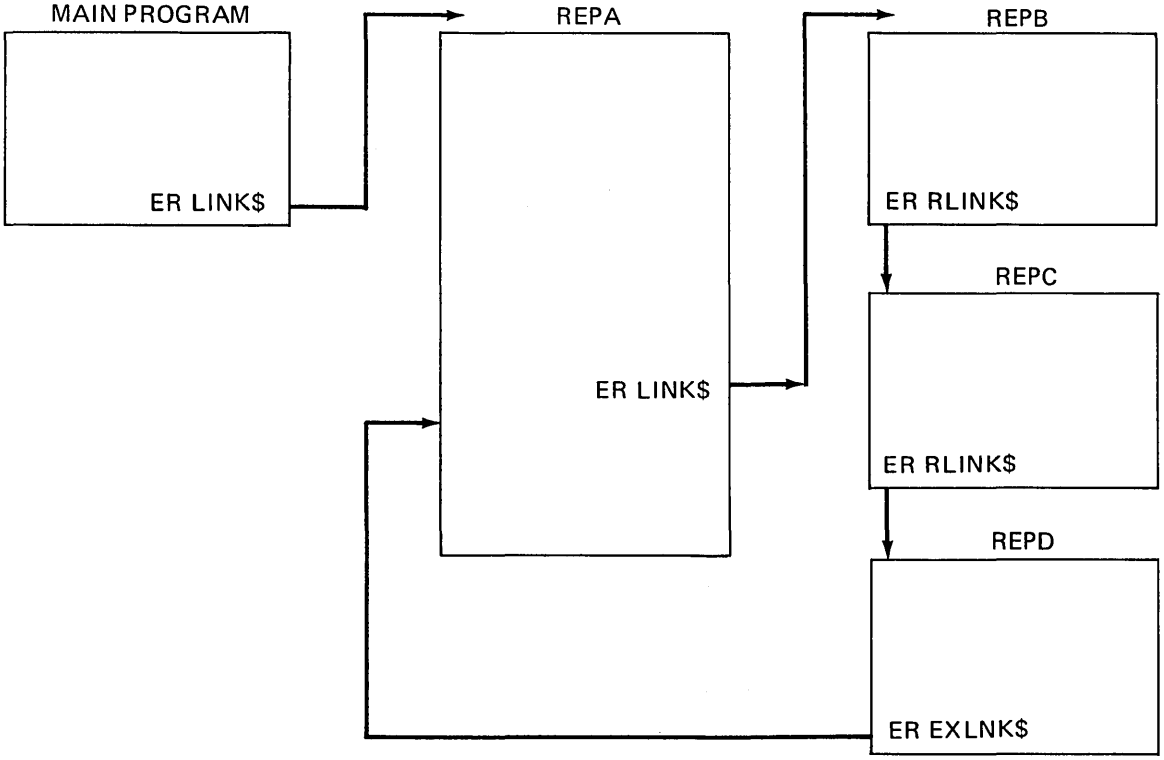
Example 1:

A main program calls reentrant processor REPA by a LINK$ request. REPA links to reentrant processor REPB by an ER RLINK$. After linkage to REPB has been established, control passes to the starting address of reentrant processor REPB; REPA is not retained by the program.



Example 2:

After linkage with REPD is established, control returns to the instruction following the LINK$ request in REPA; REPB and REPC are not retained by the program.



10.4.5. REENTRANT PROCESSOR TERMINATION (EXLNK$, UNLNK$, AND EXIT$)

To terminate the execution of a reentrant processor, three executive requests are available to the user: EXLINK$, UNLINK$, and EXIT$.

10.4.5.1. EXLNK$ EXECUTIVE REQUEST

Purpose:

Returns control to the instruction immediately following the LINK$ request.

Format:

```
ER EXLNK$
```

Description:

The contents of any control registers are unchanged after execution of the EXLNK$ request and are passed to the main program or reentrant processor that receives control.

10.4.5.2. UNLNK$ EXECUTIVE REQUEST

Purpose:

Returns control to a main program.

Format:

```
ER UNLNK$
```

Description:

Although a nested complex of reentrant processors may have been created, control is passed directly back to the main program following the original LINK$ or RLINK$ request by an UNLNK$ request.

10.4.5.3. EXIT$ EXECUTIVE REQUEST

Purpose:

Terminates the requesting activity wherever it is within a reentrant processor or nest of reentrant processors (see 4.3.2.1).

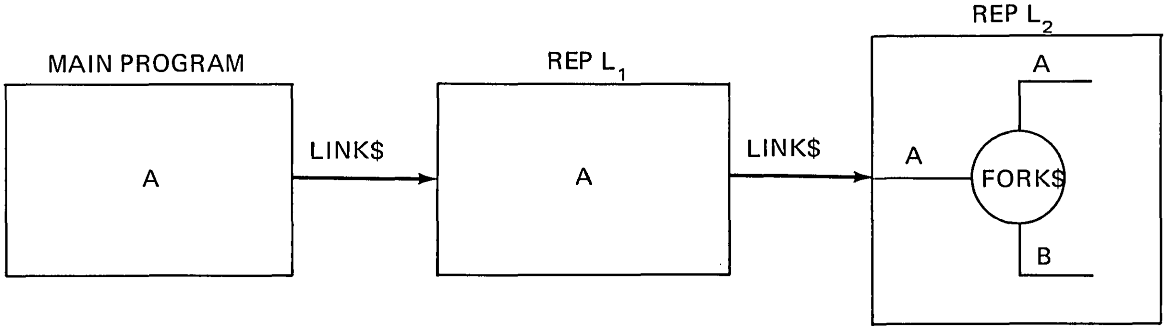
Format:

```
ER EXIT$
```

10.4.6. REENTRANT PROCESSOR FORKING

FORK$ requests (see 4.3.1.1) within a reentrant processor are allowed and the multiple activities are executed in the normal manner. The requester (creator of the new activity) and the new activity are treated as equals; no record of which activity initially entered the reentrant processor is retained. The chain of reentrant processors is made available to the newly created activity. This allows the forked activity to assume the previous path of control when an EXLNK$ or UNLNK$ request is executed.

The following illustrates a simple link-fork complex. Activity A is the original activity. $L_1, L_2, \ldots, L_n$ represents reentrant processors. Activity B is a forked activity created in the reentrant processor L_2.



(1) If both A and B perform UNLNK$ requests, both A and B return to main program.

(2) If A performs UNLNK$ and B performs EXLNK$, A returns to the main program and B returns to L_1.

(3) If A performs UNLNK$ and B performs EXIT$, A returns to the main program and B is terminated.

(4) If both A and B perform EXIT$ requests, A and B are terminated.

(5) If A performs EXIT$ and B performs UNLNK$, A is terminated and B returns to the main program.

10.4.7. REENTRANT PROCESSOR CONTROL AND RESTRICTIONS

The LINK$ or RLINK$ requests (see 10.4.4) are used to request the executive to search the various reentrant processor lists for loading the requested reentrant processor. If the specified reentrant processor is presently in main storage, control can be given immediately without the actual loading of the reentrant processor. For the frequently used standard reentrant processors, the general case does not require the actual loading of the reentrant processor. Many activities may simultaneously execute one reentrant processor.

The following restrictions are imposed by the executive's link mechanism:

- There are no rules inhibiting the use of the LINK$ and RLINK$ requests within a set of reentrant processors. For example, the executive does not take action to inhibit a reentrant processor from requesting itself or another reentrant processor.

- From a given program, simultaneous execution of more than one REP is not allowed. While this is of no consequence in a single activity program, care must be exercised in a multi-activity program to prevent attempting concurrent execution of two or more REPs.

- An MCORE$ I bank request (see 4.7.1) is not permitted from a reentrant processor.

- Uniqueness of reentrant processor names is the responsibility of the user; the executive does not ensure uniqueness for names provided in SYS$*LIB$ or those names entered by an RLIST$ request. The executive assumes the first name found is the requested reentrant processor.

When an error contingency occurs while executing a reentrant processor, the executive has three alternatives:

(1) If the program or activity has registered a contingency with the executive and the contingency routine resides in the D bank of the main program, control is given at this location.

(2) If the program or activity has registered a contingency with the executive and the contingency routine is in the I bank of the main program, the executive's standard action is taken (see 4.9.2.1).

(3) If the program or activity has not registered a contingency, the executive standard action is taken (see 4.9.2.1).

A list of error and contingency types (type 10_8) are provided in Appendix C.

10.5. REENTRANT PROCESSOR PREPARATION

The following paragraphs describe programming considerations for reentrant processors (REP). This is not a complete discussion but provides guidelines for reentrant processor generation.

10.5.1. USAGE OF A REENTRANT PROCESSOR

One function of reentrant processors (REP) is to provide for the dynamic loading of additional code. This function is somewhat analogous to loading a segment but has a much higher overhead. Another use is to provide code which can be shared, through simultaneous use, by many independent runs.

10.5.1.1. COMMON I BANKS

Reentrant processors may provide considerable savings in total main storage requirements and thus increase throughput when commonly used system processors are written as REP's. Several items must be taken into consideration before deciding to write a processor as a REP.

(1) Is the processor used frequently enough that the probabilities are that more than one run will use it each time it is loaded? The REP is not loaded as part of the initial program and the use by a single run results in more overhead than if the processor was collected as a conventional program.

(2) Is the execution time long enough to justify the time spent in processing the LINK$ and EXLNK$ requests? It is not justifiable to create a REP which is a collection of small standard library routines.

(3) Does the processor have a large instruction (I) bank and is it frequently used? If so, then coding it as a REP should definitely be considered.

10.5.1.2. ADDITIONAL INSTRUCTION SPACE

Some programs run for long times and have only occasional need for additional instruction space. One such class of programs is real time programs which have periodic or data-dependent requirements for additional instruction space. A REP may be desirable in the following cases:

(1) If the program has multiple activities but not all the activities wish to use the new space, a REP provides a means of dynamically allocating additional instruction space without affecting the D bank addressing range. If this condition is not met, the use of MCORE$, LOAD$, and LCORE$ requests provide a lower overhead mechanism for achieving the same results.

(2) If the program is real time, expansion by MCORE$ requests cannot be guaranteed. Thus,'a REP provides a means of acquiring the space wherever it is available within main storage. This may also result in less fragmentation of main storage than having the real time program collected at its maximum size and then executing an LCORE$ request immediately after loading.

(3) For nonreal time programs, segmenting should always be used instead of REP's unless parallel processing can be achieved by having some activities use the REP while others use the main program I bank.

(4) For this usage of REP's, data addresses must be passed in registers unless the main program's I and D bank code is in separate elements.

10.5.2. STORAGE ALLOCATION AND REENTRANCY

A REP is an absolute element consisting only of an I bank for which the executive dynamically allocates main storage upon request. The LINK$, RLINK$, and EXLNK$ requests result in a REP being loaded if it is not already in main storage. The requesting activity's PSR and storage limits are then changed to use the REP as the I bank. The requestor's D bank base and limits are not changed, which allows the REP to reference the D bank.

The executive system assumes that REP's never modify themselves. Thus, when time-sharing or other considerations dictate a swap out, the REP's space is simply released and a fresh copy is loaded at program reload time. General concepts of coding for multiple activity execution require that any such modification be protected with a Test and Set (TS) instruction. REP's may not modify themselves even with TS protection as the REP may be reloaded at any time. Thus, instruction modification, calling sequence modification and executive request packets which are modified in the REP as well as the use of a Store Location and Jump (SLJ) instructions to locations within the REP are prohibited.

There is no reentrancy requirement imposed by the executive on the D banks of activities which link to REP's. If more than one activity of the same program links to the REP, data must be protected by TS instructions or by specifying separate data spaces for each activity.

10.5.3. WRITE PROTECT MODE

REP's which are included in SYS$*LIB$ for general use should be protected against being overwritten by unauthorized or accidental linkages from activities with the wrong format D bank. For this reason, system configuration statements allow such REP's to be placed in write protect mode when they are referenced. In this mode, any attempt to store into the REP results in a guard mode interrupt. In this mode of operation, however, print images, I/O images, and executive request packets cannot be located within the REP because they would fail the storage limits checks.

10.5.4. D BANK ADDRESSING

As stated previously, a REP may reference the D bank of the calling activity. However, some means must be provided for the REP to determine the addresses of the desired data and instructions. Several linkages are available which may be used independently or jointly to obtain the desired results.

10.5.4.1. COLLECTION

The collector V and Y options (see 10.2.1) are provided to aid in the formation of REP's. With these options, I and D bank code may be collected together so that all absolute addresses are assigned and then absolute elements can be generated consisting of only I bank or only D bank code. The D bank absolute element may then be called by an @XQT or processor control statement and may, in turn, call the I bank absolute element by an LINK$ request. To generate a REP using the collector:

(1) Collect all elements containing I bank and D bank code.

(2) Use a @MAP, V control statement (see 10.2.1) to generate an absolute element containing only I bank code but with D bank addresses allocated. If necessary, use the ENT collector directive (see 10.2.2.6) to specify the starting address for the absolute element. This address receives control when a LINK$ request is executed.

(3) Use a @MAP, Y control statement to generate an absolute element containing only D bank code. If necessary, use the ENT collector directive to specify the starting address for the absolute element. This address receives control when a @XQT control statement is processed.

Sometimes it is desirable for more than one REP to be collected with the same D bank. This allows direct referencing of the D bank within a nested or sequential complex of REP's. Several requirements must be observed in collecting REP's in this manner.

(1) All D bank code must be in elements containing only even location counters or in elements for which copies of the I bank code are desired in all the REP's.

(2) All elements containing D bank code must appear on IN collector directives, including SYS$*RLIB$. elements. These elements must appear in exactly the same order in all collections. This is the only way to ensure that the same addresses are assigned to all D bank code in each collection.

(3) If the I banks are of varying sizes and any one has a highest address greater than 37777_8, use a SETMIN directive in the first D bank element included in the collection to force an identical starting D bank address for all collections.

(4) Use a @MAP,V control statement with each collection to form the REP. Use the ENT collector directive where necessary.

(5) Use a @MAP, Y control statement with any one of the collections to form the D bank element.

10.5.4.2. REGISTER BASING

A REP need not be collected with the D bank in order to reference it. In fact, a REP may be written so that it is independent of the overall structure and content of the D bank. In this case, all required D bank addresses must be loaded into registers by the calling activity. The REP must then reference the D bank through an index register plus offset.

An extention of this technique is to have the calling activity pass one D bank address in a register with this address being the location of a table, calling sequence, of addresses and values.

10.5.4.3. COLLECTOR PRODUCED TABLES

The collector produces tables of program information upon request. These tables, COMMN$, ENTRY$, and XREF$, equate program identifiers to absolute addresses. The addresses of these tables may be passed to the REP which uses them to locate data and code in the D bank. See 10.2.4.8 for the format of these tables.

10.5.5. REP SIZE

The highest address within a REP must be less than lowest address to be referenced in any D bank which attaches to the REP. The collector initially attempts to allocate D banks at 40000_8 or beyond the end of the I bank, whichever is larger. If the REP is mapped with the D bank, there should be no problem with addressing conflicts. If the D bank is not mapped with the REP and the highest REP address is greater than 37777_8, the minimum D bank address must be increased. This may be accomplished by specifically listing all elements to be included on IN statements with an assembler SETMIN directive in the first element.

10.5.6. EXECUTIVE REQUESTS WITHIN REENTRANT PROCESSORS

In general, all executive requests permitted to any user program are permitted from within or while attached to REP$. However, the restrictions described in the following paragraphs do apply.

10.5.6.1. MCORE$ AND LCORE$ USAGE

The MCORE$ and LCORE$ requests may only be used to change the size of the main program D bank. Changing the REP's size is not permitted.

10.5.6.2. IALL$ USAGE

Any contingency routine in effect at the time of the LINK$ request remains in effect when the REP is given control. If desired, the REP may use the IALL$ request to establish a new activity or program contingency routine. At the time that a contingency interrupt occurs, however, the executive takes standard action if the activity is attached to a REP and the contingency routine address is in the I bank.

10.5.6.3. CMS$ AND CPOOL$ USAGE

A CMS$ request may be executed while attached to a REP but the main program base and limits are retrieved for use with all ESI completion activities. Thus, it is not possible to have ESI completion code within a REP. The CPOOL$ request also causes the retrieval of the main program base and limits for use with ESI completion activities.

10.5.6.4. LOAD$ USAGE

The LOAD$ request may be used only for segments which consist entirely of D bank code. This includes RSEG's for which a D bank load address is specified. Note that indirect loading cannot be done as this requires a jump to an I bank entry point in the segment.

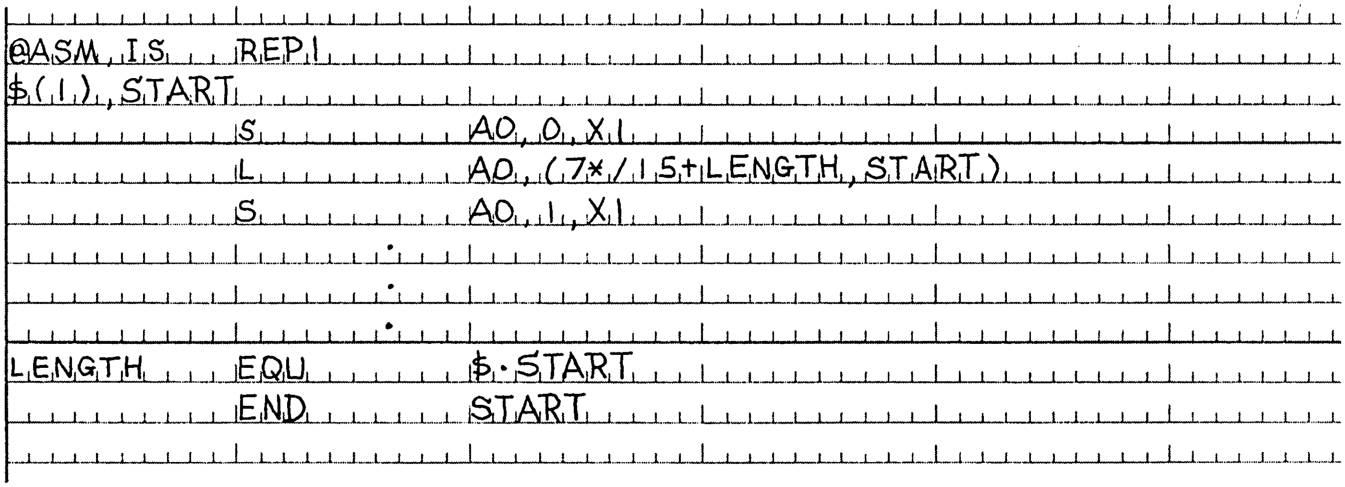
10.5.6.5. RLINK$ USAGE

The RLINK$ request from one REP to another performs a function logically equivalent to the sequence: LINK$, EXLNK$. However, a great deal of overhead is removed by using the RLINK$ request. On each LINK$ request, the executive obtains PCT space to save the return address and the name of the calling REP. In a complex or looped REP structure, this could add up to enough space to terminate a run with a PCT overflow. In addition, the result of an EXLNK$ request causes the previous REP to be loaded again so that control can be returned to it. If this REP then just performs an EXLNK$ request, the cost of loading the REP cannot be justified. Thus, the RLINK$ request should be used instead of a LINK$ request in all cases where inline return of control to the REP is not required. The RLINK$ request may be used to link from the main program to a REP, but in this case the results are identical to LINK$.

10.5.7. DUMPING REENTRANT PROCESSORS

The executive does not provide the capability to obtain postmortem dumps of REP's through the use of the PMD processor. The programmer must provide his own dumps by using SNAP$ or the XDIAG$ library routines. The following example illustrates the use of a contingency routine and SNAP$ to obtain a snapshot dump of the attached REP upon the occurrence of any error mode contingency. This example assumes the existence of several REP's any of which may be attached at the time of an error.

```
     LABEL     OPERATION          OPERAND            COMMENTS
1         10             20        30         40         50

     @ASM,IS   MAIN
     $(1),START
1.             L         AO,(0100000,CONTIN)
               ER        IALL$
               L,U       X1,SNPKT
                     .
                     .
                     .
2.             L         AO,('REP1   ')
               ER
                     .
                     .
                     .
     $(0),CONTIN
               RES       2
3.             S         AO,SNPKT+2
               L,U       AO,SNPKT
               ER        SNAP$
                     .
                     .
                     .
4.   SNPKT     RES       3
                     .
                     .
                     .
               END       START
```



5.

```
@ASM,IS   REP1
$(1),START
          S         A0,0,X1
          L         A0,(7*/15+LENGTH,START)
          S         A0,1,X1
                .
                .
                .
```

6.

```
LENGTH    EQU       $-START
          END       START
```

The code in each group performs the following functions:

(1) Establish ERR$ contingency and load dedicated register X1 with the address of the SNAP$ packet.

(2) Attach to REP1 through the LINK$ request.

(3) Define the contingency routine in the D bank. Take a snapshot dump of the attached REP.

(4) SNAP$ packet also in D bank.

(5) Store REP name in first word of SNAP$ packet. Set REP starting address, length, and register mask in second word of SNAP$ packet.

(6) Define length of REP.

11. POSTMORTEM AND DYNAMIC DUMPING

11.1. INTRODUCTION

The operating system provides a comprehensive set of diagnostic routines to aid in the checkout and debugging of user programs. The routines provided are:

- Postmortem Dump (PMD) Processor
- Dynamic Dump Routines
- Program Trace Routine (SNOOPY)

Another diagnostic capability, which produces snapshot dumps, is provided by the SNAP$ request (see 4.8.5) and the collector's SNAP directive (see 10.2.2.10). Snapshot dump output, like program trace (SNOOPY) output, is placed in the user's print file at the point at which the request was made. The output of the dynamic dump routines is handled by the PMD processor; the output of the PMD processor is listed after the print output generated by the user's program.

A diagnostic file (DIAG$), which is used for recording diagnostic information for the PMD processor, is automatically assigned to each run by the system. This file is divided into two functional parts: a dynamic dump portion and a PMD portion. The dynamic dump portion always starts at the beginning of the diagnostic file and it is followed by the PMD portion.

The dynamic dumps consist of dumps of main storage, control registers, magnetic tape files, and items written or read by the I/O handlers. The postmortem dumps consist of dumps of the final contents of main storage taken at program termination.

11.2. POSTMORTEM DUMP PROCESSOR (PMD)

The postmortem dump processor (PMD) is called by the @PMD control statement. At program termination, the system writes the final contents of the program's main storage area into the diagnostic file. The information can then be edited and printed by the PMD processor. Postmortem dumps may be taken of overlay segments, elements, or any portion of the terminated program as long as those segments, elements, and portions are in main storage when the program terminates.

11.2.1. @PMD CONTROL STATEMENT

Purpose:

Calls the PMD processor to dump all or specified portions of a program that resides in main storage at program termination.

There are two types of options: general and special. The type of @PMD control statement used depends on the type of option specified. If no options are specified or if no special options are specified, use format 1. When any of the special options are specified, use format 2.

If no parameters are specified in the operand fields, all elements residing in main storage at program termination are dumped in accordance with the specified options (applicable to both formats).

All parameters in the @PMD control statement are optional except @ and PMD.

Format 1:

@label:PMD,options eltname-1,addr/loc,length,format

Format 2:

@label:PMD,options name-1,name-2,...,name-n

Parameters:

options
The general options are described in Table 11–1 and the special options are described in Table 11–2. The special options may be used alone or in conjunction with the general options.

eltname
Specifies the element to be dumped. The names of labeled common blocks or BLANK$COMMON may also be used. Common blocks can be considered to have one location counter.

names
Specifies the element or segment to be dumped.

addr
The address, relative to the beginning of the location counter (loc), at which the dump should begin.

loc
Specifies the location counter of the element to be dumped.

length
Specifies the number of words to be dumped. If omitted, the word length is the value in the location counter of the specified element.

format
Specifies a single letter enclosed in quotation marks which references one of the standard editing formats (see 11.3.1.8.1) or a user-defined (see 11.3.1.8.2) editing format. Note that D and S standard formats and user-defined formats are not applicable for changed word dumps. If this parameter is omitted, an octal dump is produced. If none of the standard formats are desired, this field may be coded with a FORTRAN format statement enclosed in parenthesis.

Option Character	Description
C	Dumps the words which were changed during the execution of the allocated program area of main storage specified in the @PMD control statement.
E	Processes @PMD control statement only if the previous routine terminates in error.
P	Causes an octal dump of the PCT blocks used by the run to be printed preceding the dump of the program. Also the segment load tables, if any, are dumped in octal.

Table 11–1. @PMD Control Statement, General Options

Option Character	Description
A	Produces a dump of the specified main storage area of each named element or segment.
D	Produces a dump of the D-bank portion of each named element or segment.
I	Produces a dump of the I-bank portion of each named element or segment.
L	Dumps the active library elements. When used with the A, I, or D options, dumps any active system library elements.
X	Used in conjunction with the A, I, or D options. Dump all active elements except those named in the control statement and those belonging to the segments named in the control statement.

Table 11–2. @PMD Control Statement, Special Options

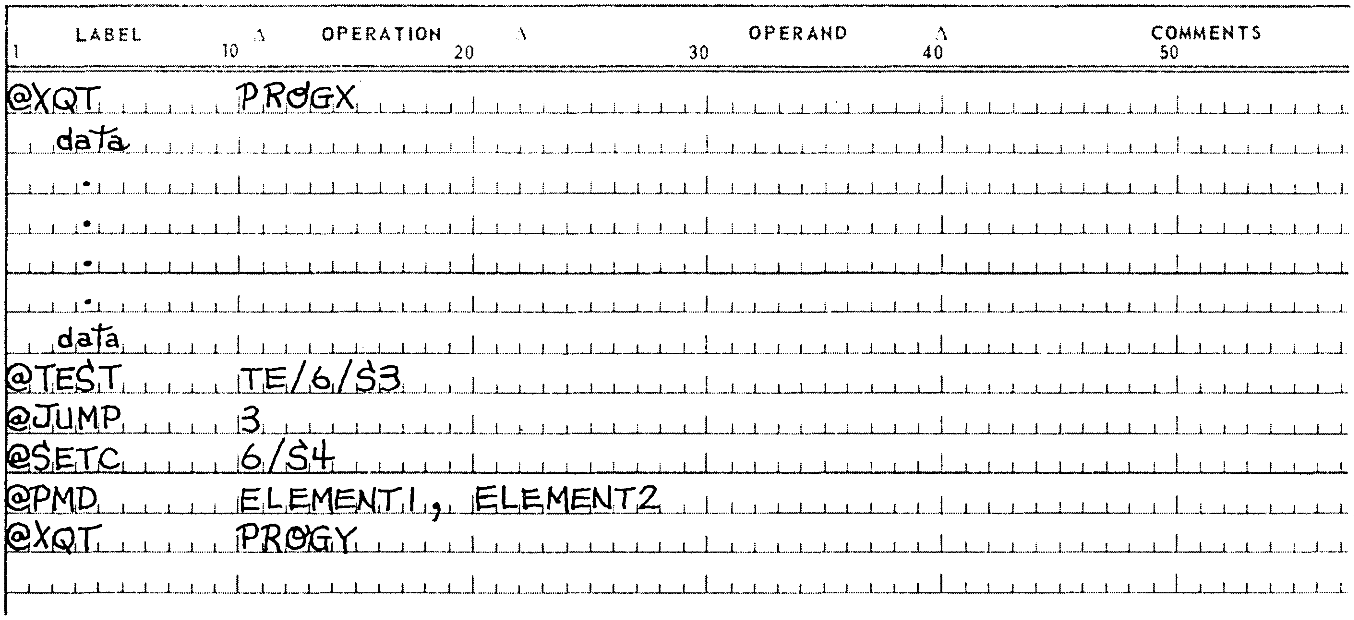
Description:

For the A, I, D, and X options, the names of labeled common blocks or BLANK$COMMON may be used as element names.

See 3.4.1 for the effect of the @RUN control statement on postmortem dumps.

If no information was saved by the system when the previous execution terminated, no dumps are possible. This condition is caused by a Z option given to the collector by the @MAP control statement. If no dump is available, a message is produced.

In order to be honored, the @PMD control statement must follow the @XQT control statement. Only data, @EOF control statement, and the conditional control statements, @SETC, @JUMP, and @TEST may intervene. For example:



```
LABEL          OPERATION              OPERAND              COMMENTS
1          10            20           30           40           50
@XQT       PROGX
  data
   .
   .
   .
   .
  data
@TEST      TE/6/S3
@JUMP      3
@SETC      6/S4
@PMD       ELEMENT1, ELEMENT2
@XQT       PROGY
```

If PROGX terminates before processing all of the data statements that follow the first @XQT control statements, and S3 of the condition word has a value of 6, S4 of the condition word is set to 6, and the @PMD and @XQT PROGY control statement are processed.

Any @PMD control statement following the execution of a program from SYS$*RLIB (that is, @FOR, @COB, @MAP, and so forth) is honored only if the Y option appeared on the @RUN control statement.

Examples:

```
   LABEL           OPERATION                   OPERAND                      COMMENTS
1             10              20              30              40              50
1. @PMD
2. @PMD,EAXL    PETER,BOB
3. @PMD,D       FLYBY
4. @PMD,ECD     REPORT
5. @PMD         ALPHA,100/3,56,A
```

1. An octal dump of all active (allocated in main storage and loaded) segments of a user's program results.

2. An octal dump of all active elements except PETER, BOB, and system library element results. The dump occurs only if the previous routine terminated because of an error.

3. Results in an octal dump of the D bank of segment FLYBY (if active).

4. Causes an octal dump of changed words in the D-bank portion of element REPORT (if active). The dump occurs only if the previous routines terminated because of an error.

5. The result of this @PMD control statement is a 56-word dump of element ALPHA in the standard alphanumeric editing format. The dump begins with relative address 100 under location counter 3.

11.3. DYNAMIC DUMPS

The dynamic dumps are discussed from the viewpoint of 1100 series assembler user. There is no inherent restriction on the employment of this facility with any other processor. All that is needed is that the proper information be written to the diagnostic file. Library routines are provided to assist in this process. The use of the dynamic dump facility by a high level language processor falls outside the scope of this manual.

Dynamic diagnostic requests are generated by procedure calls from within the user program. These procedure calls collect the dynamic diagnostic library subroutines into the user object program. The requested dynamic diagnostic information is written into the diagnostic file by the library subroutines while the object program is being executed. When called on during program execution, these subroutines preserve the complete program environment and perform the requested dynamic diagnostic request.

The amount of information which can be written into the dynamic diagnostic portion of the diagnostic file during each program execution is limited by a parameter set at systems generation time.

When the dynamic diagnostic portion of diagnostic file is filled, a message is supplied indicating that no more dynamic diagnostics can be transferred to the diagnostic file. All subsequent dynamic diagnostic requests for this program are ignored. After program termination, the dump information is retrieved from the diagnostic file, edited, and printed.

There are 18 different library subroutines associated with the dynamic diagnostic procedures. These routines can be divided into three functional classifications:

(1) Dump Procedures: X$MESG, X$CW, X$CORE, X$DUMP, X$TAPE, X$DRUM, X$FILE, and X$CREG which are used to record data in the diagnostic file.

(2) Conditional Control Procedures: XIF, XAND, X$OR, and X$TALY which are used to determine when a given dump or series of dumps should occur.

(3) Specification Procedures: X$FRMT, X$BUF, X$MARK, X$BACK, X$ON, and X$OFF which are used to specify arbitrary print-line formats, storage space for drum and tape file dumps, deletion of recorded dumps, and control for nullifying activation of diagnostic procedures.

11.3.1. DUMP CALLING PROCEDURES

The procedures available for obtaining dynamic dumps are:

X$CORE (see 11.3.1.1)

X$DUMP (see 11.3.1.2)

X$CW (see 11.3.1.3)

X$TAPE (see 11.3.1.4)

X$DRUM (see 11.3.1.5)

X$FILE (see 11.3.1.6)

X$CREG (see 11.3.1.7)

All dynamic dump procedures are executed only if the conditional dump switch is on.

Each diagnostic routine that is part of the user's program must be processed serially.

The dynamic dump procedures save and restore all control registers as well as the carry and overflow conditions.

The dynamic dump routines are not reentrant. Not more than one activity of a program should concurrently reference these procedures.

11.3.1.1. MAIN STORAGE DUMP (X$CORE)

Purpose:

Produces a dump of the specified main storage area.

Format:

```
         SLJ     X$CORE
   N$    FORM    4,14,18
         N$      index-reg,word-count,starting-addr
         +       'format',0
```

This linkage may be generated by the procedure call:

```
X$CORE    starting-addr,word-count,'format',index-reg
```

Parameters:

starting-addr — Specifies the main storage starting location of the dump

word-count — Specifies the number of locations to be dumped (37777_8 maximum)

'format' — Specifies a single letter, enclosed in quotes, which references either a standard (see 11.3.1.8.1) or a user-defined (see 11.3.1.8.2) editing format. If omitted, an octal dump is produced.

index-reg — Specifies the index register used to modify the address specified by the starting-addr parameter. This parameter, which may be omitted or left zero, can be set to values from 1 to 15 to specify an index register from X1 through A3. The value in the index register is added to the starting-addr value to get the actual dump starting address.

Description:

The main storage dump printout is preceded by the heading: **CORE DUMP**.

Examples:

```
    LABEL      Δ  OPERATION   Δ            OPERAND     Δ             COMMENTS
   1          10            20          30           40            50
1.          X$CORE        TABLEX,100,'O',X5
2.          X$CORE        TABLEY,150
3.          SLJ           X$CORE
   N$       FORM          4, 14, 18
            N$            10, 250, WW3X
            +             'A',0
```

1. The main storage dump begins at address TABLEX as modified by index register X5. The dump is 100 words in length and is presented in standard octal format.

2. The main storage dump begins at address TABLEY, has a word length of 150, and is presented in standard octal format.

3. The main storage dump begins at address WW3X as modified by index register X10. The dump is 250 words in length and is presented in alphanumeric format.

11.3.1.2. CONTROL REGISTER AND MAIN STORAGE DUMP (X$DUMP)

Purpose:

Produces a dump of the program environment, A, X, and R registers, and main storage.

Format:

```
        SLJ     XDUMP$
N$      FORM    4,14,18
        N$      index-reg,word-count,starting-addr
        +       'format',register-code
```

This linkage may be generated by the procedure call:

```
X$DUMP   starting-addr,word-count,'format','AXR',index-reg
```

Parameters:

starting-addr — Specifies the main storage starting location. If omitted, a starting location of zero is assumed.

word-count — Specifies the number of locations to be dumped (37777_8 maximum). If omitted, a length of zero is assumed and no main storage dump is produced.

'format'

Specifies a single letter, enclosed in quotes, which references either a standard (see 11.3.1.8.1) or a user-defined (see 11.3.1.8.2) editing format. If omitted, an octal dump is produced.

'AXR'

Specifies, enclosed in quotes, one or more letters representing the A, X, and R registers. The contents of these registers are printed in octal.

index-reg

Specifies the index register used to modify the address specified by the starting-addr parameter. This parameter, which may be omitted or left zero, can be set to values from 1 to 15 to specify an index register from X1 through A3. The value in the index register is added to the starting-addr value to get the actual dump starting address.

register-code

Register codes for XDUMP$ are:

No registers	0_8
R only	200401_8
A only	200202_8
R and A	400603_8
X only	200104_8
X and R	400505_8
X and A	400306_8
A, X, and R	600707_8

Description:

The printout resulting from X$DUMP is preceded by the heading: **DUMP**

The following additional information is provided following the **DUMP** heading:

- element name
- location counter
- relative program address
- hardware fault indicators

Example:

```
     LABEL      OPERATION                OPERAND            COMMENTS
1            10            20          30          40          50
1.           X$DUMP        TABLEY,200,'I','XA',X10
2.           X$DUMP        TABLEZ,500,'A','R'
3.           X$DUMP        ,,,'R'
4.           SLJ           XDUMP$
  N$         FORM          4, 14, 18
             N$            X9, 500, BSS6
             +             '0',   0200104
```

1. The main storage dump begins at address TABLEY as modified by index register X10. The dump is 200 words in length The contents of all X and A control registers are also dumped. The dump is presented in the standard integer format.

2. The main storage dump begins at address TABLEZ and has a length of 500 words. The contents of all R control registers (except R0) are also dumped. The dump is presented in the standard alphanumeric format.

3. The contents of all R control registers (except R0) are dumped in octal format.

4. The main storage dump begins at address BSS6 as modified by index register X9. The dump is 500 words in length; the contents of the X registers are also dumped. The dump is presented in octal format.

11.3.1.3. CHANGED WORD DUMP (X$CW)

Purpose:

Produces a changed word dump of specific locations within main storage. On the first X$CW call referencing a given main storage area, a complete dump of that area is produced. On subsequent X$CW calls to the same area, only those words which were changed since the last X$CW procedure call are dumped showing the previous contents and the current contents.

Format:

```
SLJ     XCW$
+       word-length,starting-addr
+       'format',0
```

This linkage may be generated by the procedure call:

```
X$CW   starting-addr,word-length,'format'
```

Parameters:

starting-addr — Specifies the main storage starting location of the dump.

word-length — Specifies the number of locations to be dumped (37777_8 maximum).

'format' — Specifies a single letter, enclosed in quotes, which references one of the following standard editing formats: A, E, F, I, or O (see 11.3.1.8.1). Standard formats D and S and user-defined formats can not be specified. If omitted, an octal dump is produced.

Description:

The number of calls on X$CW is not limited, but only five separate areas may be dumped.

Changed word dumps, whether or not actually taken, are preceded by the following heading plus the appropriate changed-word status word message:

**CHANGED WORD CORE DUMP **

Examples:

```
     LABEL     OPERATION          OPERAND              COMMENTS
1              10                 20          30          40          50
1.             X$CW               INSERT,10,'I'
2.             X$CW               REWORD,50
3.             SLJ                XCW$
               +                  750, HTR5
               +                  'F', 0
```

1. The changed word dump begins at address INSERT. The dump is 10 words in length and is presented in standard integer format.

2. The changed word dump begins at address REWORD. The dump is 50 words in length and is presented in standard octal format.

3. The changed word dump begins at address HTR5. The dump is 750 words in length and is presented in fixed-point decimal format.

11.3.1.4. TAPE BLOCK DUMP (X$TAPE)

Purpose:

Dumps the block of magnetic tape data located just prior to the current tape position by making temporary use of a previously defined buffer initialized by the X$BUFR procedure (see 11.3.3.1). The magnetic tape is moved backward one block; the block is read; and, the number of words specified in the X$BUFR procedure are dumped.

Format:

```
SLJ    XTAPE$
+      word-count,buffer-addr
+      'format',I/O-pktaddr
```

This linkage may be generated by the procedure call:

```
X$TAPE   I/O-pktaddr,'format'
```

Parameters:

I/O-pktaddr — Specifies the address of the I/O request packet (see 6.2) for the device handler. This parameter may be the address of a file control table (FCT) as is used by block buffering and other routines, since the first six words of an FCT (13.5.1) is an I/O packet.

'format' — Specifies a single letter, enclosed in quotes, which references either a standard (see 11.3.1.8.1) or user-defined (see 11.3.1.8.2) editing format. If omitted, an octal dump is produced.

Description:

Interrecord gaps separate the blocks that are recorded on magnetic tape each time an I/O write of any size word count is done. These interrecord gaps, which serve as block separators, are not to be confused with end-of-file (EOF) marks, which are a special kind of block surrounded by interrecord gaps. The X$TAPE procedure causes a move backward to the preceding interrecord gap, then a read of everything which follows (could be one word or tens of thousands of words) into the buffer initialized by an X$BUFR procedure (see 11.3.3.1) until the next interrecord gap is encountered. When the buffer is filled, the remaining words are lost.

The X$TAPE procedure is useful for dumping a block that was just read or written. No dump occurs if the magnetic tape is positioned at the load point (beginning-of-tape marker) or at the interrecord gap following an EOF mark.

No magnetic tape dump occurs if a main storage buffer is not reserved and initialized for the X$TAPE procedure.

The same buffer area can be used for both X$DRUM (see 11.3.1.5) and X$TAPE procedure calls.

The word count and buffer address are returned by the X$TAPE procedure to the first parameter word.

The tape dump printout is preceded by the heading:

```
* TAPE DUMP

** FILE filename
```

Example:

```
   LABEL         OPERATION               OPERAND                  COMMENTS
1            10            20            30            40            50
1.           X$BUFR        ALPHA,150
             X$TAPE        FILEA,'Ø'
2.           SLJ           XTAPE$
             +             800,  BUF8
             +             'Ø',  NP16
```

1. The block of data prior to the present magnetic tape position is read into the main storage location ALPHA (previously initialized by the X$BUFR procedure call) and is printed in standard octal format. FILEA specifies the I/O packet address.

2. The block of data prior to the present magnetic tape position is read into the main storage location BUF8 and is printed in standard octal format. NP16 specifies the I/O packet address.

11.3.1.5. MASS STORAGE DUMP (X$DRUM)

Purpose:

Dumps portions of FASTRAND-formatted mass storage by making temporary use of a previously defined buffer initialized by the X$BUFR procedure (see 11.3.3.1). Portions of mass storage to be dumped are read into the buffer, then the contents of the buffer are written into the diagnostic file.

Format:

```
SLJ     XDRUM$
+       word-count,location-word-addr
+       'format',I/O-pktaddr
```

This linkage may be generated by the procedure call:

```
X$DRUM    I/O-pktaddr,location-word-addr,word-count,'format'
```

Parameters:

I/O-pktaddr
Specifies the address of the I/O packet containing the internal filename (see 6.2).

location-word-addr
Specifies the address of a word which contains the relative starting sector address of the file to be dumped. (In some cases, this address may be I/O-pktaddr+5, which contains a sector address.)

word-count
Specifies the number of locations to be dumped.

'format'
Specifies a single letter, enclosed in quotes, which references either a standard (see 11.3.1.8.1) or a user-defined (see 11.3.1.8.2) editing format. If omitted, an octal dump is produced.

Description:

The mass storage dump printout is preceded by the heading:

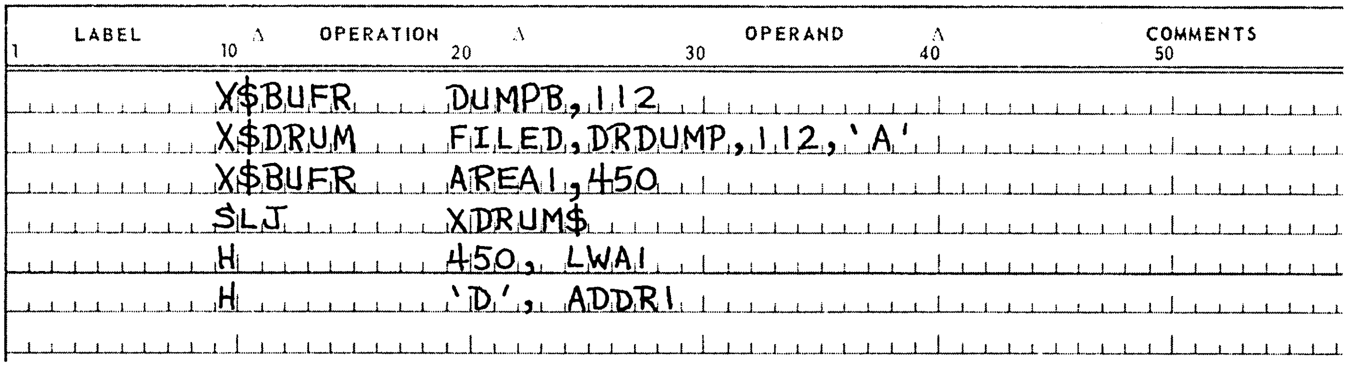
DRUM DUMP FILE filename AT RELATIVE SECTOR sector-number

Use of the X$DRUM procedure requires a main storage buffer into which the mass storage dump can be read. The mass storage area to be dumped is read into the buffer; when it is filled, the contents of the buffer are written into the diagnostic file. For FASTRAND-formatted files, it is recommended that the buffer be some multiple of 28, the length of a FASTRAND mass storage sector. While a portion of mass storage that is larger than the size of the buffer may be dumped, greater efficiency results by providing a buffer that is sufficiently large to hold all the mass storage to be dumped at one time.

If a main storage buffer is not reserved and initialized for the X$DRUM procedure, no mass storage dump occurs.

The same buffer area can be used for both X$DRUM and X$TAPE (see 13.3.1.4) procedure calls.

Example:



```
    LABEL      OPERATION      OPERAND                      COMMENTS
1              10             20             30           40          50
1.             X$BUFR         DUMPB,112
               X$DRUM         FILED,DRDUMP,112,'A'
2.             X$BUFR         AREA1,450
               SLJ            XDRUM$
               H              450, LWA1
               H              'D', ADDR1
```

1. Beginning at the relative address value specified by DRDUMP, 112 words of data from mass storage are read into the buffer starting at main storage location DUMPB that was initialized by the X$BUFR procedure call. The data is edited in standard alphanumeric format. FILED specifies the I/O packet address.

2. Beginning at the relative address value specified by LWA1, 450 words of data from mass storage are read into the buffer starting at main storage location AREA1. The data is edited in double-precision, floating-point format. ADDR1 specifies the I/O packet address.

11.3.1.6. FILE DUMP (X$FILE)

Purpose:

Provides for dumps in connection with the item handler package (see 13.4). Dynamic dumps of items can be taken whenever an item is read from or written into a particular file.

Format:

```
X$FILE   fct,option,'format'
```

Parameters:

fct — Specifies the address of the file control table (see 13.5.1).

option — The available options are:

ON — Causes subsequent items read from or written into the file to be dumped.

OFF — Terminates dumping of items from the file.

'format' — Specifies a single letter, enclosed in quotes, which references either a standard (see 11.3.1.8.1) or a user-defined (see 11.3.1.8.2) editing format. This parameter can be specified only when the ON option is specified. If omitted, an octal dump is produced.

Description:

This procedure cannot be used for an item that spans multiple blocks.

Examples:

```
     LABEL          OPERATION                 OPERAND                 COMMENTS
1                10             20          30            40           50
1.              X$FILE          BETA,'ØN','Ø'
2.              X$FILE          BETA,'ØFF'
```

1. The file whose file control table is located at BETA is conditioned to record in the diagnostic file all subsequent activity at the item level. That is, every time a request is made to an item, the item to which the item handler points is recorded in the diagnostic file and is printed in standard octal format.

2. The file whose file control block is located at BETA is conditioned not to dump any subsequent activity at the item level.

11.3.1.7. CONTROL REGISTER (USER SET) DUMP (X$CREG)

Purpose:

Dumps specified user control registers. (The A, X, and R registers and the unassigned registers at addresses 34_8 and 35_8.)

Format:

```
SLJ     XCREG$
+       register-count,starting-reg
+       'format',0
```

This linkage may be generated by the procedure call:

```
X$CREG    starting-reg,reg-count,'format'
```

Parameters:

starting-reg — Specifies the address of the first control register to be dumped.

reg-count — Specifies the number of control registers to be dumped.

'format" — Specifies a single letter, enclosed in quotes, which references either a standard (see 11.3.1.8.1) or a user-defined (see 11.3.1.8.2) editing format. If omitted, an octal dump is produced.

Description:

The register dump printout is preceded by the heading: **CREG DUMP**

Examples:

```
     LABEL          OPERATION                 OPERAND                 COMMENTS
1                10             20          30            40           50
1.              X$CREG          X1,12,'O'
2.              X$CREG          X11,10,'A'
3.              X$CREG          A14,10,'O'
```

1. Registers X1 through X11 and A0 are dumped into the diagnostic file to be edited and printed in standard octal format.

2. Control registers X11, and A0 through A8 are dumped into the diagnostic file to be edited and printed as a string of 60 alphanumeric (Fieldata) characters.

3. Control registers A14, A15, R0 through R5, and the unassigned registers 34_8 and 35_8 are dumped into the diagnostic file to be edited and printed in standard octal format.

11.3.1.8. EDITING FORMATS FOR DYNAMIC DUMPS

Each procedure for calling dynamic dumps specifies an editing format for printing the dump. Standard editing formats (see 11.3.1.8.1) are available to the user for this purpose. If, however, the user desires to define his own editing format, he must use the X$FRMT procedure (see 11.3.1.8.2).

11.3.1.8.1. STANDARD EDITING FORMATS FOR DUMPS

A number of standard editing formats are available to the user when specifying dump procedures. These formats provide the majority of printing formats desired. Table 11–3 lists the standard formats, which are specified by a single letter enclosed in quotation marks in the dump procedure calls (see 11.3.1.1 through 11.3.1.7). Figure 11–1 is an example of printouts of integer and octal dumps in standard editing format.

INSTRUCTION:

```
X$DUMP   B1,32,'I'
```

PRINTOUT:

```
**DUMP**
  CALLING ELEMENT NAME$         $( 00) RELATIVE LOCATION OF CALL   000011
  PANEL     CARRY OFF   OVERFLOW OFF

  REGISTERS

  DUMP OF ELEMENT NAME$           $( 00)   AT MAP ADDRESS   040620          CREATED ON: 20 JUL 70 AT 13:53:29
000035  040620              3           6          11          20          37          70         135         264
000045  040630            521        1034        2059        4108        8205       16398       32783       65552
000055  040640         131089      262162      524307     1048596     2097173     4194326     8388631    16777240
000065  040650       33554457    67108890   134217755   268435484   536870941  1073741854  2147483679  4294967328
```

(a) Integer Format Dump

INSTRUCTION:

```
X$DUMP   B1,32,'0'
```

PRINTOUT:

```
**DUMP**
  CALLING ELEMENT NAME$         $( 00) RELATIVE LOCATION OF CALL   000021
  PANEL     CARRY OFF   OVERFLOW OFF

  REGISTERS

  DUMP OF ELEMENT NAMES           $( 00)   AT  MAP ADDRESS  040620          CREATED ON: 20 JUL 70 at 13:53:29
000035  040620    000000000003 000000000006 000000000013 000000000024 000000000045 000000000106 000000000207 000000000410
000045  040630    000000001011 000000002012 000000004013 000000010014 000000020015 000000040016 000000100017 000000200020
000055  040640    000000400021 000001000022 000002000023 000004000024 000010000025 000020000026 000040000027 000100000030
000065  040650    000200000031 000400000032 001000000033 002000000034 004000000035 010000000036 020000000037 040000000040
```

(b) Octal Format Dump

Figure 11–1. Standard Editing Format for Integer and Octal Dumps, Sample Printout

Format Parameter	Definition	Number of Items per Line	Number of Print Positions Per Item	Number of Decimal Places
'A'	Alphanumeric	16	6	—
'D'	Double precision floating point	4	24	18
'E'	Floating decimal	8	13	8
'F'	Fixed decimal	8	12	8
'I'	Integer	8	12	––
'O'	Octal	8	12	––
'S'	Instruction	4	20	––

Table 11–3. Standard Editing Formats for Dump Printouts

11.3.1.8.2. USER-DEFINED EDITING FORMATS (X$FRMT)

Purpose:

Specifies a nonstandard editing format for use by the diagnostic dump procedure calls as an alternative to the standard editing formats described in 11.3.1.8.1, or redefines the standard editing formats. New format labels (such as 'U', 'V', or 'W', for example) may be specified, or existing standard format labels may be redefined.

Format:

```
SLJ    XFRMT$
+      format-specification-word-length,'format-label'
'format — specification'
```

This linkage may be generated by the procedure call:

```
X$FRMT   format-specification-word-length,'format-label'
'(format — specification)'
```

Parameters:

format-specification-word-length

Specifies the number of words comprising the format string. word-length

'format-label'

Specifies a single letter enclosed in quotation marks and references one of the following standard editing formats: A, D, E, I, O, or S (see 11.3.1.8.1). This action is used to redefine the standard editing formats. To specify a user-defined editing format, any letter (enclosed in quotes) except A, D, E, I, O, or S may be used.

'(format-specification)'

Specifies a string of alphanumeric characters which represent an encoding of the format to be applied to the information printed. The string of alphanumeric characters may not contain intervening blanks. The first nonblank character of the string must be a left parenthesis (preceded by a quotation mark); the last nonblank character must be a right parenthesis (followed by a quotation mark).

The format of the string of characters that comprise this parameter are specified exactly as in FORTRAN V FORMAT statements; for example, specifying '(10F8.3)' indicates that the dump information printed on one line consists of 10 words of fixed-point decimal data and that each word is eight characters long with the decimal point at the left of the third least significant character.

Description:

Any standard or used-specified editing format may be redefined; the most recent definition prevails.

Multiple line formats are allowable.

Except for the 'G' conversion code, any format that can be given in a FORTRAN V FORMAT statement can be specified. See *UNIVAC Fundamentals of FORTRAN, UP-7536* (current version).

Examples:

```
   LABEL      Δ   OPERATION    Δ            OPERAND          Δ           COMMENTS
1             10              20             30             40              50
1.             X$FRØMT     1,'Ø'
               '(6Ø14)'
2.             X$FRMT      1,'F'
               '(10F8.3)'
3.             $LJ         X1FRMT$
               +           2, 'A'
               '(12A4)'
```

1. The standard octal editing format is redefined to print six octal words per line instead of eight. The appropriate data is written into the diagnostic file so that the redefined format is effective when the diagnostic editor processes the recorded dynamic data.

2. The standard fixed decimal format is redefined to print 10 fixed decimal words instead of eight. The number of characters per word is changed to eight instead of 14, and the number of decimal places is three instead of eight.

3. The standard alphanumeric editing format is redefined to 12 words per line instead of 16 and four characters per word instead of six.

11.3.2. CONDITIONAL CONTROL PROCEDURES

The dynamic dumps can be controlled by an internal conditional dump switch. When the switch is turned off by a conditional control procedure, all dynamic dump procedures which follow are ignored until the conditional dump switch is turned on again. Note that all dump procedures are executed except when preceding conditional dump procedures cause them to be overridden. A number of miscellaneous control procedures are available to the user in addition to the conditional control procedures.

The available conditional control procedures are:

X$IF	(see 11.3.2.1)
X$OR	(see 11.3.2.2)
X$AND	(see 11.3.2.3)
X$TALY	(see 11.3.2.4)

11.3.2.1. INITIATING A STRING OF CALLS (X$IF)

Purpose:

Initiates a string of dynamic dump calls. The conditional dump switch is turned on or off depending on the value of the relational expression.

Format:

```
X$IF   addr[,index-reg] [,j-desig]   'rel'   addr[,index-reg] [,j-desig]
```

Parameters:

addr	Specifies a main storage location or a control register; indirect addressing and literals are allowed.
index-reg	Specifies an index register (X1 through X11, A0 through A3); index register incrementation is not allowed.
j-desig	Specifies any desired partial word.
'rel'	Specifies the relation between the parameters specified before and after 'rel'. Allowable codes for 'rel' are as follows:

Code	Meaning
'EQ'	Equal to
'GE'	Greater than or equal to
'GT'	Greater than
'LE'	Less than or equal to
'LT'	Less than
'NE'	Not equal to

If the relation between the tested parameters is true, the conditional dump switch is turned on; if the relation is false, the conditional dump switch is turned off.

Examples:

```
       LABEL      Δ   OPERATION      Δ            OPERAND        Δ           COMMENTS
   1               10             20             30              40             50
1.                 X$IF           ALPHA 'EQ' TAG
2.                 X$IF           ALPHA,X1,H1 'LT' TAG,X1,H1
3.                 X$IF           ALPHA,,H1 'NE' TAG,,H1
```

1. If the contents of ALPHA equal ('EQ') the contents of TAG, the conditional dump switch is turned on. If the contents of ALPHA do not equal the contents of TAG, the conditional dump switch is turned off.

2. If the contents of H1 of ALPHA, as modified by index register X1, are less than ('LT'), the contents of the H1 of TAG, as modified by index register X1, the dump switch is turned on. If the modified contents of TAG are equal to or greater than the modified contents of ALPHA, the dump switch is turned off.

3. If the relationship of the contents of the H1 portions of ALPHA and TAG is not equal ('NE'), the dump switch is turned on; if the contents of the H1 portions of ALPHA and TAG are equal, the dump switch is turned off.

11.3.2.2. LOGICAL OR CONTROL OF DUMPS (X$OR)

Purpose:

Turns on the conditional dump switch if it is already on or the current value of the realtional expression is true.

Format:

X$OR addr[,index-reg] [,j-desig] 'rel' addr[,index-reg] [,j-desig]

Parameters:

Same as X$IF procedure (see 11.3.2.1).

Examples:

	LABEL	OPERATION	OPERAND	COMMENTS
	1	10 20	30 40	50
1.		X$OR	ALPHA 'EQ' TAG	
2.		X$OR	ALPHA,X5,H2 'GT' TAG,,H2	

1. In this example, the conditional dump switch is set on or remains on if it is already on when

   □ the contents of ALPHA equals ('EQ') the contents of TAG and the conditional dump switch is on; or

   □ the contents of ALPHA equals the contents of TAG and the conditional dump switch is off.

2. The conditions for setting the conditional dump switch on are similar to those described in example 1; the condition being tested is greater than ('GT') and to turn the switch on; H2 of ALPHA as modified by index register X5 must be greater than H2 of TAG.

11.3.2.3. LOGICAL AND CONTROL OF DUMPS (X$AND)

Purpose:

Causes the conditional dump switch to remain on if, and only if, it is already on, and the current value of the relational expression is true.

Format:

X$AND addr[,index-reg] [,j-desig] 'rel' addr[,index-reg] [,j-desig]

Parameters:

Same as X$IF procedure (see 11.3.2.1).

Examples:

```
    LABEL        Λ    OPERATION      Λ              OPERAND        Λ              COMMENTS
1             10                20              30              40              50
1.           X$AND          ALPHA 'EQ' TAG
2.           X$AND          ALPHA,,T1 'LE' TAG,X10,T1
```

1. The conditional dump switch remains on if it is already on and the contents of ALPHA equal the contents of TAG.

2. The conditions for the conditional dump switch remaining on are similar to those described in example 1; the difference is that to remain on, T1 of ALPHA must be less than or equal to ('LE') T1 of TAG as modified by index register X10.

11.3.2.4. CONTROLLING THE CONDITIONAL DUMP SWITCH (X$TALY)

Purpose:

Allows a dynamic dump procedure that is embedded in a loop to be executed only when conditions specified by the user are met. The conditional dump switch remains on when these conditions are met (if it is already on), and turned off when they are not met.

Format:

```
X$TALY   start,until,every
```

Parameters:

start — Specifies the initial or starting value of the loop.

until — Specifies the maximum number of times the loop is to be executed.

every — Specifies a value which indicates the number of times the loop is to be executed before the conditional dump switch is turned on. For example, if the user specifies a value of 100, the conditional dump switch remains on every one-hundredth time through the loop; all subsequent dynamic dump procedures in the loop are executed.

Description:

The X$TALY procedure is used to set the conditional dump switch by testing a counter. The counter is set to the value of the start parameter the first time the X$TALY procedure is executed; thereafter, each time the procedure is entered for execution, the counter is incremented by one following all tests by the procedure. The tests performed on the counter (represented by the symbol Z are:

- if start $\leq$(Z)$<$until;
- if $\frac{(Z) - \text{start}}{\text{every}}$ yields a zero remainder; and
- If the conditional dump switch is already on;

the conditional dump switch remains on. If any of the tests fail, the switch is turned off.

The user should ensure that the conditional dump switch is on when an X$TALY procedure is entered; otherwise, the counter is not incremented and control is returned to the user program.

Example:

```
    LABEL          OPERATION                  OPERAND                    COMMENTS
1            10              20                      30                       40                    50
         X$TALY    0,4000,100
```

In this example, 0 indicates the start of the loop, which is to be executed 4000 times. Every one-hundreth time through the loop, up to 3999, the conditional dump switch remains on, provided it is on prior to execution of the X$TALY procedure. All other times the dump switch is turned off. Thus, we obtain any specified dumps each 100th time through the loop.

11.3.3. SPECIFICATION PROCEDURES

A number of procedures in addition to conditional control procedures (see 11.3.2) are available to allow user control of dumps. These procedures are:

X$BUFR (see 11.3.3.1)

X$ON and X$OFF (see 11.3.3.2)

X$MARK and X$BACK (see 11.3.3.3)

X$MESG (see 11.3.3.4)

11.3.3.1. INITIALIZING A BUFFER (X$BUFR)

Purpose:

Initializes an area of main storage for use as a buffer by the X$TAPE or X$DRUM procedures (see 11.3.1.4 and 11.3.1.5, respectively).

Format:

```
SLJ    XBUFR$
+      word-count,starting-addr
```

This linkage may be generated by the procedure call:

```
X$BUFR   starting-addr,word-count
```

Parameters:

starting-addr	Specifies the starting main storage address of the buffer.
word-count	Specifies the number of locations in the buffer to be initialized.

Description:

X$BUFR does not reserve a buffer area in main storage; it only initializes the area. The buffer must be defined and initialized prior to executing any X$TAPE or X$DRUM procedure.

For dumps of FASTRAND-formatted files, it is recommended that the buffer be some multiple of 28, the length of a FASTRAND mass storage sector.

Example:

LABEL	OPERATION	OPERAND	COMMENTS
	X$BUFR	TDUMP,56	

TDUMP is the main storage buffer area, 56 words in length, which is initialized for use by an X$TAPE or X$DRUM procedure.

11.3.3.2. ALLOWING AND IGNORING DUMP PROCEDURE CALLS (X$ON and X$OFF)

Purpose:

Allows overall control of the execution of dynamic dump procedure calls.

Format 1:

```
S       A0,XSTAT$
TNZ     XSTAT$
SN,H2   A0,XSTAT$
```

This linkage may be generated by the procedure call:

```
X$ON
```

Format 2:

```
SZ   XSTAT$
```

This linkage may be generated by the procedure call:

```
X$OFF
```

Description:

The word XSTAT$, which is in the XCOMN$ data area, is initially set to nonzero. If it should become desirable for all dynamic dump subroutines to return control immediately without processing any dumps, XSTAT$ may be cleared to zero by either the X$OFF procedure or the SZ instruction. To return XSTAT$ to its original nonzero status, the X$ON procedure or the equivalent three instructions may be used.

The X$OFF procedure turns off the conditional dump switch until an X$ON procedure is executed. When the X$OFF procedure is executed, the setting of the conditional dump switch cannot be changed until the X$ON procedure is encountered. Thus the X$OFF procedure causes dynamic dump and conditional procedure calls to be ignored, and the X$ON procedure allows the calls to be executed.

Care must be taken if dynamic dump procedures are used in programs consisting of independent activities and when used in I/O completion routines.

A series of dynamic dump procedure calls will not be interrupted by one of the other subprograms.

Examples:

```
     LABEL        Δ    OPERATION      Δ              OPERAND       Δ              COMMENTS
1                10                  20                  30              40                50
1.               X$OFF
2.               X$IF          ALPHA 'EQ' TAG
3.               X$CORE        ALPHA,200,'E'
4.               X$ON
5.               X$CORE        TAG,150,'I'
```

The X$OFF procedure indicates that all diagnostic system dump procedures which follow, except the X$ON procedure, are to be ignored; therefore, the X$IF and X$CORE procedures (line 3) are not executed. The X$ON procedure indicates that all subsequent diagnostic system dump procedures are allowed; therefore, the X$CORE procedure (line 5) which follows is executed.

11.3.3.3. SAVING AND DELETING DYNAMIC DUMPS (X$MARK AND X$BACK)

Purpose:

Marks the points in program execution between which dynamic dumps are saved and then deleted at the user's discretion. The X$MARK and X$BACK procedures permit a user program under checkout to include dynamic dump procedures which the user may want to execute only when a routine does not terminate normally.

Format 1:

```
SLJ   XMARK$
```

This instruction may be generated by the procedure call:

```
X$MARK
```

Format 2:

```
SLJ   XBACK$
```

This instruction may be generated by the procedure call:

```
X$BACK
```

Description:

The X$MARK and X$BACK procedures behave much as left and right parentheses surrounding portions of a program which are to be dumped only if termination occurs between them.

X$MARK and X$BACK pairs may be nested to a depth of five. The total number of occurrences of X$MARK and X$BACK is unrestricted.

Examples:

```
     LABEL        Δ    OPERATION      Δ              OPERAND       Δ              COMMENTS
1                10                  20                  30              40                50
                 X$MARK
                 X$CORE        ALPHA,200,'A'
                 X$BACK
```

X$MARK saves the current location where the next write is to be made in the diagnostic file (by X$CORE). X$BACK resets the current location pointer to the value saved by the most recent X$MARK reference. The result is that all intervening dump information is overwritten by the next dump that is taken; that is, the data recorded by X$CORE is deleted.

11.3.3.4. PLACING A MESSAGE IN THE DUMP (X$MESG)

Purpose:

Permits the user to place any message he desires into the dynamic dump.

Format:

```
SLJ    XMESG$
+      word-length-of-msg,'A'
'diagnostic-msg'
```

This linkage may be generated by the procedure call:

```
X$MESG          word-length-of-msg
'diagnostic-msg'
```

Parameters:

word-length-of-msg
Specifies a number equal to the number of computer words in the message (one computer word holds six characters).

'diagnostic-msg'
Any string of alphanumeric characters enclosed in quotes and printed exactly as assembled.

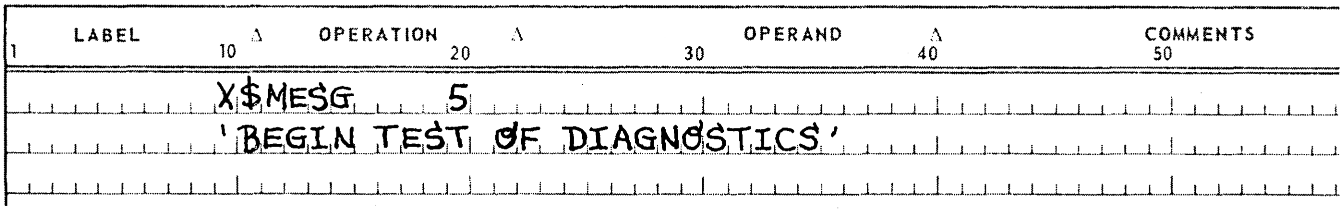
'A'
Generated by the procedure call. It is of no significance to the user, but it must be coded when the instruction form of the format is used.

Description:

The X$MESG procedure produces a line on the output listing of up to 120 alphanumeric characters. The printed line immediately follows the procedure reference.

The X$MESG procedure is executed only when the conditional dump switch is on.

Example:



```
LABEL     OPERATION      OPERAND      COMMENTS
1         10        20        30        40        50
          X$MESG    5
          'BEGIN TEST OF DIAGNOSTICS'
```

11.3.4. EXAMPLES OF DYNAMIC DUMPING

The following example indicates the effect of conditional control procedures upon dump procedures. Note that if dump procedures are interspersed with conditional control procedures,they are effective only if the conditional dump switch is set on at the time they are entered. Dump procedures have no effect on the setting of the conditional dump switch.

Assume that a program contains the variables X, Y, and Z (which have values of 78, 80, and 88, respectively), and the constants A, B, and C (which have values of Fieldata character A, 180, and 40_8, respectively). Also assume that the procedures in the following example are executed sequentially, and that they are the first group of procedures encountered.

Example:

```
                                                Conditional
                Coding                          Dump Switch     Dump Taken

(1)  $(1)
     X$MESG  7                                      ON
     ‘BEGIN TEST OF DIAGNOSTIC’                     --
     X$IF  X  ‘EQ’  A                               OFF
(2)  X$MESG  4                                      OFF
     ‘TEST DATA GROUP A’                            --
     X$IF  X  ‘EQ’  A                               OFF
     X$OR  X  ‘LT’  Z                               ON
     X$BUFR DUMPB, 150 . INITIALIZE BUFFER          --
(3)  X$CORE TABLEX, 100,‘0’                         ON             YES
(4)  X$DUMP TABLEY,200,‘I’,‘XA’                     ON             YES
(5)  X$TAPE FILEA,‘0’                               ON             YES
     X$IF  Y ‘GT’ B                                 OFF
(6)  X$TAPE FILEB,‘0’                               OFF            NO
     X$OR  Y ‘NE’ Z                                 ON
     X$BUFR ALPHA,200 . INITIALIZE BUFFER           --
(7)  X$TAPE FILEC,‘0’                               ON             YES
     X$BUFR ALPHA,112  . INITIALIZE BUFFER          --
(8)  X$DRUM FILED,DRDUMP,112,‘A’                    ON             YES
(9)  X$FILE BETA,‘OFF’                              ON             YES
(10) X$FILE BETA,‘OFF’                              ON             NO
(11) X$CREG 1,12,‘0’                                ON             YES

     $(2)   .
     DRDUMP   + 0    .  VALUE SET DYNAMICALLY BY USER
     ALPHA    RES  200 .
     BETA    (FILE CONTROL BLOCK - READ MODE)
     TABLEY   RES  200
     TABLEX   RES  100
       DUMPB   RES   150 .  BUFFER FOR DUMPING FROM
                         .  TAPE AND DRUM
     FILEA (EXEC I/O PACKET)
     FILEB (EXEC I/O PACKET)
     FILEC (EXEC I/O PACKET)
     FILED (EXEC I/O PACKET)
```

Explanation:

(1) The message BEGIN TEST OF DIAGNOSTICS is recorded in the diagnostic file because the conditional dump switch is on.

(2) The message TEST DATA GROUP A is not recorded in the diagnostic file because the conditional dump switch is off.

(3) Starting with location TABLEX, 100 words of main storage are dumped in the diagnostic file. If printed, the standard octal format is presented.

(4) The environment data, control registers X and A, and main storage locations starting with TABLEY through TABLEY + 199 are recorded in the diagnostic file. If printed, the environment data is printed as to status, control registers are printed always in octal format, and the 200 words of main storage, as specified by 'I', are printed in integer format.

(5) The block of data just prior to the present magnetic tape position and having an internal filename of FILEA is read into buffer DUMPB (initialized in (2) by the X$BUFR procedure), and is then recorded in the diagnostic file. If printed, the standard octal format as specified by '0' is presented.

(6) No dump is recorded; the conditional dump switch is turned off

(7) The magnetic tape data block, whose internal filename is FILEC, is moved backward one block, and then read forward one block. The block of data is read into the main storage location ALPHA, that is initialized by the X$BUFR procedure in (6). The data is then recorded in the diagnostic file and edited in standard octal format.

(8) Assume that the current content of DRDUMP has a value of 500. Beginning at relative word address 500 of mass storage file FILED, 112 words of data are read into the main storage location ALPHA (initialized by the X$BUFR procedure).

(9) The file, whose file control block is at BETA, is conditioned to record all subsequent activity at the item level in the diagnostic file. That is, every time a request is made to read an item, the item to which the item handler points is recorded in the diagnostic file.

(10) The file control block BETA is conditioned not to record any subsequent activity at the item level.

(11) Registers X1 through X11 and A0 are recorded in the diagnostic file and are edited in standard octal format for printing.

11.4. PROGRAM TRACE ROUTINE (SNOOPY)

SNOOPY is a program trace routine which is designed for use primarily with assembly-language programs. In batch mode, SNOOPY provides a straightforward account of every instruction executed and its effect. In the demand mode, SNOOPY acts as a powerful diagnostic routine affording user control over the trace operation.

Two formats are available for calling SNOOPY:

```
SLJ    SNOOPY
+      mode-bits,termination-addr   . mode-word
```

and

```
SLJ    TON$
```

When the first format is employed, tracing begins with the instruction following the mode-word. Tracing continues until the termination address (termination-addr) is reached or until another termination condition is encountered.

If bit 18 of the mode word is set, quarter-word mode is simulated by SNOOPY for checkout of quarter-word sensitive programs on machines without quarter-word hardware. Otherwise, SNOOPY uses either third- or quarter-word mode depending on the mode set in the PSR on entry. Bit 19 of the mode word may be set to suppress the solicitation of commands at the beginning and end of a trace when SNOOPY is used in demand mode.

When the second format is employed, tracing begins following the SLJ instruction and continues until a termination condition is encountered; quarter-word or third-word mode is determined by the mode set on entry.

When operating in the batch mode; tracing may be terminated by:

(1) Reaching the specified termination address (program execution continues).

(2) Executing a SLJ TOFF$ instruction (program execution continues). If an SLJ TOFF$ instruction is executed outside of the trace routine, it has no effect.

(3) Performing an EXIT$ request (see 4.3.2.1). This not only terminates SNOOPY but it also terminates the activity being traced.

(4) Encountering a program contingency of types 1_8, 2_8, 7_8, or 12_8 for which standard system action has been specified (see 4.9). The activity being traced terminated by EXIT$.

When operating in the demand mode, tracing may be terminated by the following method in addition to those available for batch mode:

(1) Using the TOFF$ command (see Table 11–4); program execution continues.

(2) Using the EXIT$ command (see Table 11–4). This not only terminates SNOOPY but it also terminates the activity being traced.

The following restrictions apply to SNOOPY:

(1) SNOOPY must be part of the program's main segment (see 10.2.3.3).

(2) Tracing terminates if the main segment is reloaded.

(3) Only one activity may be traced at any one time unless duplicate copies of SNOOPY are used.

(4) The program cannot be running in real time mode.

(5) No activity contingency routine may exist for the activity being traced.

(6) The only I/O executive requests permitted are IO$ and IOW$.

(7) Reentrant processor executive requests are not permitted.

A program being traced functions the same as an untraced program except that:

(1) execution time is slower

(2) contingencies that would normally cause the activity being traced to terminate in error now result in EXIT$ termination

When using SNOOPY, the u field of the instruction is edited in octal if it does not refer to main storage. When the u field falls within a certain range, it is possible to provide a symbol table to SNOOPY giving Fieldata characters instead of octal values. This is done by storing a value into the externally-defined location SYMTB$. The value in SYMTB$ may be changed at any time, even while a program is being traced. The value placed in SYMTB$ has the following form:

H1	H2
nbr-of-entries	table-addr

The symbol table at the specified address is a three-word entry in the form:

		H1	H2
Word	0	high+1	low
	1	Fieldata-symbol-name	
	2	0	symbol-addr

If the u field of an instruction satisfies low ⩽ u ⩽ high, the u field is edited as the symbol given by the second word of the entry, plus or minus an offset calculated from the symbol address in the third word. If the nbr-of-entries parameter of SYMTAB$ is zero, as it is initially, all u fields outside main storage are edited in octal, except TOFF$ and a number of ER's.

For batch users, operation is as just described. Users are cautioned that large quantities of printout may be produced when using SNOOPY.

Demand users are given a great deal of control over the behavior of SNOOPY. When entered, SNOOPY examines the program control table (PCT) to compute storage limits and the contingency routine address. At this time, the program type is checked; if it is in demand, conversational control facilities are enabled.

When tracing a demand program, the amount of output produced by SNOOPY is reduced because of the low speed devices used for output. In particular, header and tailer messages are brief, registers are dumped only on request, and line length is restricted to 72 characters. Further control over printing may be obtained through the use of commands as described in Table 11–4.

For demand programs, SNOOPY operates in two modes: trace mode, in which instructions are being traced, and command mode, in which commands are accepted that direct SNOOPY's operation. SNOOPY enters command mode under these circumstances:

(1) On entrance, before tracing any instructions; except if mode bit 19 was set.

(2) When the RBK contingency occurs (BREAK key and carriage return).

(3) On completion of the number of instructions specified by a SKIP or a numeric command.

(4) When a breakpoint specified by the BREAK command is reached.

(5) At trace termination, except if mode bit 19 was set.

In command mode, commands are solicited by the typeout "C--"; parameter fields required by the commands are solicited by a typeout indicating the nature of the parameter required. More than one command may be given on any line solicited in command mode (including parameter lines). The commands and parameters are separated by delimiters, where a delimiter is any character not in the set (A–Z, 0–9, $, or –). Excess blanks are ignored; if any other consecutive delimiters are encountered, the effect is the same as the STEP command; that is, if "C--" is answered with a line consisting of "////", it has the same effect as four STEP commands. In certain cases, specific delimiters are required. To omit a parameter entirely, the delimiter which terminated the command must be followed by another (nonblank) delimiter (For example: "PRINT /"). Trace mode is suspended, the next instruction is printed, and a soliciting message is made under the following circumstances:

(1) trace termination

(2) the break sequence is used to interrupt the trace

(3) an invalid command is encountered

(4) the CHANGE command is used

In each of these cases, all commands on the line after the last one performed are ignored and can be reentered (if so desired).

A blank line (carriage return) in reply to the "C––" typeout has the same effect as the STEP command; a blank line response to a parameter request may be erroneous or may have a special meaning depending on the nature of the command.

When command mode is entered at trace termination, the trace may be completed only through use of the GO or STEP commands or a command with an equivalent effect. Once such a command is given, no further commands are executed, and the trace terminates. The activity then continues execution or exits, depending on the type of trace termination.

In all cases where a number is called for, octal notation is assumed, unless otherwise indicated; a leading zero is *not* required. In general, SNOOPY uses octal notation everywhere except in register designations, and in the instruction cycle count printed at the end of a trace.

All commands listed may be abbreviated to the first three characters; all commands except TON$, RBK, and STEP may further be abbreviated to the first character only.

Table 11–4 lists the SNOOPY commands and their functions.

Command	Description
ABSAD	Convert relative program addresses to absolute addresses. The three parameters are: eltname,loc-counter,location. If the program is segmented and the specified element is in a segment not in main storage, a message is printed. This command is useful in conjunction with the CHANGE, DUMP, and JUMP commands.
BREAK	Automatically return to command mode when control reaches a specified point in the program. Only one breakpoint may exist at a time. The three parameters are: eltname,loc-counter,rel-addr The slash is used as a delimiter to indicate that the following parameters are deleted. For example: BREAK TEST/ BREAK TEST,1/ BREAK / If no parameters are given, any previously set breakpoint is deleted and no breakpoints exist. If only eltname is specified, the break occurs whenever control passes to any location in that element. If only eltname and loc-counter are given, the break occurs whenever control passes to any location is the specified element under the specified location counter. If all three parameters are specified, the break occurs only when control passes to the exact location. A break at an absolute program address is specified by preceding the relative address with the master space (@). A break at a different address within the same element and location counter as the currently executing instruction is specified by preceding the relative address with an asterisk. In neither of these cases is more than one parameter required. For example: BREAK @1435 BREAK *45 When a break is to occur, the command mode is entered before the instruction at the breakpoint is executed.
CHANGE	Allows the user to change the contents of control registers or main storage. The single parameter gives the location to be changed. After reading the location parameter, the current contents of the specified location are displayed as if the location parameter had been given to a DUMP command; then the new value is solicited. The new value is stored and the new contents displayed. A void new value results in no change to the indicated main storage element. If the parameter is a register name, a number, or a number preceded by an H, the new number is to be entered as a single octal number. If the parameter is a number preceded by an I, the new value to be entered consists of six numbers representing the f, j, a, x, hi, and u fields of an instruction word.

Table 11–4. Demand Mode Commands (Part 1 of 3)

Command	Description
DUMP	Display the program status. Each parameter must be separated by a comma. If no parameter or an empty parameter (that is, two consecutive commas) is given, all registers and the carry and overflow designators are dumped. The parameters are: A,X, or R – Dumps the indicated group of registers. To dump the contents of a single register, use the register mnemonic or the octal address. T – Dumps the carry and overflow designators. If a specification number is neither the address of a register, nor within the storage limits of the program, an error message is displayed. The contents of the main storage area are printed in octal. The letter I preceding a number produces an instruction-format dump. The letter H preceding a number produces a Fieldata – character dump. NOTE: The combination, HA0 is invalid; use H14.
EXIT$	Terminates the traced activity by means of an EXIT$ request. Trace mode is terminated and the last instruction is printed.
GO	Return to trace mode from command mode.
HELP	Prints a brief set of directions for use in command mode. This printout may be interrupted by the BREAK key. The first time an invalid command is encountered by SNOOPY, the HELP printout is given.
JUMP	Transfers control to a specified absolute address. The current absolute and relative P register count is displayed. If the new value is within the program storage limits, that value is set into the P register. The new value is printed in relative form and the next command is executed. This command is the only way to get out of EXIT$ mode and do further tracing by means of the TON$ command. If TON$ is used in the EXIT$ mode without a JUMP command, the TON$ command is rejected and a message is displayed. If a JUMP command is used in EXIT$ mode termination, the termination mode becomes a TOFF$ termination; a TON$ command is required if tracing is to continue.
number	Has the same effect as a SKIP n GO command (where n is the number). See SKIP command.

Table 11–4. Demand Mode Commands (Part 2 of 3)

Command	Description
PRINT	Allows modification of the amount of printing. The PRINT command recognizes only one parameter at any one time. If an invalid parameter, or no parameter is specified, the F parameter is assumed. The parameters are: C – Produce a printout omitting extraneous spaces used for formatting. E – Produce an expanded printout (formatting spaces are included). The E mode is effective until a PRINT C is encountered. F – Produce a full printout consisting of each instruction, its location, and the contents of main storage and registers (in before/after form if the value changed). For certain executive requests, the contents of the associated packet is also dumped. N – Suppresses printout. This provides a means of skipping long sections of irrelevant code. P – Produce printout of the instructions but not referenced main storage or executive request packets. If SNOOPY is in the N mode, the P mode is set automatically upon the occurrence of an RBK contingency or encountering a BREAK specified break condition.
RBK	Allows the user to simulate an RBK contingency for the executing program; the actual RBK contingency is intercepted by SNOOPY and directs a return to command mode. This command provides the means for tracing a contingency routine. If the user program does not expect the contingency, an appropriate message is displayed.
RELAD	Convert absolute program addresses to relative addresses. The only parameter is: location. Ambiguities are resolved in favor of elements residing in main storage.
SKIP n	Return to command mode after executing n number of instruction cycles. If n is omitted, any previously existing skip count is deleted and no skip interrupt occurs. Otherwise, an octal number is used to set the interrupt point. If the count is exceeded during an indirect addressing or execute remote cascade, the command mode is reentered when the instruction is completed.
STEP	Execute one instruction in trace mode and return to command mode.
TOFF$	Leave the trace mode and continue execution as if an SLJ TOFF$ command had been executed. Trace made is terminated and last instruction is printed.
TON$	Restart a trace that was to be terminated and execute one instruction. To compute the number of instruction cycles performed, use the TOFF$ command followed by the TON$ command. The TON$ command is not effected if the activity is about to terminate by means of an EXIT$ request; if it is desired to continue tracing from that point, a JUMP command must first establish a point from which execution will continue.

Table 11–4. Demand Mode Commands (Part 3 of 3)

12. DEMAND PROCESSING

12.1. INTRODUCTION

Demand processing is defined as a mode of operation in which run processing is dependent on manual interface with the executive during processing. Basically, it is a conversational mode of operation requiring a demand and response type of activity. Conversational operation via a remote terminal causes the executive, a demand processor, or an active program to immediately react and respond. Demand processing terminals are generally thought of as being remote from the computer site and to have a printer or a cathode ray tube and a keyboard. An example of a demand terminal is the teletypewriter keyboard and printer.

The distinction between batch-mode processing and demand processing lies in the frequent interaction with the user that occurs during demand processing. The terminal user is considered to be in conversation with the executive, special demand function, user programs, or the batch functions of the executive on a unit basis.

Tasks executed by the demand terminal user normally have frequent but short bursts of computation. Progress is always insisted upon; however, to receive a substantial amount of computation may require a long period of time. Access to computation is a percentage of the total computing facility and is scheduled in small increments of time at frequent intervals to provide immediate responses. This action gives the appearance of total system control to the user and the impression that he is the only user currently running. The more a user is required to interact with a demand program, the shorter the bursts of computation required to service a given request. The bursts of computation are time-shared within the executive to provide an apparent immediate response, with the program placed in a dormant mode during idle periods awaiting response from the user.

While a demand program is in a dormant mode, it may be necessary to swap the program from main storage. Normally, thi: happens only when main storage is full and another program, which is currently on mass storage, has work to do.

The executive supports the use of the following terminals to access the system in the demand mode:

- UNISCOPE 100 Display Terminal
- UNISCOPE 300 Visual Communications Terminal
- UNIVAC DCT 1000 Data Communication Terminal
- UNIVAC DCT 500 Data Communications Terminal (Teletypewriter and Semi-Automatic Mode)
- Teletypewriter Models 33 and 35 (KSR and ASR)
- Friden 7100

12.1.1. GENERAL DEMAND TERMINAL OPERATIONAL PROCEDURES

The following is a brief discussion of the operational procedures for initialization of a demand terminal, submission of a demand run, termination of the demand run, and deactivation of the demand terminal.

12.1.1.1. INITIALIZATION

Before the demand terminal can be initialized, the user must turn it on, set the various switches to the proper position, and establish the proper line connection if operation is on a switched line network. A discussion of the hardware characteristics of the various terminals can be found in the appropriate programmer/operator reference manuals. The manuals describing the operational procedures for Univac equipment are:

- *UNIVAC UNISCOPE 100 Display Terminal Operator Reference Manual, UP-7788* (current version)

- *UNIVAC UNISCOPE 300 Visual Communications Terminal Operator Reference Manual, UP-7615* (current version)

- *UNIVAC DCT 500 Data Communications Terminal Operator Reference Manual, UP-7832* (current version)

- *UNIVAC DCT 1000 Data Communications Terminal Operator Reference Manual, UP-7827* (current version)

Once the connection is made and the terminal is initialized, the demand user must send a six-character remote site-id to the operating system. Each terminal is assigned a site-id, and the executive maintains a list of all valid site-id's. The site-id submitted by the demand user is compared to this list and if the system responds with the message

```
UNIVAC 1100 TIME-SHARING EXEC VER xx.xx.xx
```

the demand user can assume that the initialization operation is completed (xx.xx.xx is the version of the 1100 operating system operational at the central site). If the site-id is invalid or already in use, the site attempting to establish communications is not recognized. If the message is not received within approximately one minute, the demand user should retransmit the site-id (the previous transmission may have been received in error by the executive).

12.1.1.2. DEMAND RUN STREAM SUBMISSION

After the initialization is completed, the terminal user must submit a @RUN control statement (see 3.4.1) from the primary input device, that is, the keyboard. All run stream data must follow the submission of a @RUN control statement. The submitted data must observe the rules and formats for run streams. There are three conditions during which the system will not accept data:

(1) After submission of a @RUN control statement and before receipt of the date and time message from the operating system.

(2) After submission of a @FIN control statement (see 3.4.2) and before receipt of the accounting information from the operating system.

(3) During those times that input is backed up because of other processing in the system. This condition is indicated by the message ****WAIT**** on the output device. When the wait condition is removed, the message READY is displayed. The number of messages which can be backed up before the wait condition depends on the size of the message transmitted.

A terminal user normally terminates his run stream with a @FIN control statement, and it is advisable that he do so. The symbiont generates a @FIN control statement for the system under the following conditions:

(1) Transmission by the terminal user of an end-of-transmission sequence from a terminal while a run is active.

(2) Transmission of a new @RUN control statement without a preceding @FIN control statement being submitted for the previous run.

(3) When a lack of activity timeout occurs during an active run.

If no input has been received by the symbiont from a terminal in a prescribed interval (unique to the specific demand symbiont), the warning message TIMEOUT is sent to the terminal device. The terminal user is then allowed an additional time interval to submit input. If he fails to do so, the run is terminated and the message TIMEOUT TERMINATION is sent to the terminal. The terminal is then considered to be in the inactive state.

12.1.1.3. TERMINATION

The standard termination procedure is performed when a @FIN control statement is received by the system. The symbiont retains control of the line terminal until all output destined for the site has been sent.

The symbiont then returns to the command mode. Either another @RUN control statement or an end-of-transmission sequence should follow. The telephone connection is available for further communication until the end-of-transmission sequence is transmitted from the terminal. An end of transmission sequence leaves the telephone connection available on the UNISCOPE 100, DCT-1000, UNISCOPE 300, and semi-automatic DCT-500.

When the demand run is terminated with the end-of-transmission or by the onsite operator, the terminal is immediately released. The run is terminated with no indication of termination being sent to the terminal. Any information previously received by the symbiont and not processed is discarded. Likewise, any accumulation of output by the symbiont is also lost. If the console operator downs the subsystem and unit for a terminal, the symbiont treats the site as though it timed out.

12.1.1.4. DEMAND TERMINAL/SYSTEM INTERFACE MESSAGES

Table 12–1 lists the messages and their meanings used as aids in communicating between the system and the user.

Message	Interpretation
NO RUN ACTIVE	This message is sent to the terminal whenever an image is received from it and no run has been initiated. A @RUN control statement must then be submitted to properly initiate the demand mode.
TIMEOUT	No activity has occurred on the line for a predefined interval. If another time interval elapses without activity, the terminal is terminated and the message TIME-OUT TERMINATION is displayed at the terminal.
READY	Informs user that the symbiont is conditioned to receive input. This message is only transmitted if a ***WAIT*** had previously been sent to the terminal. The READY message is not sent to the terminal if output from the run is available. In either case, the wait condition is terminated.
WAIT	This message is sent to the terminal when: (1) An attempt is made to input from the terminal before the @RUN control statement has been completely processed (no input is accepted until the @RUN control statement is processed). The ***WAIT*** message is displayed following each character the user attempts to input. (2) An attempt is made to input from the terminal before the @FIN control statement has been completely processed (same conditions as for (1)). (3) The executive is executing programs of higher priority. (4) Both 28-word input buffers are full. The user is notified that additional input can be accepted by the READY message (no output is available) or the symbiont output to the terminal. A line image can be considered as accepted if the CR input character results in a LF/CR sequence and no ***WAIT*** is displayed.

Table 12–1. Demand Terminal Interface Messages

12.1.2. DEMAND SYMBIONT/USER INTERFACE

Several executive requests are available to a user program which desires to use the demand symbiont for two-way communications with a remote demand terminal. A comprehensive discussion of the specific requests and their format is given in Section 5. The following is a brief discussion of the standard symbiont interface.

- Input

  The user program requests input from the demand terminal by executing a READ$ request (see 5.2.1). The standard image length from a demand terminal is 72 characters. The image usually contains text data unless the user program specifically requests otherwise.

- Output

  The user program initiates data transfer to the demand terminal by executing a PRINT$ request (see 5.3.1). The spacing specifications for this request may vary depending directly on which demand symbiont performs the data transfer.

- Output/Input

  The TREAD$ request (see 5.2.5) enables the requestor to output an image and select the input with a single executive request.

- Print and Screen Control

  Special executive requests are available for print and screen control. These requests are used with the UNISCOPE 100, DCT 500 (semi-automatic), and DCT 1000.

12.1.3. EXECUTIVE LANGUAGE INTERFACE

The demand user communicates with the system through the standard executive control statements. There are a few exceptions to the interpretation and use of some control statements when operating in the demand mode. These exceptions are:

- @BRKPT Control Statement (see 3.6.2) – The @BRKPT control statement is used in the same manner as it is used for a batch run. @BRKPT PRINT$/filename puts the normal print file into an SDF-formatted file. @BRKPT PRINT$ may not be submitted from the UNISCOPE 100, UNISCOPE 300, DCT-1000, and the DCT 500 (semi-automatic) terminals.

- @CKPT and @RSTRT Control Statements (see Section 17) – Full @CKPT and @RSTRT control statements are intended for batch mode use only.

- @HDG Control Statement (see 3.6.1) – This control statement is ignored when submitted through a demand terminal except when the output is directed elsewhere by a @BRKPT control statement.

- @START Control Statement (see 3.4.3) – Any run scheduled by a @START control statement submitted through a demand terminal is scheduled as a batch run. All output generated by the run is queued to the outout device associated with the primary onsite card reader.

- @SYM Control Statement (see 3.6.3) – This control statement can be used to direct output to an onsite device or remote batch terminal, but not to a demand terminal.

12.2. GENERAL OPERATION OF THE DEMAND SYMBIONTS

Each demand terminal supported by the operating system is controlled by a specific demand symbiont. There are four demand symbionts contained in the system and they are listed below:

- Teletypewriter Demand Symbiont (see 12.2.1)
- DCT 500 (Semi-Automatic) Demand Symbiont (see 12.2.2.)
- UNISCOPE 100/DCT 1000 Demand Symbiont (see 12.2.3)
- UNISCOPE 300 Demand Symbiont (see 12.2.4)

The DCT 500 (teletypewriter model) and the Friden 7100 are described as extensions of the teletypewriter symbiont.

12.2.1. TELETYPEWRITER DEMAND SYMBIONT

The teletypewriter demand symbiont provides support for Models 33 and 35 (KSR/ASR), the DCT 500 operating in the teletypewriter mode, and the Friden 7100.

12.2.1.1. OPERATIONAL CONSIDERATIONS

Initialization, run stream submission, termination, and remote system messages are as described in 12.1.1 except:

(1) Control statements entered by a teletypewriter may use either the @ or # as the lead character to indicate that the image is a control statement.

(2) The symbiont accepts two forms of paper tape input (see 12.2.1.2).

(3) Several characters are recognized by the symbiont as control sequences (see 12.2.1.3).

(4) If a timeout occurs when a user program has a registered contingency activity, the contingency is activated and the activity is passed as a error type 2_8 and contingency type 10_8. The site's timeout process is again initialized. If no contingency is registered, the site is terminated as described in 12.1.1.

(5) If a ***PARITY ERROR*** message is displayed at the terminal, the symbiont has detected a parity error on at least one character and the entire input image is discarded.

(6) The @TABSET control statement is available to teletypewriter users as an aid in formatting input data at the teletypewriter terminal (see 12.2.1.5).

(7) A special routine for communications between the central site operators console and the teletypewriter terminal (see 12.2.1.6).

12.2.1.2. PAPER TAPE INPUT

Two forms of paper tape input are permitted; they are:

- Form I — Interactive Mode
- Form II — Continuous Mode

12.2.1.2.1. FORM I PAPER TAPE INPUT

Images on paper tape consist of a string of up to 80 characters followed by the character sequence:

LF X-OFF CR DEL

where:

LF is line feed

CR is carriage return

DEL is delete (or rub out)

The DEL may not be required depending upon the teletypewriter model (experimentation may be required).

In the tape mode, all images must be in this format, or the results are unpredictable.

The paper tape mode is initiated by inserting a tape in the reader, sending the character X–ON (control Q) and on the ASR–35 models, switching from keyboard to tape mode. This causes the symbiont to send an X–ON back to the teletypewriter which then reads one image. After the end-of-image sequence is received, any available output is sent. When the symbiont is ready to accept another image, an X–ON is sent to the teletypewriter. At no time should the teletypewriter operator manually initiate paper tape motion except by the X–ON key.

The paper tape mode is terminated by a series of two X–OFF characters in a row followed by a DEL and this causes the message 'END OF TAPE' to be displayed. These may be on the tape or entered manually.

Several of the special characters are treated differently for form I paper tape input.

BREAK — Terminates paper tape mode (no more X–ON characters are sent to the teletypewriter). The normal rules for manual input after the BREAK key input apply. Paper tape mode may be reinitiated by pressing the X–ON key. The BREAK key should not be used while the tape is in motion.

? — Causes the image in which it occurs to be ignored; however, the image must still end with the LF X-OFF CR DEL sequence.

" — Causes a one-character backspace.

LF — Needed in the end-of-image sequence to produce a readable copy on the teletypewriter printer. The LF is never considered part of the image text and is treated like a DEL.

If a tape is improperly formatted, or if characters are typed in manually while in the tape mode and the symbiont is not ready for more input, the tape mode is terminated and the message ***WAIT*** is sent to the teletypewriter. The tape mode may be reinitiated with X-ON.

If, for any reason, no input or output occurs for more than five minutes, the tape mode is terminated and the message TIMEOUT is displayed. If no further action occurs within another five minutes, the site is terminated. A tape which ends without the end-of-tape sequence can cause this since the symbiont will have sent a request for input (X-ON) and cannot do output until the request is satisfied. This problem may be cured by inserting three X-OFF's manually.

The control statements @RUN, @FILE, and @FIN (see 3.4.1, 3.8.1, and 3.4.2, respectively) should never appear on a paper tape, except that a @RUN control statement may occur while in the @DATA mode (see 18.3).

The model 33 teletypewriter must have the option which allows the teletypewriter to initiate tape motion by sending an X-ON to the teletypewriter. This feature also includes the ability to have the tape stop when an X-OFF is read.

12.2.1.2.2. FORM II PAPER TAPE INPUT

Form II provides for continuous paper tape input with no interaction until the end-of-tape signal is received. The images are buffered on mass storage. At end-of-tape, the buffered input is internally added to the input stream.

The only requirement as to the image format is that a CR character mark the end-of-image. A LF, however, is useful for monitoring the tape as it is read. Without a LF, overprinting of each image will result. No X-ON or X-OFF characters are needed or desirable on the tape.

The following procedure is used for form II paper tape:

(a) The user must have an active run. If an attempt is made to read form II without a run active, the following message is printed at the terminal:

NO RUN ACTIVE

(b) To start the tape input, the terminal operator must press keys CNTL and TAPE. The message.

TAPE START

is printed at the terminal. If the tape is in the reader and the reader MODE switch is set to AUTO, the tape is read automatically. If the tape is not in the reader, the operator must place it there and set the reader MODE switch to MANUAL READ.

(c) When the tape read is completed, the terminal operator then must press keys CNTL and TAPE with LINE. The message

END OF TAPE

is printed at the terminal. Tape input images are now added to the input stream.

The @RUN and @FIN control statements are disregarded if they are on the input tape. It is suggested that neither should be used, except when using @DATA or @ELT,D control statements (see 18.3 and 18.2, respectively).

The EOT character is recognized in the form II mode. The terminal run is terminated as in manual mode.

The rub-out character may be used to rub out errors during preparation of the tape. The rub-out is ignored by the handler as are nulls.

The question mark and double quote are not recognizable as delete characters.

12.2.1.3. SPECIAL CONTROL SEQUENCES

Table 12–2 lists the teletypewriter control characters and their functions. These control characters are used to control image formats, image input, and so forth.

Keyboard Key	Function	Description
?	Delete	When received from the terminal, the current image is discarded. The symbiont responds to delete function with a CR/LF sequence.
RETURN	End of Image	Used to indicate the end of the input image. Maximum input image length is 72 characters. The symbiont responds with a LF.
EOT	End of Transmission	Terminates and disconnects the teletypewriter terminal.
"	Erase	One preceding character is deleted each time the key is pressed. Characters are deleted right to left. When the character count is reduced to zero, the symbiont responds with a CR/LF sequence.
ESC	Escape	When the key is pressed, the next character and only the next character is inputted in the escape mode. This control character allows the user to input the ?, #, and " characters as data.
BREAK or RTS	Interrupt	Causes the symbiont to suspend its current operation and accept an input image immediately. The message INTERRUPT LAST LINE is sent to the terminal. When the user keys in the CR following his input image, the line of output that was interrupted is immediately resumed. (See 12.2.1.4.)

Table 12-2. Teletypewriter Control Characters

12.2.1.4. BREAK KEY

The BREAK key character is represented on the keyboard as BREAK or RTS. The BREAK key may be used at any time other than within an input image; that is, once an input image has been started. A CR or a ? (line delete) must be sent before the BREAK key can be used. If used within an input image, the BREAK key is inserted in the image as an unknown character.

Upon receiving the BREAK key, the symbiont suspends its current operation and sends the following message to the terminal:

INTERRUPT LAST LINE

Teletypewriter models 33 and 35 require that the BRK/RLS key be pressed before any character can be sent from the terminal after a BREAK key.

The symbiont is now ready to receive one of three possible commands.

(1) Terminate user execution. This is accomplished by the character X followed by a CR. If the X is received while in the @ADD mode, all @ADD files and backed-up input are discarded. The next input is then expected from the terminal.

(2) Contingency interrupt. The contingency routine specified for the user's run is given control with the error code of 0_8 in the contingency status word when any character other than an XCR (for example, just a CR is received by the symbiont). If a contingency routine has not been specified, the execution is terminated as if the BREAK key was followed by an XCR.

(3) Remove a run from a facilities-held status. From a terminal, the BREAK key followed by the keyin of an X may be used to take a demand run out of the hold status due to its facility requirements. All backed-up input including the facilities request is discarded.

If the terminal operator should decide not to enter a command or if he decides he has made an error after pressing the BREAK key, he may use the question mark (?) character to signal the symbiont to disregard the BREAK key interrupt.

12.2.1.5. TAB CONTROL STATEMENT (@TABSET)

The demand terminal user can define horizontal tab columns for the input by use of the @TABSET control statement. The only optional parameter is the label parameter. The format of the @TABSET control statement is

```
@label:TABSET   X1,X2,...,X18
```

where X1 is tthe numeric specification of the tab column ranging from 1 through 80, specified in ascending order. A maximum of 18 tab positions may be specified. The following is an example of a @TABSET control statement:

```
@TABSET   3,20,30,57,60
```

Once the @TABSET control statement has been introduced from the terminal, it is in effect until either another @TABSET control statement or a @FIN control statement is received by the symbiont.

If the @TABSET control statement is accepted (no error message), the tab character (press the CNTL and I keys on the keyboard) spaces the next character of input to the next position specified by the @TABSET control statement.

Should the symbiont encounter the tab character when a @TABSET definition has not been specified or when the last defined position has been exceeded, the character is placed in the input image as a Fieldata 0.

If the @TABSET control statement is in error. The message

```
TAB STATEMENT ERROR
```

is displayed on the terminal and any previous tab definition is ineffective.

12.2.1.6. CENTRAL SITE TO REMOTE SITE OPERATOR COMMUNICATION

Two unsolicited keyins are available which enable the central site operator to initiate remote site communications.

The teletypewriter broadcast keyin can be used to display the specified text at all active teletypewriter terminals. The format of the keyin is

```
TB   text
```

where text is a maximun of 50 characters including imbedded blanks.

The teletypewriter message keyin can be used to display the message at the active terminal specified in the keyin. The format of the keyin is

```
TM   site-id/options   text
```

where the site-id/options and text parameters cannot exceed 50 characters including imbedded blanks. The only option available is the I option which specifies that the text be immediately displayed on the terminal. If the specified site is inactive, the following message is displayed on the operator's console:

```
SITEID NOT ACTIVE
```

There can be only one teletypewriter message to a specific terminal outstanding at any time. If the operator attempts to display a message on a teletypewriter which already has an outstanding message, then the following message is displayed on the operator's console:

TM OUTSTANDING

Except when the I option is specified, the broadcast or teletypewriter message is not displayed until after execution of the current task or when the system encounters a @RUN control statement.

12.2.1.7. FRIDEN 7100

The Friden 7100 is supported as a demand terminal by the executive. All the operating procedures for the teletypewriter apply (see 12.2.1.1 through 12.2.1.6) to the Friden 7100 with the following exceptions:

(1) The at symbol @ is accepted as the Fieldata master space by the teletypewriter input symbiont. The Friden does not have an @ sign on its normal keyboard (only as a control key character).

(2) The teletypewriter output symbiont converts the Fieldata master space @ to an # sign for the Friden terminals.

(3) All Friden site-ids have the character F as the second character. This allows the teletypewriter symbiont to recognize the terminal as a Friden.

(4) For the teletypewriter, a line can be considered accepted by the symbiont if the input character, CR sequence gives a LF/CR sequence without causing a WAIT message. This is not true on the Friden, however, due to its giving a hardware LF when the CR key is pressed. Therefore, there is no positive means of knowing if the line was accepted from the Friden.

12.2.1.8. DCT 500 IN TELETYPEWRITER MODE

A DCT 500 operating in teletypewriter mode is very similar to teletypewriter operation. There are, however, some minor considerations:

(1) The DCT 500 must be strapped to appear to the system as if it were a teletypewriter. Specifically, the DCT 500 hardware must have the following:

(a) The RID, SID, and STX feature must be inhibited.

(b) The parity select feature must be set to ignore parity on data received from the system.

(c) The DCT 500 must be in the master mode.

(d) The DCT 500 full/half-duplex option must be set to the half-duplex mode.

Once the terminal has established a line connection with the central site, the terminal operator must depress the PROCEED key to establish clear-to-send at the DCT 500. The CLEAR TO SEND indicator lights if the data set is in data mode when the PROCEED key is pressed. Once this sequence is performed, the terminal operator can send a site-id to the system.

(2) The DCT 500 has the capability of generating the full ASCII character set; however, the teletypewriter symbiont does not handle the full ASCII set. Lower case characters are translated as upper case characters, that is, a lower case a and an upper case A will produce a Fieldata A after translation by the teletypewriter symbiont. It should be noted that the idle line logic does not recognize lower case ASCII; therefore, the terminal operator must key-in the alphabetic characters of the site-id in upper case. There is no upper case for the ASCII numerics. The second character of site-id must be a D to signify that the terminal is a DCT 500. The teletypewriter symbiont allows the DCT 500 to receive up to 132 characters of output per line.

(3) Whenever the terminal operator desires to utilize the break (interrupt last line) feature, press the INTRPT key. This key is analogous to the BREAK key on the teletypewriter.

(4) When submitting form II paper tape from the DCT 500:

(a) The teletypewriter CNTL-TAPE character is a CTL-R on the DCT 500 and the CNTL-TAPE-LINE character is a CTL-T.

(b) The teletypewriter rub-out character is the shift with underline on the DCT 500.

12.2.2. DCT 500 DEMAND SYMBIONT (SEMI-AUTOMATIC)

The DCT 500 demand symbiont provides support of the DCT 500 operating in the semi-automatic mode, that is, a DCT 500 operating on a multidropped, polled network.

12.2.2.1. OPERATIONAL CONSIDERATIONS

Initialization, run stream submission, termination, and remote system messages are as described in 12.1.1 with the following additional considerations:

(1) A brief description of the switches, indicators, and general operation of the device is given here to facilitate demand use of the DCT 500. It should be noted that not all DCT 500 configurations have all these switches, and thus, not all the capabilities.

MASTER/SLAVE — Permits the DCT 500 to be the initiating station if in MASTER position. To operate within the symboint, this switch must be in the SLAVE position.

XMIT OFF/REC MON — This switch is effective only when the DCT 500 is in full-duplex mode. Since the DCT 500 must be in half-duplex mode to operate with the symbiont, the setting of this switch has no function.

BAUD RATE — This switch is used to set the clock in the DCT 500 to the same rate as clock in the communications terminal at the central site to which the DCT 500 is connected. The operator must know the rate of the line he is using and set this switch accordingly.

ON LINE/OFF LINE — This switch must be set to ON LINE in order to make connection with the computer.

KEYB'D/OFF — Since the primary input device is the keyboard, this switch must be in the KEYB'D position.

PRINTER/OFF — This switch should be in the PRINTER position to allow the printer to be selected.

READER/OFF — This switch should be in the OFF position if paper tape is not to be read. It must be in the READER position before paper tape can be read.

PUNCH/OFF — This switch should be in the OFF position if paper tape is not to be punched. It must be in the PUNCH position before paper tape can be punched.

The keyboard indicators are also important to the user. The INTRPT indicator, when lit, indicates a break signal was received. This normally happens when the line is dropped. It can be cleared by placing the DCT 500 in the offline mode and pressing the PROCEED key. The CLEAR TO SEND indicator, when lit, indicates the keyboard is selected for input. This is the only time the terminal operator will be able to enter data from the keybaord. The PARITY CHECK indicator, when lit, indicates a character was received which had bad parity. The character with the bad parity is printed or punched as an asterisk and the indicator extingushes when the output device is deselected. The FORMS OUT indicator on the printer lights when the forms are depleted and stops normal operation of the terminal.

After the line connection is made, the terminal operator should watch the CLEAR TO SEND indicator to know when he has been polled and is able to enter the site-id for his terminal. Any letters in the site-id must be entered in upper case by striking the SHIFT key along with the letter key. Two time restrictions are placed on the operator in the entering of the site-id. He has only a few seconds from the time the CLEAR TO SEND indicator lights to enter a character. If this time passes without a character being sent, the indicator extingushes indicating he has been deselected and he must then wait to be polled again. After the first character has been struck, he has 15 seconds to enter a valid site-id. The terminal is deselected if this time elapses and the operator must again wait to be polled. When the valid site-id is received, the standard initialization message is sent to the terminal printer. The terminal is now considered an active terminal, and the keyboard is selected so that the user can start his input.

(2) Input images to the computer must be 72 characters or less. If this limit is exceeded, the message MESSAGE TOO LONG is sent to the printer and the image is lost. An input image consists of one or more characters with the last character being a CR. The response of the symbiont is a CR/LF to the printer. The tab control statement feature can be used to give the terminal user flexibility in entering images (see 12.2.1.5).

The primary input device is the keyboard and the @RUN control statement should be the first input from the keyboard following terminal activation. After the @RUN control statement has been entered, the input device can be changed by the terminal operator to the paper tape reader if he desires to do so and he has one attached to his DCT 500. The input device can subsequently be changed back to the keyboard. This switching of input devices is done from the terminal by a CTL-R being entered from the selected input device.

(3) Paper tape input from the DCT 500 paper tape reader is handled the same as form II paper tape input from the teletypewriter. A complete description of form II paper tape is given in 12.2.1.2.2. The following additional considerations apply when form II is inputted from a DCT 500 paper tape reader:

(a) To start the tape input from the DCT 500, the terminal operator must press the CTL-R keys. If the tape is in the reader and the READER switch is set to on (up), the tape is automatically read. If the tape is not in the reader, the operator has about five seconds to ready the tape and reader before the keyboard is selected and made the input device.

From the user program the tape is started by giving the system the control image D,RD. through the PRTCN$ request (see 12.2.2.3).

(b) The format of the paper tape is as follows:

TEXT CR LF TEXT CR LF ..... TEXT CR LF CTL-R

Paper tape mode is terminated when the CTL-R is read from the paper tape. The CTL-R cannot be entered from the keyboard as the input device currently selected is the paper tape reader until selection is switched by the CTL-R character being received.

(c) Special Considerations:

The CTL-EOT character is recognized in the form II mode. The terminal and run are terminated as in manual mode. It is recommended that the tape mode not be terminated by the CTL-EOT.

The shift with underline character may be used to rub out errors during preparation of the tape. The shift with underline is ignored by the handler as are nulls.

The double quote is not recognized to delete characters.

The question mark causes image deletion.

The CR without preceding data characters does not pass a blank card to the system.

The CTL-T causes the selection bit for the output devices to be switched; that is, the punch is the output device now instead of the printer or the printer is instead of the punch.

The pound sign (#) is not recognized as a master space (@).

If the CTL-R is not the last character on the tape, additional tapes may be read until a a tape is read which has a CTL-R on it.

(4) No special action is required by the operator in handling output to the output device. He can control which output device is selected as the output device. The primary output device is the printer. A CTL-T being entered from the input device changes the output device selection. If the hardware is set up for it, the output can also be stopped by pressing the INTRPT key.

While the punch is the output device, overprinting may result. Therefore, the operator must press the NEW LINE key and then the RETURN key when he ends an input image.

The user should be aware that the DCT 500 hardware performs an automatic form feed every 60 lines. This might be useful if the user is formatting his output and desires to go to a new page. The user can count his output lines and issue the appropriate number of blank lines to get to the top of the new page.

(5) It is the responsibility of the terminal operator to remove his terminal from the active status when he is finished. The normal way to terminate the terminal is to enter a @FIN control statement and, when the message ***LINE INACTIVE*** comes out on the printer, enter the DEOT sequence. This sequence is the DLE character followed by the EOT character. The DLE is the CTL-P key and the EOT is the CTL-D. The terminal now becomes inactive and the line connection is dropped on a dialed line with no other terminals on the line.

12.2.2.2. SPECIAL CONTROL SEQUENCES

Table 12–3 lists the DCT 500 control characters and their functions. These control characters are used to control image formats, image input, and so forth.

Keyboard Keys	Function	Description
?	Delete	When received from the terminal the current image is discarded. The symbiont responds to the delete function with a CR/LF sequence.
RETURN	End of Image	Used to indicate the end of the input image. Maximum input image length is 132 characters. The symbiont responds with a LF.
" or ←	Erase	One preceding character is deleted each time the key is pressed. Characters are deleted right to left. When the character count is reduced to zero, the symbiont responds with a CR/LF sequence.
INTRPT	Interrupt	Causes the symbiont to suspend its current operation and accept an input image immediately. The message INTERRUPT LAST LINE is sent to the terminal. When the user keys in his CR following his input image (the X keyin is the only keyin recognized by the symbiont), the line of output that was interrupted is immediately resumed.
X CR	Task Termination	Terminates the last task submitted to the executive if the task is not already completed. This keyin can be submitted only after a break (INTRPT).
DLE EOT	End of Transmission	Terminates and disconnects the teletypewriter terminal.

Table 12–3. DCT 500 Control Characters

12.2.2.3. USER/PROGRAM INTERFACE

The standard symbiont services requests as defined in Section 5, which enables the user program to interface with the DCT 500. The print control functions (PRTCN$) have been expanded to facilitate the users control of DCT 500 terminal. The following character strings describe the control functions that are defined for DCT 500 symbiont:

D,KB. — Causes the symbiont to make the keyboard the input device regardless of what is considered the input device.

D,RD. — Causes the symbiont to make the paper tape reader the input device and initialize form II.

D,PU. — Causes the symbiont to make the paper tape punch the output device.

D,PR. — Causes the symbiont to make the printer the output device.

There can be two input and two output devices on a terminal; however, only one input and one output device is designated the input and the output device at a time. These functions from the user program and the CTL-R and CTL-T characters from the terminal are the methods the user has of controlling which devices are designated the input device and the output device. This causes the symbiont to send the proper device-id when it polls the terminals.

12.2.3. UNISCOPE 100/DCT 1000 DEMAND SYMBIONT

The UNISCOPE 100/DCT 1000 demand symbiont provides support for the following terminals:

- Combination of single, multiplexed, and multidropped UNISCOPE 100's and associated communications output printer.

- DCT 1000, keyboard, and printer

12.2.3.1. OPERATIONAL CONSIDERATIONS FOR THE UNISCOPE 100

Initialization, run stream submission, termination, and remote system messages are as described in 12.1.1 with the following considerations:

(1) Before the UNISCOPE 100 becomes an active terminal, the operator must turn it on and establish the proper line connection. Data may be entered on the screen prior to establishing the line connection. Data entered in such a fashion is not destroyed by the symbiont as long as it initially does not exceed line 10. This is useful for preentering useful messages prior to accumulating telephone line charges.

After the device has been turned on and the line connection is made, the device is polled at 30- second intervals (the inactive device polling interval which may be changed at system generation time). The first message transmitted from an inactive device must be the six-character configuration site-id for this device. The site-id may be transmitted from anywhere on the screen preceded by an SOE (⊳) character.

If the transmitted message is received properly, the initialization message is sent to the device and the SOE and cursor character are positioned at the standard insert point ready for the next operator input. If the message is not received properly, no response is sent to the device. If the operator is sure that everything is functioning properly and that the transmitted site-id is valid, then the following steps should be taken after a waiting period of approximately 60 seconds:

(a) Press the WAIT switch (to unlock the keyboard)

(b) Transmit the message again.

The operator sees a positive action when the device is polled to pick up the transmitted message by the reappearance of the cursor character. The cursor disappears from the screen when the TRANSMIT key is pressed and reappears when the device is polled.

(2) Input messages are restricted to 72 characters or less in length. If the operator desires to transmit information which exceeds the 72 character limit, it will have to be broken into smaller messages through the use of additional SOE (⊳) characters.

After the operator has pressed the TRANSMIT key, the cursor character disappears from the screen. When the device is polled to pick up the transmitted message, the cursor reappears. If the message is received properly by the computer, the symbiont generates and sends a keyboard unlock message, moves the screen up or down (depending on current setting – see 12.2.3.3) one line, and positions an SOE (▷) character and the cursor in columns one and two of the insert line. If the inputted message produced a response, it is sent to the insert line and thus destroys any information the operator may be entering. If no immediate response is produced, the operator is free to enter a new line to be transmitted.

All data transmitted by the remote operator must follow submission of an @RUN control statement except for special control sequences. (See 12.2.3.3.)

(3) Operator input activity is timed to determine and prevent unnecessary communication facility in 2.5 minute intervals. The following symbiont action takes place at the successive increments of no activity.

1st Increment – The active poll interval is doubled. If the standard active poll rate is used, it is now eight seconds.

2nd Increment – The active poll interval is tripled. For the standard active poll rate, it is now 12 seconds. A timeout message is sent to the device. Any break hold on output is removed.

3rd Increment – The active poll interval now becomes four times its original value. If standard value is used, it is now 16 seconds.

4th Increment – At this time, set by system generation option, one of the following will occur:

(a) The device is terminated and reverts to an inactive state or

(b) The timeout message is sent to the device again and repeated every increment time following. The poll interval is now set at five times the original value (20 seconds for standard poll interval).

(4) No special action is required of the operator in handling output to the device. The user does have a control option to skip a certain number of successive output lines (see 12.2.3.3). He also has the option of putting a temporary hold on any output by use of the message waiting button (break mechanism – see 12.2.3.3).

The symbiont controls the screen (unless a user program assumes control) with output messages (see 12.2.3.4). The operator must use caution in entering new data if his previous input is expected to produce a response. The symbiont control could possibly cause an overwrite of the data being entered by the operator.

(5) Normally the terminal operator need not be concerned with transmission error recovery. The symbiont controls all retry sequences. No retry sequences are performed on the initial activation message. If the retry attempts are unsuccessful, the device is automatically returned to the inactive state. The terminal operator has to perform the initialization procedure after the transmission problem has been corrected. Any uncompleted terminal activity must be redone after the device is reinitialized.

Several messages can be returned to the terminal operator from the symbiont in response to an incorrect action.

IMPROPER SYM CONTROL MSG

The operator has submitted a symbiont control message which contains improper information.

MESSAGE TOO LONG

The operator has attempted to transmit a message which exceeds the input limit of 72 characters.

(6) It is the terminal operator's responsibility to remove his device from the active status when he is finished. This is accomplished by transmitting the symbiont control message ""TERM.

12.2.3.2. OPERATIONAL CONSIDERATIONS FOR THE DCT 1000

The general operating procedures for the DCT 1000 are very much the same as those for the UNISCOPE 100 (see 12.2.3.1) with the following additional considerations:

(1) Initialization Procedure:

(a) Establish line connection

(b) Set switches to the following positions:

- AUTO/MAN – AUTO
- MONITOR ON/OFF – ON
- ON LINE/OFF LINE – ON LINE
- KEYB'D/OFF – KEYB'D
- All other device switches – OFF

(c) Press CLEAR key and set RUN/STOP switch to STOP position and then run.

(d) Enter the six-character configuration site-id for this terminal from the keyboard.

(e) Press the TRANSMIT key.

If the transmitted site-id is valid for this particular terminal and is received properly, the standard initialization message is printed at the terminal. If the message is not received properly by the central site, no message is sent. The operator should then validate the site-id entered and repeat the initialization sequence.

(2) Input messages are limited to 80 characters. When this limit is exceeded, the,symbiont rejects the message and notifies the terminal operator with the message MESSAGE TOO LONG. After the operator has pressed the XMIT key, the KEYB'D READY indicator extinguishes and remains extinguished until the text is acknowledged by the symbiont. If no response is produced by the input, the operator is free to enter a new line for transmission.

Unlocking of the keyboard after an input text transmission is not necessary with the DCT 1000. The acknowledgment to the text has the effect of releasing the keyboard for more input. Because of the hardware reactions input should not be attempted if output is pending. Once input is initiated, that is, the initialization procedure has been performed, the user must press the XMIT key as soon as possible; otherwise, the symbiont will disconnect the terminal.

(3) No special action is required of the terminal operator in handling output, that is, no output device switch need be set. No interrupt key is available on the DCT 1000 and, therefore, output cannot be halted once it begins. The special control sequence to skip output and terminate can be used if it can be entered at a point when no output is expected, for example, prior to a statement which will create output.

A CR control character is inserted in text messages to the DCT 1000 by the symbiont and, as such, user control of line spacing is not allowed. Control characters may be placed in the text if the user desires; the symbiont, however, inserts its own carriage control regardless of what the user does.

(4) Termination of the DCT 1000 is identical to that of the UNISCOPE 100.

12.2.3.3. SPECIAL CONTROL SEQUENCES

Certain control messages and sequences are available for use by the device operator for controlling the symbiont's handling of the device. The messages are defined as starting with two double quote characters. The format of the control message is

""xxxx yyyyyy

where xxxx is the control type and yyyyyy is parameter information.

Table 12–4 lists the control messages and characters along with their functions for the UNISCOPE 100 and the DCT 1000.

Control Message or Action	Function on UNISCOPE 100	Function on DCT 1000
''''LINE xnn	Changes roll direction and insert point. x – When U is specified, roll direction is changed to up; when D is specified, roll direction is changed to down. If neither U nor D is specified, no change in roll direction occurs. nn – Sets insert point to line nn. The range of nn is dependent upon the number of lines on the screen. For a 12 x 80 screen, the range is 1–12. It is not advisable to set the insert point to line 12 because some output messages exceed one line.	Not applicable
''''PRNT	Turns printer on or off (if one is attached to UNISCOPE 100). The ''''PRNT message always reverses the previous setting, that is, off to on or on to off. When printing is initiated, the symbiont attempts to select the printer. If successful, the printer's SELECT indicator lights and all data to and from the UNISCOPE 100 is printed as well as displayed on the screen. If the selection cannot be made (printer out of forms, cover up, and so forth), the message PRINTER CANNOT BE SELECTED is displayed, the printer's SELECT indicator does not light, and no printing occurs. When terminating printing, the UNISCOPE 100 deselects the printer and all subsequent data is displayed on the screen only. If the ''''PRNT message is sent from a UNISCOPE 100 that does not have the auxiliary interface needed for the printer, the message IMPROPER SYMBIONT CONTL MSG is sent to the terminal and no additional action occurs.	Not applicable
''''SKIP nnnn	Skips nnnn number of output images. The range of nnnn is 1–2047. If nnn is omitted, then a maximum skip value of 2048 is assumed. The skip process is terminated either by skipping the specified number of output images or by transmitting another ''''SKIP message with nnnn 0. The skip option is normally used with the MESSAGE WAITING key (break mechanism) to put a hold on output.	Same as for UNISCOPE 100

Table 12-4. UNISCOPE 100 and DCT 1000 Control Messages and Sequences (Part 1 of 2)

Control Message or Action	Function on UNISCOPE 100	Function on DCT 1000
""TERM	Removes terminal from the active list. If a run is active, a @FIN control statement is generated and accounting information is displayed.	Same as for UNISCOPE 100
Delete	Performed by using the editing control keys (see UNIVAC UNISCOPE 100 Display Terminal Operator Reference Manual, UP–7788 – current version)	Line deletion is accomplished by pressing the CLEAR key. A CR should also be done to avoid the confusion of entering a new input line on the same line.
End of Image	Implemented by pressing the TRANSMIT key. ıd	Press XMIT key.
Erase	Performed by using the editing control keys (see UNIVAC UNISCOPE 100 Display Terminal Operator Reference Manual, UP–7788–current version).	To erase a character, press the ←(backspace) key. This has the effect of erasing the previous character. Exercise care when doing this, however, because there is no visual way of determining what is actually in memory.
Interrupt	Press MESSAGE WAITING switch. It places a hold on the output unit the next input message is received from the terminal. The hold is timed and is removed after five minutes even if no message is input. The hold interval can be changed at system generation time. The interrupt (or break) mechanism is normally used for: (a) used to input ""SKIP control messages, or (b) used to input X keyins. When the interrupt is received by the symbiont, the message INTERRUPT LAST LINE is sent to the terminal.	Not applicable – no interrupt is possible with a DCT 1000
X Keyin	Terminates the last task submitted to the executive if that task has not already been completed. The X keyin can be submitted only after pressing the MESSAGE WAITING switch. The X must be placed immediately following the SOE character in order for it to be recognized.	Not applicable – DCT 1000 cannot be interrupted

Table 12–4. UNISCOPE 100 and DCT 1000 Control Messages and Sequences (Part 2 of 2)

12.2.3.4. USER PROGRAM INTERFACE

The standard symbiont executive requests as described in Section 5 enable the user program to interface with the UNISCOPE 100 or the DCT 1000 terminals. There are, however, the following exceptions to the standard use of the symbiont executive requests when interfacing with the UNISCOPE 100 or the DCT 1000 terminals.

- OUTPUT

  The spacing specifications of the PRINT$ request have no affect on UNISCOPE 100 or DCT 1000 image output. The output images may contain control information such as cursor to home, cursor address, line feed, form feed, and so forth. Be careful when using screen control information because the symbiont maintains control of the screen unless directed otherwise by a PRTCN$ request from the user program.

- OUTPUT/INPUT

  The spacing specifications of the TREAD$ request have no affect on UNISCOPE 100 or DCT 1000 image output. On the UNISCOPE 100, the TREAD$ output image is handled differently by the symbiont in that the SOE character and cursor are left-positioned following the output image. For a PRINT$ image, the SOE and cursor are positioned at the start of the next line.

- PRINT AND SCREEN CONTROL

  The print control functions (PRTCN$) have been expanded to facilitate the user' control of the UNISCOPE 100 terminal. These functions are not available to DCT 1000 user.

The following control functions are defined for screen and print control:

Insert Point — Defines the insert point on the scrren to the line number nn. This causes the screen to roll up from this insert point unless a specific function is used to roll it down.

Roll Direction— Provides for changing the movement direction of the screen. UP causes the screen to roll up from the insert point (standard or L defined). DOWN causes it to roll down.

Printing — Provides printer control. START selects a printer provided one is configured and ready to be selected (that is, turned on and not out of forms). Text to the UNISCOPE 100 is printed as well as being displayed following the selection, provided a valid selection was made. If the selection is unsuccessful (for example, of forms, cover-up, printer turned off), the message PRINTER CANNOT BE SELECTED is displayed and no printing takes place. If the auxiliary interface is not present, the command is ignored. STOP deselects the printer and turns printing off.

Control — Allows the user to take over complete control of the terminal screen and to switch it back to the symbiont. If the control message is X,USER, then the following action takes place:

(1) The symbiont will not embed screen control sequences in the message. The user assumes full responsibility for screen formatting and display.

(2) The symbiont will not print output on the communications output printer even though a printer has been selected and is available for use.

(3) The address of the SOE is included in all input messages. This SOE address is not included in the 72-character count limit for input messages.

(4) The user remains in control of the screen until one of two things happens; the sending of a X.SYM message through a PRTCN$ request or the execution of the user program terminates.

To summarize the options available to the user program by a PRTCN$ request, the following are the legal image formats:

X,nn.	To change insert point nn
X,UP.	Change roll direction to up
X,DOWN.	Change roll direction to down
X,START.	Start printing
X,STOP	Stop printing
X,USER.	Switch screen control and printing to user
X,SYM.	Switch screen control and printing to symbiont

12.2.4. UNISCOPE 300 DEMAND SYMBIONT

The UNISCOPE 300 demand symbiont provides support for single station or multistation UNISCOPE 300 display terminals.

12.2.4.1. OPERATIONAL CONSIDERATIONS

Initialization, run stream submission, termination, and remote terminal messages are as described in 12.1.1 with the following additional considerations:

(1) After the operator has dialed the line with which his UNISCOPE 300 is configured, he initializes the station with the following procedure:

(a) Setup message on screen: SOM, (site-id), cursor

For example: △ SCOP1 ⌉

(b) Press TRANSMIT key.

(c) Wait for the standard initialization message.

(d) Submit a @RUN control statement (using normal executive system format except that # is used in place of @.

If the site-id is not the assigned value for that station, it is ignored and no response is received from the symbiont. This results in a keyboard lock (WAIT indicator lights) which requires a manual turn off, turn on by the operator to recover. The operator may then submit the correct site-id as described above.

(2) Input images use the format:

(a) Start-of-message (SOM) character

(b) Text (maximum of 72 characters)

(c) Cursor

For example: △ t e x t ...... ⌉

The input is initiated by the operator pressing the TRANSMIT key. This action also lights the WAIT indicator. Image acceptance by the symbiont is indicated at the station by the extinguishing of the WAIT indicator and the appearance of an SOM character in the line following the one containing the input image.

Several system messages are associated with input, they are:

Message	Description
NO RUN ACTIVE	The last input was an invalid #RUN control statement.
MESSAGE TO LONG	Input image contained more than 72 characters.
IMPROPER SYM CNTRL MSG	Input message was an invalid control message.
BAD INPUT SEQUENCE	The input control message was rejected because it conflicts with the current status of the run.
WAIT	Symbiont interface is not ready to accept additional input. Station must wait for output to occur or until the message does not reappear. Any input causing this message must be resubmitted.

(3) When the user is finished, the station may be deactivated by the control message ""XMY". If this is the last station on the line, the line is also deactivated and returned to the idle line monitor mode. In either case, the system message STATION INACTIVE is sent to the station prior to deactivation.

12.2.4.2. SPECIAL CONTROL SEQUENCES

The MESSAGE WAITING key provides the interrupt capability. An input image or control sequence can be sent to the central site when the output has been interrupted. Output is restarted when the MESSAGE WAITING key is pressed a second time. This key can be pressed at any time on a multistation unit, but on a single station unit it must be pressed just after the line is polled and just prior to the subsequent acknowledge poll.

Table 12–5 lists the control sequences which allow the user to control the UNISCOPE 300. The MESSAGE WAITING key must be used to interrupt the symbiont prior to transmitting these control sequences.

Control Sequence	Description
""ABF"	If the ""AFT" control sequence is not transmitted, all output for the current run is discarded. No indication is returned to the terminal that the print file is being discarded unless a #FIN control statement was previously processed for the run.
""AFT"	Any output generated after the submission of this control sequence is transmitted to the terminal.
""DPP"	Delete one page of print output (64 images)
""HIS"	The speed of output relative to the polling rate can be set to the fastest rate with this message. Initially the speed is set to high. (Output rates depend not only on the speed selected and the fixed polling interval, but will slow considerably if other stations on the same line are also outputting continuously.)
""LOW"	Set output speed to about 1/4 of the ""HIS" rate.
""MED"	Set output speed to about 1/2 of the ""HIS" rate.
""ROL"	An upper roll limit for screen contents is defined on the screen by sending this message to the central site from a particular line (vertical position) on the screen. For subsequent output, that line plus all others below it are scrolled. The lines above it remain fixed on the display until altered by another ""ROL" message. Initially all screen lines are scrolled.
""XMY"	Places terminal in the inactive state. If this is the only active terminal on the line, the line is deactivated and placed in the idle line state.
""XQX"	Terminates the currently executing run.

Table 12–5. UNISCOPE 300 Symbiont, Control Sequences

12.2.4.3. USER PROGRAM INTERFACE

The standard symbiont executive requests (see Section 5) enable the user program to interface with the UNISCOPE 300.

12.3. TERMINAL USER TECHNIQUES

Although the demand user can use all the capabilities of the operating system, certain techniques more effectively utilize the system. The following techniques are given only as aids and are not mandatory:

(1) Avoid sending program code directly to the language processors (except a conversational processor such as CFOR). An appropriate alternative is the ED, ELT, or DATA processors.

(2) Avoid using the SNAP$ request whenever possible.

(3) Copy information stored on magnetic tape to mass storage as quickly as possible and then release the tape unit.

(4) When using a processor that requires an end sentinel, input the end sentinel as quickly as possible.

12.4 EXAMPLE OF A DEMAND RUN

A simple example follows to demonstrate the use of the system is a demand processing mode from a teletypewriter.

```
U1108A ◄──────────────────────────────────────────── User enters site-id.

UNIVAC 1108 TIME/SHARING EXEC  VERS 27.20.12A-MP ◄──── System responds

@RUN,B HARRIS,489331,APL,10,500 ◄──────────────── User: @RUN image
DATE: 060171       TIME: 082429 ◄───────────────── System responds
@MSG 032? ◄──────────────────────── User makes mistake and erases line.
@MSG HARRIS, 489331, NEED TAPE 428C, NO RINT"G ◄ User gets message right on second try.
                                                 Note deletion of character.
@ASG,CP HARRIS1,F/5/POS/10 ◄──────── User assigns files, system responds with READY
READY
@FREE HARRIS1
READY
@ASG,CP HARRIS2,F/5/POS/10
READY
@FREE HARRIS2
READY

$$$$ OP MSG HARRIS*ASHLAND WISHES TO CONTACT YOU ◄── System operator and terminal
@MSG IS ASHLAND DOWN THERE                            user converse.

@ASG,T     TA,T,428 ◄─────────────────────────────── User assigns tape
WAITING ON FACILITY ◄─────────────────────────── No tape units available
READY ◄───────────────────────────────────────── Servo becomes available
```

```
**TAPF START**<--------------User starts form II paper tape by entering tape-on control character.
@COPIN TA.,HARRIS2.          System responds with message.
@ELT,IDL HARRIS2.BATCH2
@RUN,B HARRIS,489331,APL,10,500
@ASG,A HARRIS1.
@FREE TPF$
@ASG,T TPF$,F/5/POS/10
@ASM,S HARRIS1.INP,HARRIS1.REL
@HDG,P EXECUTION OF APL: RETURN TO TJ HARRIS
@HARRIS1.XQT,YZ ?<-----------------------------------------Tape input
@HARRIS?
@XQT,YZ HARRIS1.ABS
'STRING'
'SWILLTOON'
2 2 4 $R 'ABC'
@ADD DATAPL
@END

END OF TAPE<---------------------------------------End paper tape input.
FURPUR 0021-06/01-08:34<-------------FURPUR processes @COPIN control statement


 12 SYM   3 REL   5 ABS
ELT PROCESSOR LEVEL      4
000001        000     @RUN,B HARRIS,489331,APL,10,500
000002        000     @ASG,A HARRIS1.
000003        000     @FREE TPF$
000004        000     @ASG,T TPF$,F/5/POS/10
000005        000     @ASM,S HARRIS1.INP,HARRIS1.REL '
000006        000     @HDG,P EXECUTION OF APL: RETURN TO TJ HARRIS
000007        000     @HARRIS1.XQT,YZ
000008        000     @HARRIS
000009        000     @XQT,YZ HARRIS1.ABS
000010        000     'STRING'
000011        000     'SWILLTOON'
000012        000     2 2 4 $R 'ABC'
000013        000     @ADD DATAPL
                      <--------------------Paper tape input is completely processed.
```

```
@PRT,T HARRIS2.<---------------------------------------- User lists contents of a file element.

APL*HARRIS2
REL   CRELAD$
REL   SNOOPY
ABS   TDP
ABS   PCT
DOC   EDITDOC(1)
DOC   TDPDOC(1)
DOC   PCTDOC(1)
ABS   EDIT
ABS   EDREP$
ELT   BATCH1(1)
ELT   FIX(1)
ELT   ASMXQT(3)
ELT   ASMTTY(2)
ASM   ASMRUN(5)
ELT   EXECUTE(3)
ASM   ASMTB(5)
ASM   INP(2)
REL   REL
MAP   MAP(1)
ABS   ABS
ELT   BATCH2(1),

@COPY HARRIS2.,HARRIS1.<-------------- User copies files and starts a batch run.
@START HARRIS1.BATCH2
@MSG HARRIS STARTING A BATCH RUN
    89 BLOCKS COPIED
@FIN <-------------------------------- User terminates.

RUNID:  HARRIS   ACCOUNT: 489331        PROJECT: APL <---- System prints this
                                                           termination message.

  HARRIS*MSG: HARRIS, 489331, NEED TAPE 428C, NO RING

  HARRIS*MSG: IS ASHLAND DOWN THERE

   LOAD 428     10/6 TA           -1 HARRIS

TIME: 00:00:01.605    IN: 55     OUT: 0       PAGES:   4

INITIATION  TIME:    08:24:29-JUN  1,1971

TERMINATION TIME:    08:45:36-JUN  1,1971

  HARRIS*MSG: HARRIS STARTING A BATCH RUN


                          ****LINE INACTIVE**** <---- Terminal ready for submission
                                                      of another @RUN control statement
                                                      or an end-of-transmission sequence
```

13. ITEM HANDLER AND BLOCK BUFFERING

13.1. INTRODUCTION

This section describes how the data handling routines are used for the internal manipulation of data files at both the item and block buffering levels. The user program/data handling routine interface requirements, system and data levels of file formatting, file organization, buffer pooling, and error processing are also discussed in this section.

13.2. GENERAL INFORMATION

The data files are considered as either sequential or random, according to the way they are referenced. The items or blocks in a file are further identified as being either fixed or variable in length. Files referenced randomly reside on FASTRAND-formatted mass storage and contain fixed-length items and blocks. Files referenced sequentially reside either on tape or FASTRAND-formatted mass storage and contain fixed- or variable-length items and blocks.

Both sequential and random requests can be used interchangeably to access the same FASTRAND-formatted mass storage file if the file is organized with fixed-length items and blocks. It is therefore possible to process data sequentially up to a certain point and then process it randomly. Conversely, data may be processed randomly until some specific data occurs, and then processed sequentially.

When access to a specific file is requested, the request may be interlocked since the same file may be in use by another activity which is executing at the same time. A lockout feature called exclusive use is provided to perform this function. Exclusive use specifies lockout for a complete file or individual items or blocks in a file. A file being processed randomly is either locked out in its entirety or locked out by items or blocks.

Locking out an entire file requires the appropriate option on the @ASG control statement for the file (see 3.7.1). If a run has a file assigned with the X option (exclusive use option) on the @ASG control statement, any other run which attempts to assign that file is held. Users should be aware that in order to free the exclusive use lockout the @FREE control statement (see 3.7.4) must be processed or the run must terminate. Locking out individual items or blocks of a file is accomplished by using the exclusive read random request (see 13.4.2.4). In this case, only the item or block currently being read is locked out. References to the file by other activities are honored only if the reference is not for the locked out item or block. An activity referencing a locked out item or block is automatically placed in a wait state until the data is released. The next request to the file from the activity that initiated the lock releases the item or block previously held.

A certain amount of independence from peripheral devices is achieved with the data handling routines; that is, files are written on mass storage or any tape type without program alterations simply by changing the @ASG control statement from tape to mass storage or vice versa.

The only restrictions are that all files processed in the in/out mode or referenced randomly must reside on mass storage.

13.3. HANDLING DATA FILES AT THE BLOCK BUFFERING LEVEL

Data handling at the block level is accomplished by a set of reentrant subroutines contained in the block buffering package (BBP). By calling these subroutines, the user can read data blocks from and write data blocks into the files residing on the external storage devices (tape and FASTRAND-formatted mass storage) assigned to the program. In addition, the user may request the initiation, closing, and manipulation of data files.

The BBP subroutines also control the buffer pool with which each file being processed is associated. The buffer pool is shared by all other files in the object program. The data blocks placed in these buffers may be read either sequentially or randomly by the BBP. Sequential read and write operations handle block sizes that are fixed or variable in length. Fixed-length data blocks are used in random read and write operations since the operation is restricted to FASTRAND-formatted mass storage files. References to fixed-length blocks require that their relative block numbers be specified. (Actual sector address of a block and partial filling of a sector are disregarded for random read and write operations.) For variable-length blocks on FASTRAND-formatted mass storage, the BBP precedes each block with a single word to specify the word size of the block.

13.3.1. SUBROUTINES OF THE BLOCK BUFFERING PACKAGE

The BBP consists of ten basic subroutines, two of which are used apart from the BBP. The functions performed by these subroutines are executed in three modes of operation: input, output, and in/out. Some functions are performed in any of the three operating modes while others are restricted to one or two modes only. The mode in which each file is manipulated or processed by the BBP subroutines is defined when the file control table (FCT) is initiated for the file. (See 13.5.1 for a description of the FCT.) Those files defined as input files are limited to read operations whereas output files are restricted to write operations. In/out files are capable of both read and write operations. The functions performed by the subroutines of BBP and the modes in which they are used are listed in the following table:

Function	Mode		
	In	Out	In/out
Initialize FCT To Open File	√	√	√
Initialize and Expand Buffer Pool	√	√	√
Sequential and Random Read	√		√
Sequential and Random Write		√	√
Write Hardware End-Of-File (EOF) Mark		√	√
Closing Tape Reels and Files	√	√	√

13.3.2. DATA FILE MANIPULATION AT BLOCK LEVEL

The manipulation of data files at the block level is accomplished by the initiation of the BBP subroutines. These subroutines are initiated by procedure calls coded within the object program or by requests coded within the user's program. Both methods are described in 13.3.2.1 through 13.3.2.9.

13.3.2.1. INITIALIZING AN FCT FOR SUBSEQUENT BLOCK BUFFERING OPERATIONS (BOPEN$)

Purpose:

Initiates an FCT (see 13.5.1) for subsequent block buffering operations for files processed in the input, output, and in/out modes.

Format:

```
L,U    A0,pktaddr
LMJ    X11,BOPEN$
```

This linkage and the packet may be generated by the procedure call:

BOPEN 'file-mode-mnemonic' FCT-addr(1)[,option] ...FCT-addr(n)[,option]

Description:

Register A0 is loaded with the address of the following one-word packet when the linkage is executed:

S1	S2	S3	H2
[XREAD-flag]	file-mode	[option]	FCT-addr

where:

XREAD-flag — Applicable only to I/O mode files. Places an exclusive-use lock on entire file when set to nonzero value.

file-mode — Specifies processing mode of file. Entries for this field are:

INPUT (2_8) – Input with forward motion (permits read operations only).

REVRSE (3_8) – Input with backward motion (permits read operations only)

OUTPUT (40_8) – Output (permits write operations only)

IN/OUT or INOUT (41_8) – Input/output (for FASTRAND-formatted mass storage file only; permits both read and write operations)

option — The options are:

E (13_8) – Applicable to I/O mode files. Specifies that writing in file starts beyond previous EOF location rather than within former limits of file.

N (23_8) – If a tape file, do not rewind tape.

If the options are omitted, the file is rewound if it is a tape file.

FCT-addr — Address of FCT (see 13.5.1) to be initialized.

The interpretation of the parameters in the procedure call is the same as that specified for the packet word.

The EOF address for a mass storage file must be obtained from the master file directory item and stored in the FCT prior to a BBP open input request for BBP files not created by the present execution. The EOF address is obtained by using a MSCON$ request specifying the DREAD$ function (see 22.3.3).

13.3.2.2. SEQUENTIAL READING OF DATA BLOCKS (BREAD$)

Purpose:

Sequentially reads the next block of data from a file processed in the input or in/out mode.

Format:

```
    L,U    A0,pktaddr
    LMJ    X11,BREAD$
```

Description:

Register A0 is loaded with the address of the following one-word packet when the linkage is executed:

H1	H2
0	FCT-addr

where:

FCT-addr — Address of the FCT (see 13.5.1) for the file being processed.

The buffer area into which the block of data is read is selected by the BBP. Upon completion of the subroutine, the block's length and location are returned in register A0.

If the data block is to be moved into some specific area in the user's program, the following linkage must follow the calling sequence for the BREAD$ subroutine:

```
    L,U    A0,pktaddr
    LMJ    X11,B$MOVE
```

Register A0 is loaded with the address of the following one-word packet when the linkage is executed:

H1	H2
move-to-area-word-count	move-to-area-addr

where:

move-to-area-word-count — Specifies the length of the user's area into which the block is placed.

move-to-area-addr — Location of user-defined area into which block is placed.

The linkage and packet for BREAD$ and, if desired, the B$MOVE subroutines may be generated by the procedure call:

```
    BREAD   FCT-addr[move-to-area-word-count,move-to-area-addr]
```

13.3.2.3. RANDOM READING DATA BLOCKS (BRRED$)

Purpose:

Reads randomly a fixed-length block of data from a specific block location in a FASTRAND-formatted mass storage file processed in the input or in/out mode.

Format:

```
L,U    A0,pktaddr
LMJ    X11,BRRED$
```

This linkage and the packet may be generated by the procedure call:

```
BRRED   block-nbr   FCT-addr[move-to-area-word-count,move-to-area-addr]
```

Description:

Register A0 is loaded with the following three-word packet when the linkage is executed:

		H1	H2
Word	0	exclusive-use-block-flag	FCT-addr
	1	[move-to-area-word-count]	[move-to-area-addr]
	2	block-nbr	

Word 0

exclusive-use-block-flag — Applicable to in/out files. If set to a nonzero value, it prevents the block from being accessed by any concurrent routines calling on BBP with a different FCT (see 13.5.1) pointing to the same file. A subsequent read unlocks (nullifies) exclusive use placed on previous block read.

FCT-addr — Address of FCT for the file being processed.

Word 1

The parameters of this word are the same as specified for the BREAD$ subroutine packet word (see 13.3.2.3).

Word 2

block-nbr — Number of the specific block of data to be read from the file.

If the exclusive use feature is required, substitute BXREAD for BRREAD in the procedure call; if you choose to code the instructions, substitute BXRED$ for BRRED$.

13.3.2.4. SEQUENTIAL WRITING OF DATA BLOCKS (BWRIT$ and B$MOVE)

Purpose:

Sequentially writes a block of data (currently residing at the block location returned in register A0 after execution of an open or write request) into an output or in/out file.

```
L,U    A0,pktaddr
LMJ    X11,BWRIT$
```

Description:

Register A0 is loaded with the address of the following one-word packet when the linkage is executed:

H1	H2
0	FCT-addr

where:

FCT-addr Address of the FCT (see 13.5.1) for the block being written.

If the block of data to be written resides at some location other than the one returned to register A0, a block move operation is required. To initiate this operation, the following linkage must be coded preceding the BWRIT$ calling sequence:

```
L,U    A0,pktaddr
LMJ    X11,B$MOVE
```

Register A0 is loaded with the address of the following one-word packet when the linkage is executed:

H1	H2
move-from-area-word-count	move-from-area-addr

where:

move-from-area-word-count Specifies the length of the area into which block is written.

move-from-area-addr Specifies the location of the area into which block is written.

The linkages and packets for the BWRIT$ and, if required, the B$MOVE subroutines, may be generated by the procedure call:

```
BWRIT   FCT-addr[move-from-area-word-count,move-from-area-addr]
```

13.3.2.5. RANDOM WRITING DATA BLOCKS (BRWRT$ and B$MOVE)

Purpose:

Writes randomly a fixed-length block of data (currently residing at the location returned to register A0 after the execution of an open or write request) into a specific block position of an output or in/out mode FASTRAND-formatted mass storage file.

Format:

```
L,U    A0,pktaddr
LMJ    X11,BRWRT$
```

Description:

Register A0 is loaded with the address of the following two-word packet when the linkage is executed:

	H1	H2
Word 0	0	FCT-addr
1	block-nbr	

where:

FCT-addr — Address of the FCT (see 13.5.1) for the file being processed.

block-nbr — Block number position of file into which block of data is written.

If the block of data to be written resides at some location other than the one returned in register A0, a move operation is required. To initiate this operation, the following linkage must be executed preceding the BRWRT$ calling sequence:

```
L,U    A0,pktaddr
LMJ    X11,B$MOVE
```

Register A0 is loaded with the address of the following one-word packet when the linkage is executed:

H1	H2
move-from-area-word-count	move-from-area-addr

where:

move-from-area-word-count — Specifies the length of area into which the block of data is moved.

move-from-area-addr — Location of the area into which block of data is moved.

The linkages and packets for the BRWRT$ and, if required, the B$MOVE subroutines, may be generated by the procedure call:

```
BRWRIT    FCT-addr    block-nbr[move-from-area-word-count,move-from-area-addr]
```

13.3.2.6. WRITING HARDWARE EOF MARKS (BMARK$)

Purpose:

Writes an EOF mark on an output tape file processed in the output mode.

Format:

```
L,U    A0,pktaddr
LMJ    X11,BMARK$
```

This linkage and packet may be generated by the procedure call:

```
BMARK    FCT-addr(1)    ...FCT-addr(n)
```

Description:

Register A0 is loaded with the address of the following one-word packet when the linkage is executed:

H1	H2
0	FCT-addr

where:

FCT-addr — Address of the FCT (see 13.5.1) for the file being processed.

13.3.2.7. CLOSING CURRENT TAPE REEL (BCLOR$)

Purpose:

Closes the current tape reel assigned to an input, output, or in/out mode file and to initiate a new tape reel for that file.

Format:

```
L,U     A0,pktaddr
LMJ     X11,BCLOR$
```

This linkage and packet may be generated by the procedure call:

```
BCLOSE   'REEL'   FCT-addr[,option]
```

Description:

Register A0 is loaded with the address of the following one-word packet when the linkage is executed:

T1	S3	H2
0	[option]	FCT-addr

where:

option — Applicable only to tape files. If the N option (23_8) is specified, the tape is rewound; if omitted, do not rewind the tape.

FCT-addr — Address of the FCT (see 13.5.1) for the file being closed.

13.3.2.8. CLOSING FILE CONTROL TABLES (BCLOF$ and BREL)

Purpose:

Closes the FCT for an input, output, or in/out file and releases its pool of buffers.

Format:

```
L,U     A0,pktaddr
LMJ     X11,BCLOF$
```

This linkage and packet may be generated by the procedure call:

```
BCLOSE   FILE   FCT-addr[option]
```

Description:

Register A0 is loaded with the same one-word packet as used for the BCLOR$ subroutine (see 13.3.2.7).

If the FCT (see 13.5.1) is for a tape file and the file is no longer needed for the run, the file should be freed dynamically with a CSF$ request (see 4.8.1). The code to set up linkage and packet for the BCLOF$ subroutine and to set up the instructions necessary to release the tape file, are generated by the following procedure call:

```
BREL   FCT-addr
```

This procedure call also generates both test and jump instructions which prevent the calling of the BCLOF$ subroutine if the file is already closed.

The BCLOF$ subroutine references the mass storage EOF address located in word 21 of the FCT and records this value in MBBI (word 12) of the respective file's master file directory main item. Execution of this request is allowed only for catalogued mass storage files. This feature is also available directly to the user through the BBEOF$ or MSCON$ requests (see 13.3.2.9 and 22.3.10, respectively).

13.3.2.9. CHANGING BLOCK BUFFERING EOF SECTOR ADDRESS (BBEOF$)

Purpose:

References the EOF address located in word 21 of the FCT (see 13.5.1) and records this value in MBBI (word 12) of the respective file's master file directory main item.

Format:

```
L,U    A0,pktaddr
ER     BBEOF$
```

where:

pktaddr is the address of the one-word packet:

```
+0,FCT-addr
```

Description:

Excecution of this request is allowed only for catalogued mass storage files. An additional executive request (ER MSCON$) is available to not only change the EOF address but also block size and item size (see 22.3.10).

13.3.3. LAYOUT FOR DATA BLOCKS

The data blocks processed by the BBP are recorded on tape or FASTRAND-formatted mass storage and may be either fixed or variable in length. Blocks written on FASTRAND-formatted mass storage have a control to specify the size of variable-length blocks. The layout of fixed-and variable-length data blocks contained on tape and FASTRAND-formatted mass storage are shown in Figure 13–1.

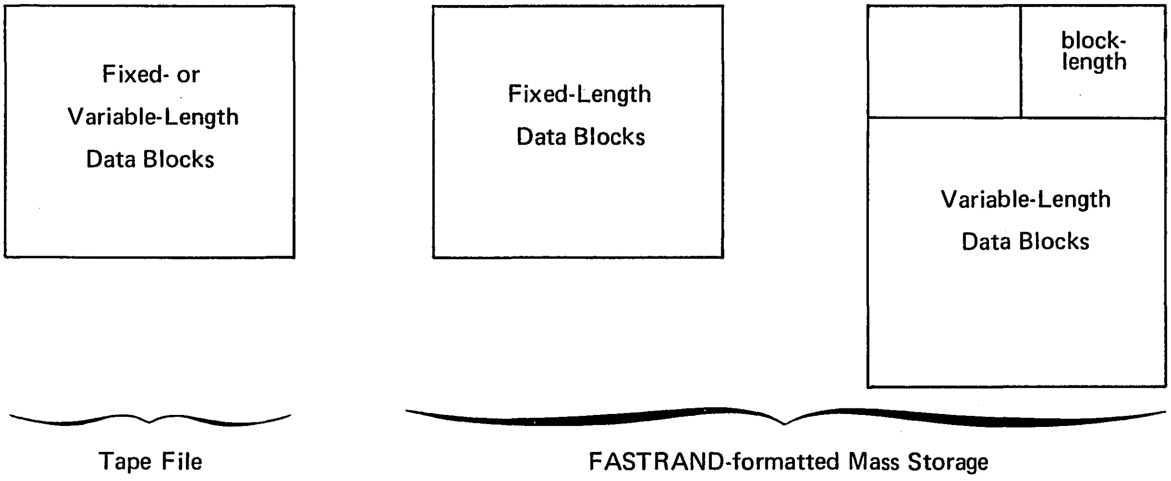


Figure 13-1. Data Block Layout

13.3.4. TRANSFER OF CONTROL FROM BBP TO USER'S PROGRAM

Control is returned to the user's program after a request for the initiation of a BBP subroutine has been executed, or if an abnormal or error condition has been detected (see 13.8 for error processing). When control is returned, related information is also returned to the user by means of an access word which is loaded into register A0. The nature of the information depends on the type of subroutine from which control was returned and the mode in which the subroutine performed its function.

On return from each open or read function performed in the input or in/out mode, the following access word is loaded into register A0:

H1	H2
word-count-of-block-read	addr-of-block-read

On return from each open, write, mark, or close reel function performed in the output or in/out mode, the following access word is loaded into register A0:

H1	H2
word-count-of-next-block-to-write	addr-of-next-block-to-write

The word count specified in H1 of register A0 is either a fixed-length block size as defined by the user or, as in the case of a variable-length file, the maximum area available in the buffer. When a file is opened in the in/out mode, the data block pointed to by the contents of register A0 is the first block that is written out on the next write request processed. This permits the file's label block to be easily updated.

When a BBP subroutine is used to close a catalogued file, the block size specified in register A0 is saved in the master file directory if the size pertains to a fixed-length block. The complement of the block size is saved if block size is variable in length. This information is automatically retrieved and used by the executive when a subsequent opening is required and the block size is not specified.

13.4. HANDLING DATA FILES AT THE ITEM LEVEL

Data handling at the item level is accomplished by a set of reentrant subroutines contained in the item handler. Through the use of these subroutines, the user can write items into a data file residing on an external storage device, read items from an existing data file residing on an external storage device, or modify an existing data file residing on a mass storage device without the necessity of recreating the file. The modes of operations in which the subroutines perform these functions are output (write), input (read), and in/out (read/write), respectively. In addition to controlling the internal manipulation of data files, the item handler also performs functions such as

- blocking and deblocking of data items;
- referencing the format definitions required to create or validate items written into or read from a data file;
- transferring items between the user-defined storage areas and the buffers assigned to the file;
- maintaining those portions of the FCT required for item processing and error handling;
- referencing the block buffering package when it is required to read in, write out, or swap buffer areas; and
- referencing the tape handler routines for processing end-of-reel and tape swapping functions.

13.4.1. SUBROUTINES OF THE ITEM HANDLER

The subroutines of the item handler are divided into two functional categories, output and input. These subroutines are called upon to perform data handling functions at the input, output, and in/out modes of operation. Some of the functions performed are common to all three modes while others are unique to a specific mode. For example, regardless of the mode in which a file is processed, it must be opened before the file can be processed. An exclusive read function, however, is unique to files processed in the in/out mode of operation. The functions performed by the input and output subroutines and the modes in which they are used are shown in the following table:

Function	Mode		
	In	Out	In/Out
Initialize FCT To Open File	√	√	√
Sequential Read	√		√
Random Read	√		√
Exclusive Random Read			√
Sequential Write		√	√
Random Write		√	√
Buffer Read and Buffer Write	√	√	
Closing Tape Reel	√	√	
Closing File	√	√	√
Releasing File and Device	√	√	√

13.4.1.1. INPUT SUBROUTINES

The input subroutines function in the input or in/out modes and provide for the reading of existing data files residing on an external storage medium. These subroutines are concerned only with the manipulation of the data requested by the object program. If a format definition is provided for the data being read (see 13.5.3), then the subroutine validates the format of the data prior to presenting the data to the object program. Otherwise, format validation does not take place.

13.4.1.2. OUTPUT SUBROUTINES

The output subroutines function in the output or in/out modes and provide for the creation (writing) of files on an external storage medium. These subroutines are concerned only with the manipulation of the data presented by the object program.

Prior to writing this data into the output files, the subroutines reformat labels, blocks, end-of-reel, and end-of-file sentinels in accordance with their respective format definitions (see 13.5.3) if given. Otherwise, the data is processed as originally presented by the object program.

13.4.2. DATA FILE MANIPULATION AT ITEM LEVEL

The manipulation of data files at the item level is accomplished by the subroutines of the item handler. These subroutines are initiated by the execution of the appropriate linkage coded into the object program or by the procedure call coded into the user's program. Both methods are described in 13.4.2.1 through 13.4.2.11.

13.4.2.1. INITIALIZING AN FCT FOR SUBSEQUENT ITEM HANDLING

Purpose:

Initiates or opens the FCT (see 13.5.1) for the reading or writing of subsequent data items for data files processed in the input, output, and in/out modes of operation.

Format:

```
L,U     A0,pktaddr
LMJ     X11,IHOPN
```

Description:

This linkage and the packet may be generated by the procedure call:

```
[label]   OPEN   'mode'   FCT-addr(1)[,'option']...FCT-addr(n)[,'option']
```

Register A0 is loaded with the address of the following one-word packet when the linkage is executed:

S1	S2	S3	H2
0	mode	option	FCT-addr

where:

mode — Specifies the mode of file processing. Entries are:

2_8 — Input mode file, forward operation (applicable to tape read operations)

3_8 —Input mode file, reverse operation (applicable to tape read operations)

40_8—Output file mode (applicable to tape write operations)

41_8—In/out file mode (applicable to read/write operation on FASTRAND-formatted mass storage files)

option — Specifies conditions or constraints that must be considered for the particular mode of file processing performed. Entries are:

E — Opens FASTRAND-formatted mass storage file for extension. Applicable to files processed in in/out mode. Permits user to write items after the specified EOF location of the file. (Applicable to FASTRAND-formatted files only.)

N — Inhibits tape files from being rewound prior to the initialization of the FCT. Applicable to files processed in the input or output modes. Not applicable to files processed in the input, reverse operation, or in/out modes of operation.

If this option is omitted, the tape files are rewound prior to the initialization of the FCT if the input or output modes of the file processing are specified. Opens FASTRAND-formatted mass storage file for updating if the in/out mode is specified. (Updating permits user to update each item in the file.)

FCT-addr — Address of the FCT to be intialized.

Interpretation of the parameters specified in the procedure call is the same as that specified for the packet word. The manner for coding the mode parameter differs in that the procedure call uses a mnemonic rather than a numeric entry. The following list correlates the two; mnemonic entries are coded as shown.

Packet entry	Procedure Call
2_8	INPUT
3_8	REVRSE
40_8	OUTPUT
41_8	IN/OUT

There are no restrictions on the number of OPEN requests which may be coded to initialize FCT's for input, output, or in/out modes of operation. However, only one request may be selected by the program to perform the actual function.

The manner in which label blocks are handled depends upon the mode of file processing and whether or not format definitions have been provided. See 13.6 for a discussion of label handling and 13.5.3 for format definitions.

13.4.2.2. SEQUENTIAL READING OF DATA ITEMS

Purpose:

Reads from the input buffer the next consecutive item recorded in the data file being processed in the input or in/out modes of operation.

Format:

```
L,U    A0,pktaddr
LMJ    X11,IHRD
```

This linkage and its packet may be generated by the procedure calls:

```
[label]   READ   FCT-addr   EOF-exit
```

or

```
[label]   READ   FCT-addr   move-length,move-addr   EOF-exit
```

Description:

Register A0 is loaded with the address of the following three-word packet when the linkage is executed:

		H1	H2
Word	0	0	FCT-addr
	1	move-length	move-addr
	2	0	EOF-exit

Word 0

FCT-addr — Address of the FCT (see 13.5.1) for the file being processed.

Word 1

move-length — Specifies the size of the item read. Must be specified for all spanned items; all fixed- or variable-length items move from the input buffer to the user's area for processing, or when the item is read from a file processing in the in/out mode of operation. Entries are:

0	Causes the entire item to be moved from buffer area to user's area for processing.
nbr-of-words	Specifies a specific number of words within the item to be moved from the buffer area to the user's area for processing. The move length parameter may be omitted only when the item does not need to be moved from the buffer to the user's area for processing.

move-addr — Specifies the address in the user program to which the item read is moved. This parameter is coded when the move-length parameter is specified.

Word 2

EOF-exit — Specifies the address in the user's program to which control is returned when an EOF condition is detected.

Interpretation of the parameters specified in the procedure calls is the same as that specified for the packet words.

The first format shown for the procedure call cannot be used for reading items that are spanned (item which overlaps blocks) or which must be moved from the input buffer to the user's area for processing. If the next item requested by this call is not in the buffer, another block is read into the buffer and the first item is obtained from it. The user may then process this block without removing it from the buffer.

The second format shown for the procedure call is used when reading spanned items as well as items of fixed- and variable-length. If the item is spanned, several requests to the input medium from the item handler are necessary to obtain the entire item.

Before the item read is presented to the user, the item handler interrogates the format definitions for items (see 13.5.3) to validate the item. If a format definition has not been specified, the item is presented to the user unaltered.

13.4.2.3. RANDOM READING OF DATA ITEMS

Purpose:

Reads a specific data item from a file residing on FASTRAND-formatted mass storage. File must consist of fixed-length items and blocks and is processed in the input and in/out modes of operation.

Format:

```
L,U     A0,pktaddr
LMJ     X11,IHRDRN
```

This linkage and the packet may be generated by the procedure call:

```
[label]   READRM   FCT-addr   move-length,move-addr   item-nbr
```

Description:

Register A0 is loaded with the address of the following three-word packet when the linkage is executed:

	H1	H2
Word 0	0	FCT-addr
1	move-length	move-addr
2	0	item-nbr

Word 0

FCT-addr — Address of the FCT (see 13.5.1) for the file being processed.

Word 1

move-length — Specifies the size of the item to be moved from the input buffer to the user's area for processing. Entries are:

0	Causes the entire fixed-length item to be moved to the user's area for processing.
nbr-of-words	Specifies a specific number of words within the item to be moved to the user's area.

move-addr — Specifies the address in the user's program to which the item is moved for processing.

Word 2

item-nbr   Specifies the position of the requested item in the file relative to the beginning of the file.

Interpretation of the parameters specified in the procedure call is the same as that specified for the packet words.

The block containing the requested item is read into the input buffer from the input medium. Before the item is moved to the user's area for processing, the item handler interrogates the format definition for items (see 13.5.3) to validate the item. If a format definition has not been specified, the item is presented to the user unaltered.

EOF detection is not necessary since items residing in files being read randomly are read many times over in any order; these items therefore have no logical EOF.

Those items read randomly and processed in the in/out mode are obtained with an exclusive read function (see 13.4.2.4).

13.4.2.4. REQUESTING EXCLUSIVE RANDOM READING OF DATA ITEMS

The exclusive reading of data items is a feature of the executive which permits the user to request exclusive use of a specific fixed-length item. The item must be contained in a FASTRAND-formatted mass storage file and must be processed in the in/out mode of operation. Once initiated, the item is interlocked from all references until it is released by the next reference to the file.

The exclusive read feature is initiated by execution of the same linkage specified for a normal read random request (see 13.4.2.3). The packet for this linkage remains basically the same except for H1 of word 0 which contains the numeric value 1, indicating that the packet pertains to an exclusive read request.

If the user wishes to initialize the exclusive read random request by means of a procedure call, he does so by prefixing the request name of the normal read random procedure call (see 13.4.2.3) with the letters EX. All other parameters of the procedure call are the same.

If the user holds an item for a long period of time, a timeout occurs and the file error exit specified in the FCT is taken. The error code and the reentry code that the user must use to return control to the item handler are available in register A1.

H1	H2
error-code (10_8)	EX-READ-addr

Error processing is described in 13.8.

13.4.2.5. SEQUENTIAL WRITING OF DATA ITEMS (IHWRT)

Purpose:

Writes into the output buffer the next consecutive item to be recorded in the file being processed in the output or in/out mode of operation.

Format:

```
L,U    A0,pktaddr
LMJ    X11,IHWRT
```

This linkage and the packet may be generated by the procedure call:

```
[label]   WRITE   FCT-addr   [move-length]

or

[label]   WRITE   FCT-addr   move-length,move-addr
```

Description:

Register A0 is loaded with the address of the following two-word packet when the linkage is executed:

	H1	H2
Word 0	0	FCT-addr
1	move-length	move-addr

Word 0

FCT-addr — Address of the FCT (see 13.5.1) for the file being processed.

Word 1

move-length — Specifies the size of the item to be written. Must be specified if file is processed in the in/out mode or if the item size differs from that defined in FCT. Entries are:

0	Specifies that item size is the same as that previously defined in FCT.
nbr-of-words	Specifies the specific number-of-words (size) in the item to be written when it differs from that defined in FCT. If user has placed the item in a predefined area, this field specifies an item of less or greater size than defined in FCT. If this parameter is omitted, it specifies that the item to be written is the same as defined in FCT and that the user has already placed item in a predefined area.

move-addr — Specifies the address of the item to be written. Must be specified for in/out mode processing. May be omitted if user has placed item in a predefined area.

The manner in which an item is written into the buffer depends on whether item size is defined as fixed or variable in the file's FCT. If item size is defined as being fixed, the balance of the item area is zero filled for items smaller than that specified in the FCT. If item size is defined as variable, then the number of words specified in the move-length parameter is written into the item area. If item length is greater than that defined for the item, the item is either spanned or written into the next item block; that is, if the block size for items is defined as fixed, the item is spanned between two blocks. If the block size is defined as variable, the current buffer is written and the entire item is placed in the next buffer.

13.4.2.6. RANDOM WRITING OF DATA ITEMS

Purpose:

Writes a fixed-length item in a specific position within a file residing on mass storage. The file is processed in either the output or in/out modes of operation.

Format:

```
L,U    A0,pktaddr
LMJ    X11,IHWTRN
```

This linkage and the packet may be generated by the procedure call:

```
[label]  WRITRM  FCT-addr  move-length,move-addr  item-nbr
```

Description:

Register A0 is loaded with the address of the following three-word packet when the linkage is executed:

	H1	H2
Word 0	0	FCT-addr
1	move-length	move-addr
2	0	item-nbr

Word 0

FCT-addr — Address of the FCT (see 13.5.1) for the file being processed.

Word 1

move-length — Specifies the size of the item to be moved from the area specified by the move-addr parameter. The item moves into the buffer to be recorded as a random item in the file that is being prócessed. Entries are:

0	Causes the entire item of the size previously defined in the FCT to be read into the buffer.
nbr-of-words	Specifies a specific number of words within the item (less than the item size defined in FCT) to be read into the buffer.

move-addr — Address of the item to be written.

Word 2

item-nbr — Specifies the position that the item will occupy in the file relative to the beginning of the file (item number 0).

Interpretation of the parameters specified in the procedure call is the same as that specified for the packet words.

The block containing the item is located and retrieved from the output medium by means of the item address specified by the user. The item is then moved from the location specified by the move address to its position in the buffer. The buffer is rewritten to its original position in the file. Prior to writing the item in the buffer, the item handler structures the item according to the format definition for items (see 13.5.3). If a format definition has not been specified, the item is written into the file unaltered.

In cases where the size of the item written is less than that defined for fixed-length items in the FCT, the unused portion of the item is either zero filled or left undisturbed. If the file is processed in the output mode, the unused portion is zero filled. The unused portion of the item is left undisturbed when the file is processed in the in/out mode.

13.4.2.7. SEQUENTIAL AND RANDOM READ/WRITE REQUESTS

Sequential and random read/write requests pertain to items contained in files residing on mass storage and processed in the in/out mode of operation. The item handler uses the same routines as those specified for the sequential reading and writing of data items (see 13.4.2.2 and 13.4.2.5, respectively) and the random reading and writing of data items (see 13.4.2.5 and 13.4.2.6, respectively). The difference is in the logical sequence in which the requests for these subroutines are executed. For example, the following logic is implied when sequential read/write requests are executed:

- A read request makes the next item available for processing.
- A write request following a read request rewrites the item.
- Consecutive read requests without intervening write requests make available consecutive items which are not rewritten.
- Consecutive write requests without intervening read requests cause the writing of successive items.
- A read request following several consecutive write requests obtains the item which sequentially follows the last item written into the file.

13.4.2.8. READING AND WRITING THE CURRENT BUFFER CONTENT

Purpose:

Manipulates data into or out of the buffer currently assigned to the file being processed. The specific function performed is based upon the mode (input or output) in which the file is processed.

Format:

```
L,U    A0,pktaddr
LMJ    X11,IHDRN
```

This linkage and packet may be generated by the procedure call:

```
[label]   DRAIN   FCT-addr(1)   ...FCT-addr(n)
```

Description:

Register A0 is loaded with the address of the following one-word packet when the linkage is executed:

H1	H2
0	FCT-addr

where:

FCT-addr   Is the Address of the FCT (see 13.5.1) for the file being processed.

When the IHDRN subroutine is executed in the input mode (read operation), the current buffer in use is relinquished and another block of data is acquired from the file and read into the buffer. Any unused items in the buffer are ignored. If the buffer has a spanned item, several buffers are bypassed to obtain the next valid item. The next read request for the file being processed is directed to the first valid item in the newly acquired buffer.

An IHDRN subroutine executed in the output mode (write operation) causes the contents of the current buffer to be immediately written into the file being processed. A request to initiate the IHDRN subroutine is ignored if the buffer is empty. The method in which the buffer is written is based on the following:

- If the block size is specified as variable in the FCT, the buffer is truncated to the last valid item.
- If the block size is specified as fixed, the buffer is not truncated. An end-of-data code (777777777777_8) recorded following the last valid item.

Regardless of which method is used, the format definition for blocks (see 13.5.3) is used to complete the block structure before it is written. If a format definition is not specified, the block is written unaltered.

13.4.2.9. CLOSING A TAPE REEL

Purpose:

Terminates the reading of data items from an input tape reel or the writing of data items onto an output tape reel assigned to the file being processed. It also initiates, automatically, the reading or writing of subsequent tape reels assigned to the same file.

Format:

```
L,U    A0,pktaddr
LMJ    X11,IHCLR
```

The linkage and packet may be generated by the procedure call:

```
[label]   CLOSE   'REEL'   FCT-addr(1)[,'option']   ...FCT-addr(n)[,'option']
```

Description:

Register A0 is loaded with the following one-word packet when the linkage is executed:

S1	S2	S3	H2
0	0	option	FCT-addr

where:

option — Specifies conditions or constraints that must be considered when closing a tape reel. Entries are:

L — Rewinds tape with interlock after closing is completed.

N — Inhibits tape reel from being rewound after closing is completed.

If these options are omitted, the tape file is rewound without interlock after closing is completed.

FCT-addr — Address of the FCT (see 13.5.1) for the file being processed.

Interpretation of the parameters specified in the procedure call is the same as that specified for the packet word.

Closing an input tape reel releases all buffers to the buffer pool, initiates the tape rewind as specified by the option parameter, and initiates the reel swapping procedure. The user's label access word, if one was specified, is loaded into register A0 when control is returned to the user's program. If a user's label was not specified, register A0 is zero filled.

When an output tape reel is closed, any unrecorded items in the buffer are constructed according to the format definition for blocks (see 13.5.3) and recorded on tape. If a format definition has not been given for end-of-reel sentinels, a block is generated and written, and then the EOF mark is written. A final EOF mark is recorded before the rewind option is executed and before tape reel switching procedure is initiated.

The location of the next item area in the buffer is loaded in control register A0 upon return of control to the user's program. The handling of labels and sentinels is as specified in 13.6.

13.4.2.10. CLOSING A DATA FILE

Purpose:

Terminates the reading or writing of data files processed in the input, output, or in/out modes of operations.

Format:

```
     L,U    A0,pktaddr
     LMJ    X11,IHCLF
```

This linkage and the packet may be generated by the procedure call:

```
     [label]   CLOSE   'FILE'   FCT-addr(1)[,'option'] ···FCT-addr(n)[,'option']
```

Description:

Register A0 is loaded with the address of the following one-word packet when the linkage is executed.

S1	S2	S3	H2
0	0	option	FCT-addr

where:

option — Specifies the tape rewind procedure to be initiated when the data file resides on magnetic tape. The option field is ignored when the data file resides on mass storage. Entries are:

L — Rewinds tape with interlock after closing is completed.

N — Inhibits tape reel from being rewound after closing is completed.

If the option is omitted, the tape file is rewound without interlock after closing is completed.

FCT-addr — Address of the FCT (see 13.5.1) for the file being processed.

Interpretation of the parameters specified in the procedure call is the same as that specified for the packet word.

There are no restrictions on the number of close requests coded to reference an individual input, output, or in/out file, but only one request is selected by the program to perform the actual function. A single request can be used to close all files in the mode in which they are processed.

When an output file is being closed, the output buffer is constructed according to the format definition for blocks (see 13.5.3) and written on the output recording device. If a format has not been given for EOF sentinels, the block is generated and written, and then an EOF mark is written. A final EOF mark is recorded on the output file before the rewind option is initiated. If the file resides on disc or FASTRAND-formatted mass storage, the rewind option is ignored and the next address to be written is saved in the item directory.

If a close request for an in/out file is executed with a format definition for EOF sentinels specified, the sentinel is written according to the following rules:

- The sentinel is not updated if the original sentinel was not overwritten and if recording in the file was not extended beyond the sentinel.
- The orginal sentinel is erased and a new one is recorded after the last block in the file if the original sentinel was not overwritten but recording was extended beyond the sentinel.
- A new sentinel is recorded after the last block in the file if the original sentinel was overwritten.

13.4.2.11. RELINQUISHING USER PROGRAM ASSOCIATION WITH DATA FILE

Purpose:

This request is used to immediately relinquish a program's association with a data file that is processed in the input, output, or in/out mode of operation. The medium on which the file resides is also released by this request.

Format:

```
    [label]    RELESE    FCT-addr(1) [,option] ...FCT-addr(n) [,option]
```

where:

FCT-addr	Address of the FCT (see 13.5.1) for the file being released.
option	The physical assignment for the file can be held by coding the letter S.

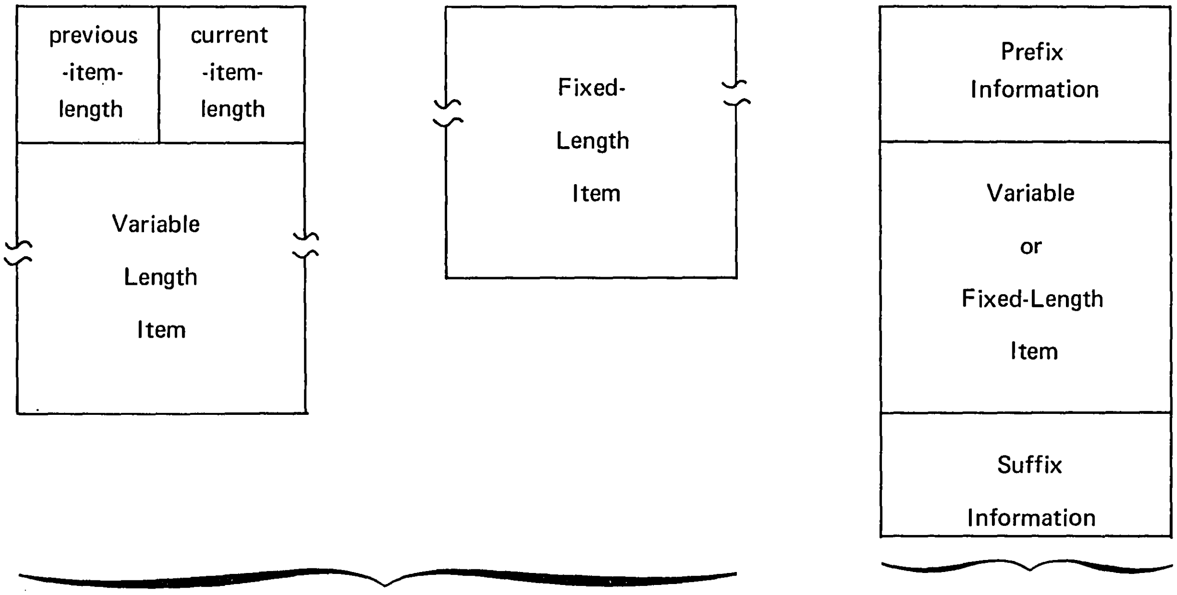
Description:

Disposition of released file is based upon the cataloguing options specified on the @ASG control statement (see 3.7.1) for devices assigned to the file and the option specified on the RELESE request. Before disposition occurs, the item handler calls upon the close file subroutine (see 13.4.2.10) to close the file if the file is not already closed. If the file resides on tape, the tape reel is rewound with interlock when the close subroutine is initiated. The user, therefore, is not required to issue a separate close request to terminate the file if the file is to be terminated by the RELESE request. Once closed, the file can only be reopened by the program if it is a catalogued file.

13.4.3. LAYOUT FOR SINGLE AND BLOCKED ITEMS

The items processed by the item handler are recorded on tape and FASTRAND-formatted mass storage files as single or blocked items. The layout for each type is determined by whether the items are defined as being fixed or variable in length and if the item has been given a format definition. The processing of a variable-length item requires that the user specify a control word which defines the length of the item. If a format definition (see 13.5.3) has been specified for the item, the control word is embedded within the definition. Fixed-length items are of a given size and therefore have no control information.

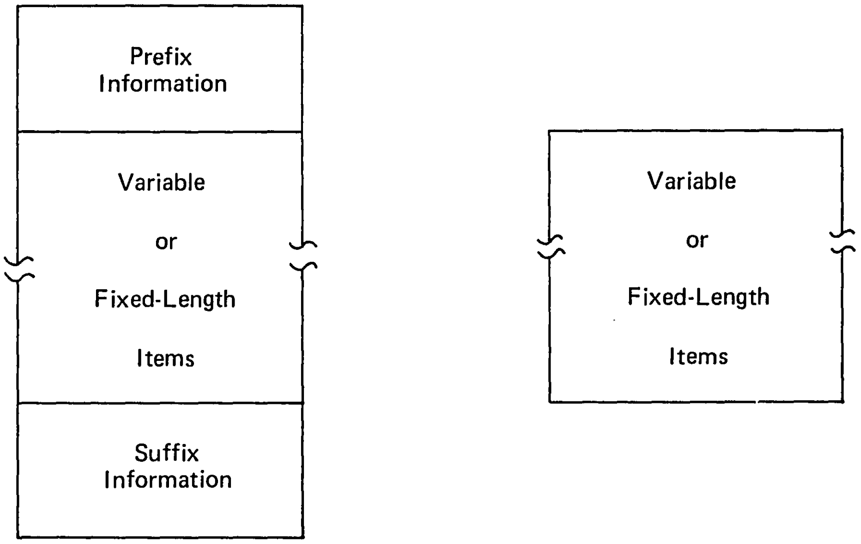
The layout of single and block items are shown in Figures 13–2 and 13–3, respectively. These examples illustrate both formatted and unformatted layouts.



Unformatted Single Item

Formatted Single Item

Figure 13-2. Single Item Layout


Prefix Information
Variable or Fixed-Length Items
Suffix Information
Variable or Fixed-Length Items

Formatted Blocked Item

Unformatted Blocked Item

Figure 13-3. Blocked Item Layout

13.4.4. TRANSFER OF CONTROL FROM ITEM HANDLER TO USER'S PROGRAM

Control is transferred from the item handler to the user's program after a request or procedure call for one of the item handler subroutines has been processed or when an end-of-reel or end-of-file condition is encountered. When control is returned to the user's program, information related to that return is provided to the user in the form of an access word which is loaded into registers A0 or A1. The nature or interpretation of the access word depends on the specific function being performed and the mode in which that function is processed.

13.4.4.1. CONTROL TRANSFER DURING OUTPUT MODE PROCESSING

The access word loaded into register A0 after the processing of an open, write (sequential or random), or drain output subroutine request defines the buffer area into which the next item to be processed is placed. The size of the buffer area and its location are made available to the user program. The size of the area given depends on whether the user has defined items as being fixed or variable in length when he generated the FCT (see 13.5.1) for the file. For fixed-length items, the area given is equal to the item size. For variable-length items, the area given is equal to the current unused portion of the buffer. The format of the access word in register A0 is:

H1	H2
nbr-of-words	buffer-addr

Control is also returned to the user's program when the tape device assigned to the file physically reaches the end of a reel. This only occurs, however, if the user has specified an end-of-reel exit code in the FCT for the file. Otherwise, close reel procedures are automatically initiated by the item handler. When the end-of-reel exit is taken, register A1 is loaded with an access word which provides the location of the FCT and the address to which the user program returns control after initiating his close procedures in order to reenter the main execution sequence. Once control is returned to the user program, the FCT address in H1 of register A1 must be reset to 0. The format of the access word in register A1 is:

H1	H2
FCT-addr	return-addr

The location of the next item area in the buffer is given in register A0 after the user program has initiated the close reel procedures.

13.4.4.2. CONTROL TRANSFER DURING INPUT MODE PROCESSING

When control returns to the user program after a read sequential or random request, register A0 is loaded with the access word which defines the actual size and buffer location of the current item read. The format of register A0 is:

H1	H2
nbr-of-words	buffer-addr

The buffer address specified provides the user with the location at which the image of the current item resides. If the read function processed required a move function, then a second image of the current item is also made available to the user. This second image resides at the location specified by the address parameter of the procedure call requesting the read function. The item image in the specified buffer area, however, remains available to the user program until the next request for a read function is issued.

The item image is not made available to the user if the read request issued pertained to a spanned item. The access word returned to register A0 in this case contains only the size of the current item. The buffer address is zero filled. The format of register A0 is:

H1	H2
nbr-of-words	0

Register A1 is used when a read function returns control to the user program through an EOF or end-of-reel exit; that is, when an EOF block or end-of-reel block is read and validated, the routine returns control to the user program by using the appropriate EOF or end-of-reel exit address specified in the FCT (see 13.5.1) for the file. The address of the FCT in which the exit address exits is specified in H1 of register A1. This address is reset (zero filled) before the user returns control to the item handler. H2 of register A1 contains the address to which the user returns control after he has initiated his close procedures in order to reenter the main sequence of execution. If the user has not specified an end-of-record exit in the FCT, an automatic close reel procedure is performed and the tape is rewound with interlock. The length and location of the format block are loaded into register A0.

The EOF exit is also taken when an EOF mark is detected and the format definitions for EOF and end-of-reel sentinels have not been specified.

13.4.4.3. CONTROL TRANSFER DURING IN/OUT MODE PROCESSING

Since the functions performed in the in/out mode are basically a combination of the functions performed in the input and output modes (see 13.4.4.1 and 13.4.4.2), registers A0 and A1 serve similar duties in returning control to the user program.

For read/write functions, register A0 always contains an access word for the largest size item that can be written. If the file is being updated, this size represents the actual size of the item just read. Otherwise, it represents the size of the next area of the buffer into which the next item is written. The maximum item size permitted when extending a file is the maximum item size defined by the user.

13.5. INTERFACING THE DATA HANDLING ROUTINES WITH THE USER'S PROGRAM

Prior to referencing a data file for processing, the user must establish a means of communications between his program and the executive data handling routines used to access and manipulate that file. He accomplishes this through the generation of a file control table (FCT) which specifies the information necessary for interfacing the user's program with the data handling routines and the data handling routines with one another. If the user program requires the handling of data files at the block buffering level, he is then responsible for setting up a pool of buffers for those files. He is also responsible for providing the format definition describing the physical organization of those files requiring label and sentinel processing to the executive.

13.5.1. FILE CONTROL TABLES

File control tables (FCT) provide the basic interface between the user's program and the data files located on the peripheral storage devices of the system. An FCT is generated and constructed within the user's program for each data file reference within that program. The FCT's provide the executive with user-specified information, such as the name and type of file requested, the mode in which the file is processed, the amount of I/O control required to access and manipulate the file, and all information pertinent to the physical organization of the file and the handling of error conditions which may be encountered. The data handling routines use the FCT's to communicate with the user's program.

13.5.1.1. FILE CONTROL TABLE FORMAT

The format as well as the length and complexity of an FCT is determined by the level at which the file maintained by this table is processed A file processed at the item level requires the highest level of interface and therefore must have an FCT containing all three functional areas: I/O control, block buffering, and item control. If the file is handled at the block level, then only the I/O control and the block buffering areas need to be generated. The I/O control area, however, is always generated regardless of the level of file processing. The format of a complete FCT is shown in Figure 13–4.

The I/O control area comprises the first six words (0 through 5) of an FCT. It contains the format of the appropriate request packet required for communicating with the executive I/O device handlers. The I/O request packets specified in this area are limited to those handlers associated with magnetic tape and FASTRAND-formatted mass storage devices. (I/O request packets are described in detail in Section 6.)

The block buffering area occupies words 6 through 24 of the FCT. This area provides a general means of file communications for the user and is the lowest level of program interface for data handling. The information in this section primarily concerns the internal control required by the subroutines of the block buffering package (see 13.3) for handling data at the block level. The user, however, provides the block buffering package with sufficient information to assure proper handling of the data blocks requested. This is accomplished through the execution of the file procedure call used to generate the FCT (see 13.5.1.2).

The item control area also provides a general means of file communications for the user and is considered the highest level of program interface. The information presented in this area provides the interface required for the internal control and handling of data at the item level. The information is used by the subroutines of the item handler (see 13.4) to manipulate items and to build data blocks from these items. In addition, the contents of this area are used, when necessary, to interface the item handler with the format definitions specified for file organization (see 13.5.3) and the block buffering package (see 13.4) for handling the data blocks generated.

Most of the fields in the FCT are zero filled prior to the opening or initializing of the FCT for a given file. The parameters enclosed in brackets are those specified by the user in the FILE procedure call which generates the FCT (these parameters are optional). If the user attempts to generate an FCT without using the FILE procedure call he must make certain that these parameters are provided and specified in their proper location before any reference is made to the FCT. The initial value set for these parameters depends upon the level at which data is handled. See 13.4 for data handling at item level; see 13.3 for data handling at block level. The parameters shown in italics are used for internal control purposes by their respective handlers and therefore require no further attention by the user.

Some of the data presented in the FCT is retained by the executive for future use when a file is closed. At the block level, the executive places the block size entry and, if the file resides on FASTRAND-formatted mass storage, places the end-of-file sector address in the main item of the master file directory for the catalogued file. The complement of the block size is used if the block size is variable. The item size is also retained if the data is handled at the item level. This information is automatically retrieved by the executive if the user fails to specify the entry on a subsequent opening of the FCT. The location that the information occupies in the master file directory item depends on the specific type of file catalogued (magnetic tape or FASTRAND-formatted mass storage). In either case, the block size is expressed as a negative value if it is considered as a variable, and the item size is always stated as 0 when it is variable. See Section 22 for information concerning the master file directory.

<table>
<tr><th></th><th></th><th>S1</th><th>S2</th><th>S3</th><th>S4</th><th>S5</th><th>S6</th><th></th></tr>
<tr><td>Word</td><td>0</td><td colspan="6">filename</td><td rowspan="6">I/O Control Area</td></tr>
<tr><td></td><td>1</td><td colspan="6">filename</td></tr>
<tr><td></td><td>2</td><td colspan="2"><i>used-by-executive</i></td><td>interrupt-activity-id</td><td colspan="3">interrupt-activity-start</td></tr>
<tr><td></td><td>3</td><td><i>status</i></td><td>function</td><td><i>AFC</i></td><td colspan="3">substatus</td></tr>
<tr><td></td><td>4</td><td>G</td><td colspan="2">word-count</td><td colspan="3">buffer-addr</td></tr>
<tr><td></td><td>5</td><td colspan="2">0</td><td colspan="4">drum-addr</td></tr>
<tr><td></td><td>6</td><td>open-flag</td><td>look-ahead-factor-(LAF)</td><td><i>file-mode</i></td><td colspan="3"><i>current-buffer-held-by-I/O</i></td><td rowspan="12">Block Buffering Area</td></tr>
<tr><td></td><td>7</td><td colspan="3">max-block-word-count</td><td>fixed-block</td><td><i>lock-flag</i></td><td><i>FASTRAND-flag</i></td></tr>
<tr><td></td><td>8</td><td colspan="3">[I/O-error-exit]</td><td colspan="3">[BBP-call-error-exit]</td></tr>
<tr><td></td><td>9</td><td colspan="3"><i>user-buffer-starting-addr</i></td><td colspan="3">[abnormal-error-exit]</td></tr>
<tr><td></td><td>10</td><td colspan="6">[sentinel-value]</td></tr>
<tr><td></td><td>11</td><td colspan="3">addr-of-buffer-packet</td><td colspan="3"><i>activity-reentry-addr</i></td></tr>
<tr><td></td><td>12</td><td colspan="6"><i>cumulative-block-count</i></td></tr>
<tr><td></td><td>13</td><td colspan="3"><i>addr-of-last-queued-buffers</i></td><td colspan="3"><i>addr-of-first-queued-buffers</i></td></tr>
<tr><td></td><td>14</td><td colspan="3"><i>current-data-located-in-buffer</i></td><td colspan="3"><i>save-BBP-routine-return-addr</i></td></tr>
<tr><td></td><td>15</td><td colspan="3"><i>word-length-of-data-block</i></td><td colspan="3"><i>data-buffer-starting-addr</i></td></tr>
<tr><td></td><td>16</td><td><i>item-flag</i></td><td><i>frame-count</i></td><td><i>checkpoint/ restart</i></td><td>0</td><td><i>I/O-flag</i></td><td><i>queue-count</i></td></tr>
<tr><td></td><td>17</td><td colspan="3"><i>exclusive-read-return</i></td><td colspan="3"><i>relative-FCT-addr</i></td></tr>
</table>

Figure 13-4. File Control Table Format (Part 1 of 3)

	S1	S2	S3	S4	S5	S6	
Word 18	*sector-count*						Block Buffering Area
19	0	*tape-equipment*	*I/O-read*	0	*abnormal-lock*	*activity-switch-location*	
20	*highest-FASTRAND-addr*						
21	*FASTRAND-EOF-addr*						
22	[test-and-set]	[sentinel-position]	*read-option*	*user-reentry-addr*			
23	[mask-for-block-sentinel-check (all bits set if undefined)]						
24	[eight-word-register-save-indicator]						
25	*I/O-flag*	*debug-flag*	*do-I/O-flag*	[format-entry-name]			Item Control Area
26	*length-data-area-left-in-block*			*add-current-item-data-area*			
27	*nbr-of-words-left-to-process*			*nbr-of-words-requested-for-move*			
28	*length-left-in-block-pools-format*			*add-current-item-prefix*			
29	[end-of-reel-exit]			length-current-item-pools-format			
30	[fixed-item-size]			nbr-of-items-in-block			
31	block- flag	reel-flag	write-flag	read-flag	lock-flag	user-function	
32	T-S-flag	span -flag	EOF-flag	exclusive-read-addr			
33	mark-flag	label-format	item-format	block-format	EOR-format	EOF-format	
34	[user-label-words]						
35	[user-free-words]						

Figure 13-4. File Control Table Format (Part 2 of 3)

Word	S1	S2	S3	S4	S5	S6
36	three-word-packet					
37	for-block-buffering-requests					
38	from-item-handler					
39	user-device -error- exit			user-file-error-exit		
40	user-abnormal-exit			SDF-flag		
41	routine-return-location			routine-return-location		
42	item-prefix-length			item-suffix-length		
43	block-prefix-length			block-suffix-length		
44	routine-return-location					
45				diagnostic-dump-routine-location		

Item Control Area

Figure 13-4. File Control Table Format (Part 3 of 3)

13.5.1.2. GENERATING THE FCT

The FCT is generated by coding the procedure call FILE within the user program. Each parameter, with the exception of the first parameter ('filename', format), is identified to the executive by a title enclosed in quote marks. The order in which the parameter is specified, with the exception of the first parameter, is of no consequence. The subfields, however, must appear in the order shown. The format illustrated lists the parameters on separate coding lines only for clarity.

Format

```
[label] FILE 'filename',format;

              ['SIZE'[,block] [,item] ] ;

              ['POOL',link,laf] ;

              ['ERROR',file,device,abnormal] ;

              ['SENT',value,mask,position] ;

              ['LABEL',nbr-of-words,addr] ;

              ['FREEWD',nbr-of-words,addr] ;

              ['EOR',addr] ;

              ['REG',A]
```

Parameters:

'filename'

Specifies the external filename of the file for which the FCT is being built. The filename specified normally agrees with that specified in the @ASG control statement (see 3.7.1) or an internal filename equated to it by means of the @USE control statement (see 3.7.5).

format

Specifies the name (label) of the format definition entry point required to use this file. (See 13.5.3.1 for format definition.)

If no specific format is desired (file consists of items within blocks), the system-defined coding 'NULFOR' is specified in this subfield.

For a file to be handled at the block buffering level (see 13.3), the system-defined coding 'BBP' must be specified in this subfield.

For a file to be handled at the item level (see 13.4), by use of the LION format, the coding *LIONA or *LIONB is specified in this subfield. The coding 'LION' refers to a format element related to the UNIVAC 1107 system. It is provided for those users wishing to convert to the 1100 system format at the item level. The asterisk preceding this entry informs the executive that additional storage area must be reserved for the counters used by the LION format subroutines. The asterisk also causes the assembler to place a 1 in the item flag (S1 of word 16) of the FCT (see 13.5.1.1), LION sentinel (/./././) in word 10 of the FCT, and a sentinel mask (777777777777_8) in word 23 of the FCT when it processes the FILE procedure.

'SIZE'

Describes the item/block relationship for the file specified. If this parameter is omitted, the SIZE parameter saved in the file's master file directory item during cataloguing is used. If the file is not catalogued and the SIZE parameter is omitted, the assembler substitutes the buffer size less 3 for the block size and expects variable-length items when processing at the item level.

The SIZE parameter can be coded in any of the following formats:

(1) 'SIZE',block,item

(2) 'SIZE',*block,item

(3) 'SIZE',block

(4) 'SIZE',*block

Format 1:

Specifies fixed-length item processing in variable-length blocks. Items are recorded until the area remaining in a block is less than the size of the next item. The block is then written and a new buffer is acquired. The user may vary the size of the block by varying the number of fixed-length items in that block. This is accomplished by use of the drain request (see 13.4.2.8) which truncates the block before it is written.

Format 2:

Specifies fixed-length item processing in fixed-length blocks. Asterisk denotes that the value specified in block parameter is used for computation of block length in which an integral number of items are included. The computed block length constitutes the size for each block written into the file. The user may vary the number of items in the block by use of the drain request. Truncation of the block, however, does not occur for this format. This format must be specified when random file processing is used.

Format 3:

Specifies variable-length item processing in variable-length blocks. Items are recorded until the area remaining in block is less than the size of the next item. The block is then written and a new buffer is acquired. A 'BBP' specified in the format parameter results in the variable-length block processing for this format.

Format 4:

Specifies variable-length item processing in fixed-length blocks. Asterisk denotes that the value specified in block parameter is the actual size of the block. This format permits spanning (overlapping blocks with an item). The item handler spans blocks with an item when the item presented is larger than the block size specified or larger than the block size specified or larger than the area remaining in the block. This format specifies that fixed-length blocks are processed when the format parameter contains a 'BBP' entry.

block — Specifies the maximum number of words in the format definition (see 13.5.3) for each item and for the block.

item — Specifies the actual number of data words in each fixed-length item.

'POOL — Specifies the amount of buffering required for the file.

link — Specifies the address of the buffer control word (see 13.5.2).

laf — Specifies an estimate of the number of blocks to be read ahead for input files. The value specified is set to, and may not exceed, one less than the number of buffers in the buffer pool.

'ERROR' — Permits the user to specify which error conditions are to be processed. If this parameter is omitted, the item handler takes an ERR$ exit and no error message is printed. The standard system error procedure results in program termination after the appropriate error code and the active reference is recorded in register A1.

file — Specifies an address in user's program to which control is returned when a contingency pertaining to the operation of user files arises.

device — Specifies an address in user's program to which control is returned when a contingency pertaining to physical operation of a peripheral device occurs.

abnormal — Specifies an address in user's program to which control is returned when the following conditions are detected

- sentinel found,
- end-of-record information,
- end-of-tape, or
- end-of-mass-storage area

If data is being handled at the item level, control is returned to the abnormal location when a format error is discovered by a format subroutine.

'SENT' — Defines sentinel being looked for when reading a file at the block buffering level. (User writes his own sentinel blocks on output operations.) A sentinel check is not made if this parameter is omitted and the file is not catalogued. If this parameter is omitted and the LION format is specified, then standard LION sentinel values are generated as if the user had coded 'SENT', '/./◊/.' into his program.

value

Specifies up to six numeric or alphanumeric characters which the item handler uses in detecting a sentinel block. Alphanumeric entries are enclosed by quote marks.

mask

Specifies a mask against which the data word in the block is applied for determining a sentinel block. If this parameter is omitted or specified as zero, the item handler generates a mask of all 1 bits for full-word detection. When the mask is provided in the FCT, the block buffering package (BBP) does not check FASTRAND-formatted files for EOF to stop the input. The input is stopped only when a sentinel block is detected. When operating, strictly at the block buffering level, however, the sentinel block is determined by comparing the sentinel word in the FCT to the first word in each block in which the item flag (S1 of word 16 in FCT) is zero.

Position

Specifies the position of the data word within the block. This word is applied to the mask specified by the mask parameter for the purpose of determining the sentinel block.

'LABEL'

Defines the location and length of the user's label image. If this parameter is omitted, user label processing is not performed. This parameter is ignored by the block buffering package when processing is performed at the block buffering level, and at the item level if a format definition was not provided for a label block or if the definition of a label block did not provide for user label.

nbr-of-words

Specifies the number of words contained in the user's label.

addr

Specifies the address of the user's label.

'FREEWD'

Defines the 1107 LION label block area referred to as free words. If this parameter is omitted, free word processing is not performed. This operation can be used to extend the length of a label block to make the file compatible with EXEC II or EXEC I.

nbr-of-words

Specifies the number of words comprising the free word information.

addr

Specifies the address of the free word information.

'EOR'

Defines the end-of-reel exit. If this parameter is omitted, end-of-reel procedures are performed automatically. This parameter is ignored when processing is performed at the block buffering level.

addr

Specifies the address location in the user's program to which control is returned upon detection of an end-of-reel condition.

'REG'

Defines the manner in which the block buffering package handles the user's register set A1 through A5 and R1 through R3. If this parameter is omitted, the contents of the user's registers are saved in the eight-word buffer contained in the block buffering package.

A

Specifies the coding which saves or destroys the contents of the user's registers. The contents of the user's registers are saved in an eight-word buffer in the block buffering package when a 0, +0, or blank entry is made in this subfield. The contents of these registers are also saved in a user-defined buffer area providing the address of that buffer is specified in this parameter.

13.5.2. ESTABLISHING BUFFER POOLS (BPOOL$ AND BJOIN$)

Prior to requesting a data file, the user must establish a pool of buffers for handling the data blocks being generated or being manipulated by the data handling routines. The buffer pool may be assigned to one particular file or to many files. In either case, it provides temporary storage for data and occupies a portion of the object program's storage area. Although the location and control of the buffer pool are maintained by the block buffering package, the size and number of buffers contained in the pool and the number of files assigned to the pool are specified by the user. For maximum efficiency, the number of buffers for each file should be one more than the look-ahead-factor (LAF) specified in the FCT (see 13.5.1) for the file. The block buffering package assumes that the number of buffers provided equals the LAF parameter plus 1.

The size of each buffer in the pool is equal to the maximum block size specified for the files plus an additional three words for control purposes.

If the LAF entry cannot be satisfied due to insufficient buffers in the user buffer pool, the user-specified file-error-exit is taken with an error code of 11_8 and returned to the user. If the link control word is not specified or the buffer size and the control word are less than the maximum block size of the file, the file error exit is also taken by the executive. (See 13.8 for error processing.)

The file supervisor supplies the routines the user needs to set up and, when needed, expand the buffer pool. A buffer pool is a portion of the user's main storage to be used as an I/O area for one or more files. A pool can contain any number of buffers.

To establish a buffer pool, execute the following linkage:

```
L,U     A0,pktaddr
LMJ     X11,BPOOL$
```

Register A0 is loaded with the address of a two-word packet having the following format:

		H1	H2
Word	0	buffer-size	addr-of-current-first-buffer-in-pool
	1		main-storage-length

where:

buffer-size — Used to specify size (in words) of each buffer in the pool. This size must include the area needed for data plus two words for control information (first two words of buffer).

addr-of-current-first-buffer-in-pool — Starting address of the pool and, therefore, the first buffer in the pool.

main-storage-length — Total length (in words) of the main storage assigned to the buffer pool.

The number of buffers set up in the pool can be determined by dividing buffer size into main storage length. The buffers set up by the BPOOL$ request have the following format:

		H1	H2
Word	0		link-to-next-buffer-in- pool
	1	length-of-buffer -data- area	data-area-addr
	2	data-area-of-buffer	
	n		

Word 0

Starting address of the next buffer in the pool or zero if this is the last buffer in the pool.

Word 1

An I/O access control word with the buffer size (minus two words) in H1 and the address of word 2 in H2.

To expand a previously established buffer pool, execute the following linkage:

```
    L,U     A0,pktaddr
    LMJ     X11,BJOIN$
```

The two-word packet addressed by the contents of A0 contains the address of the initial control packet (BPOOL$ packet) in word 0, and the length and starting address of the main storage area to be added to the buffer in word 1. The format of the BJOIN$ packet is:

		H2	H2
Word	0		addr-of-BPOOL$-control-packet
	1	length-of-additional-main-storage-area	addr-of-additional-main-storage-area

If the block buffering package is not being used, the user must handle the buffer control scheme. The procedure as follows: To remove a buffer from a buffer pool, replace H2 of word 0 of the BPOOL$ control packet with H2 of word 0 of the buffer removed from the pool. This procedure must be preceded by testing for zeros in H2 of word 0 of the buffer being removed. If the result of the test for zero is true, the buffer just requested is the last buffer, therefore, exhausting the pool.

To return a buffer to the pool, replace H2 of word 0 of the BPOOL$ control packet with the address of the buffer being returned, and replace H2 of word 0 of the buffer being added with H2 of word 0 of the BPOOL$ control packet.

Buffers are removed from and returned to the beginning of the buffer chain.

If more than one activity is utilizing a single buffer pool, a test and set instruction must be used while the words are being updated to avoid timing problems.

Buffer pools are set up at assembly time by using the procedure call B$GPUL. The B$GPUL procedure generates a single initial control word at the procedure line, followed immediately by the pool. The format of the B$GPUL procedure call is:

```
    B$GPUL   nbr-of-buffers,buffer-size
```

13.5.3. DEFINING PHYSICAL ORGANIZATION OF DATA FILES

When handling data at the item and block levels, the user must describe the physical organization of a file to the item handler. This is accomplished by means of a procedure element set which the user calls upon from within his program. Through the use of these procedures, the user defines the type and contents of each data block that the format of a file contains. For example, block types are defined as label, data, end-of-record, and EOF; the contents of each type are defined as constants, flags, or an executive recognizable subroutine name which performs a specific action. All definitions given are controlled and manipulated automatically by the item handler and usually are not available to the user. Format definitions therefore are file independent. They are coded to describe a block, end-of-reel, or format for a family of files rather than a particular file.

Formats are defined and processed within programs or are independent of programs. An independently processed format is stored as an element on mass storage and is recalled by an object program. Since more than one format is required for a given program, it is advisable to independently define and store all formats used at a given installation. Individual users, therefore, need only call on the format desired, thus enabling the format to be used by several files. This does not preclude a user from defining his own format when necessary. Although format definitions are program independent, they occupy a portion of the object program's area since they interact with the item handler subroutines.

13.5.3.1. FORMAT DEFINITION PROCEDURES

Presently, five types of procedures are used to define a file format. They are:

- □ Format
- □ Record-type
- □ Section-name
- □ Subroutine-name
- □ End-format name

13.5.3.1.1. FORMAT PROCEDURE

The format procedure identifies a list of definitions to the item handler as being the format for a file. It also establishes the name by which the definition is referenced.

The format procedure is coded as:

```
label   FORMAT
```

where:

label Identifies to the item handler the user-specified name by which a program references this format definition.

FORMAT Specifies the name of the procedure call and is coded as shown.

The format procedure call can only appear once in a list of definitions for a format and it must appear as the first entry in that list (see Figure 13–5). The name expressed by the label parameter is placed in the format entry name field of the file's FCT (see 13.5.1).

13.5.3.1.2. RECORD-TYPE PROCEDURE

This procedure defines the type and logical beginning of each record-type contained in a format definition.

LABEL Identifies the name and logical beginning of a label record.

ITEM Identifies the name and logical beginning of an item record.

BLOCK Identifies the name and logical beginning on a block record.

EOR Identifies the name and logical beginning of an end-of-reel record.

EOF Identifies the name and logical beginning of an end-of-file record.

Only one of each record-type may be specified in any one format definition and it must be coded as shown (see Figure 13–5). Once a record-type is specified, all definitions which follow in the listing, up to the next record-type specified, are included as part of this record.

The record names are specified in any order.

13.5.3.1.3. SECTION-NAME PROCEDURE

This procedure is called upon to subdivide the record-type procedure being defined into one of three logical areas, or to write an EOF mark on a tape at a position other than the end of a reel or a file.

The section-name procedure is coded as:

PREFIX [length] Identifies the logical area immediately preceding a data item, dat block, or user's label into which the item handler controls the placement or extraction of information. The definitions that follow this section name describe the area to the item handler. The length parameter is used to specify the length of the prefix. If omitted, the prefix length is computed by the item handler.

TEXT Identifies the definitions which describe the actual data item or block.

SUFFIX [length] Identifies the logical area immediately following a data item, block, or user's label into which the item handler controls the placement or extraction of information. The definitions that follow this section-name describe the area to the item handler. The length parameter is used to specify the length of the suffix. If omitted, the prefix length is computed by the item handler.

EOFMRK Specifies that an EOF mark is to be written on tape at a position other than the end of reel or end of file. The use of this call is limited to the following file locations:

- one EOF mark following a label block
- one EOF mark preceding an end-of-file or end-of-reel format block

If specified within an end-of-record or EOF record type (see 13.5.3.1.2), the EOF is coded as the first section-name. When specified within a label record, the EOFMRK call is coded as the last section-name.

At least one section-name procedure must be given for each record-type procedure. The section-name procedures may appear, with the exception of the EOFMRK section-name, in any order within a record-type. The EOFMRK only appears as previously described (see Figure 13–5).

13.5.3.1.4. SUBROUTINE-NAME PROCEDURE

This procedure defines the disposition of the individual parameters in a section and directs the collector in the collection of those subroutines required to process the format. As many subroutine-name procedures as are necessary to completely define a section are used (see Figure 13–5).

The call for the subroutine-name procedure is coded as:

```
SENT   location   list

   or

SUBR   location   list
```

The first format is used for calling the subroutine which is used to specify the sentinel in each label and end-of-reel or EOF block in the file being processed.

For input file mode processing, control is given to the specified SENT subroutine to check the type of block processed. If a comparison is made, control is immediately returned to the item handler for further processing. If a sentinel is found, control is given to other record subroutines which perform a further examination of the parameters within a record.

The second format is used for calling all the other subroutines required for processing the parameters within a record or block.

The parameters are interpreted as follows:

SENT Identifies the procedure call as that for a sentinel subroutine.

SUBR Identifies the procedure call as that for collecting the subroutines required for processing the parameters of record or block.

location Specifies the location of the subroutine to which the control is given for record or block processing. The subroutine processes each word (one at a time) and disposes the results in the parameter designated by the list parameter in the procedure call.

list Specifies the parameters to be written or checked, depending upon the mode of file processing, by the requested subroutine.

The parameters specified in the list field are specified in either of the following two formats:

wd-nbr,j-designator,value

or

wd-nbr,CHAR (char-size,char-pos,nbr-char),value

The first format is used when the referenced subroutine is processing at the block level. The second format is used when the referenced subroutine is processing at the character level.

The parameters of the list format are interpreted as follows:

wd-nbr Specifies the word number, relative to the start of the appropriate section-name specified (PREFIX, TEXT, SUFFIX), that is to be processed by the referenced subroutine. The calls need not be in sequence, and only those words or portions of words necessary to define the format are used. An exception to this is when the prefix or suffix length parameter is not specified on the section-name procedure call. In this case, the last word in the section must be called. The first word in each section is word number 0.

j-designator Numeric value from 0_8 to 15_8 specifying the portion of the word to be processed.

value Specifies the value to be placed in the portion (j-designator) of the word specified by the wd-nbr parameter for output file mode processing, or checked for input file mode processing.

CHAR Specifies that the word being processed is character oriented.

char-size Specifies the number of bits in a character.

char-pos Specifies the position of the leftmost character to be processed. Character numbering starts with the leftmost character of the word number as character 1.

nbr-char Specifies the number of characters (as defined by char-size) to be processed by the subroutine.

The following examples illustrate the coding and interpretation of the SENT and SUBR procedure calls:

```
      LABEL        OPERATION                  OPERAND                         COMMENTS
   1          10          20          30          40          50
1. SUBR   BLKCT     3,0
2. SENT   SENTNL    0,1,'/./.'     1,2,06
3. SUBR   CNT       2,CHAR(4,3,2),0
```

1. The subroutine named BLKCT generates and checks a block count in the complete fourth word of a section.

2. Calls for the sentinel routine SENTNL to place the sentinel /./. in H2 of the first word and the sentinel 06 into H1 of the second word of a record. These two words are used to find a specific record input.

3. The subroutine CNT is to generate and check the third word of a record. The word is a four-bit character-oriented word, and the count is maintained in two characters starting with the third character.

13.5.3.1.5. END-FORMAT PROCEDURE

This procedure defines the end of a complete format definition and appears as the last procedure in the definition (see Figure 13–5).

The end-format procedure is coded as:

FOREND   Specifies to the item handler that no more definitions are given for this format.

13.5.3.2. EXAMPLE OF A COMPLETED FORMAT DEFINITION

The complete format definition shown in Figure 13–5 describes the LION format to the item handler data handling routines. The coding for the format, record-type, section-name, subroutine-name, and end-format procedures are illustrated and called out.

13.5.3.3. RULES FOR CODING FORMAT SUBROUTINES

When a specific format must be generated or checked on a file, control is given to the appropriate subroutine by the item handler. To determine whether to generate or check a format, control is given with register A0 positive for output files and negative for input files.

The following registers are set as indicated when the subroutine receives control and must not be destroyed by the ꞩubroutine.

A1 – H1 – Return to item from format
H2 – Return to format from subroutine

A2 – H2 – Item request packet location

A3 – H1 – Return to user from item
H2 – File control table location

```
Format name         LION   FORMAT
Return type                LABEL
Section name               PREFIX
                          ⎧SENT        LLABST      0,0,'/././.'     1,13,060
                          ⎪SUBR        LBLKCT      1,0
Subroutine name           ⎨SUBR        LBLKLT      2,2
                          ⎪SUBR        LITEMS      2,1
                          ⎩SUBR        LFWDS       3,2
                           SUB         LFREE       3,1
Section name               SUFFIX
                          ⎧SUBR        LLSRWD      0,1
                          ⎪SUBR        LVERSN      0,2,4
                          ⎪SUBR        LITMCT      1,0
Subroutine name           ⎨SUBR        LCKSUM      2,0
                          ⎪SUBR        LBLKLT      3,2
                          ⎪SUBR        LITEMS      3,1
                          ⎪SUBR        LBLKCT      4,0
                          ⎩SUBR        LLABST      4,13,060         5,0,'/././.'
```

Figure 13-5. Example of Complete Format Definition (Part 1 of 2)

Record type	BLOCK			
Section name	PREFIX			
Subroutine name	SUBR	LNRITM	0,2	
	SUBR	LNRWDS	0,1	
Section name	SUFFIX			
	SUBR	LDSRWD	0,1	
	SUBR	LBCKSM	1,0	
Subroutine name	SUBR	LNRITM	2,2	
	SUBR	LNRWDS	2,1	
Record type	EOR			
Section name	PREFIX			
	SENT	LEORST	0,0,‘/././.’	1,13,020
	SUBR	LERBCT	1,0	
	SUBR	LERBLT	2,2	
Subroutine name	SUBR	LERITS	2,1	
	SUBR	LZEROW	3,0,0	4,0,0
	SUBR	LVERSN	5,2,4	
	SUBR	LZEROW	6,0,0	
	SUBR	LSUFLC	7,0,0	
Section name	SUFFIX			
	SUBR	LVERSN	0,2,4	
	SUBR	LERICT	1,0	
	SUBR	LERCKM	2,0	
Subroutine name	SUBR	LERBLT	3,2	
	SUBR	LERITS	3,1	
	SUBR	LERBCT	4,0	
	SUBR	LEORST	4,13,020	5,0,‘/././.’
Record type	EOF			
Section name	PREFIX			
	SENT	LEOFST	0,0,‘/././.’	1,13,0
	SUBR	LEFBCT	1,0	
	SUBR	LEFBLT	2,2	
Subroutine name	SUBR	LEFITS	2,1	
	SUBR	LZEROW	3,0,0	4,0,0
	SUBR	LVERSN	5,2,4	
	SUBR	LZEROW	6,0,0	
	SUBR	LSUFLC	7,0,0	
Section name	SUFFIX			
	SUBR	LVERSN	0,2,4	
	SUBR	LEFICT	1,0	
	SUBR	LEFCKM	2,0	
Subroutine name	SUBR	LEFBLT	3,2	
	SUBR	LEFITS	3,1	
	SUBR	LEFBCT	4,0	
	SUBR	LEOFST	4,13,0	5,0,‘/././.’
End format	FOREND			

Figure 13–5. Example of Complete Format Definition (Part 2 of 2)

Register A0 is set as follows on entry to the subroutine. The contents in S1 must not be destroyed; the remaining portion of the register may be destroyed.

A0 S1 – Positive for output files

Negative for input files

H2 – Location of packet containing parameters from subroutine call line. The packet format appears as follows:

	S1	S2	S3	S4	S5	S6
Word 0	value					
1	wd-nbr	j-designator	code	char-size	char-pos	nbr-char

The parameters are as explained in 13.5.3.1.4 and the values for the code parameter are:

code = 0_8 for prefix section

1_8 for text section

2_8 for suffix section

H2 = 0_8 if the first subroutine form is used

Registers X11, A4, A5, R1, R2, and R3 are used freely. All others are saved and restored before exiting from the subroutine. Because the item handler is reentrant, all subroutines are treated as such; therefore, they are coded with this in mind.

The exits from the subroutines for normal conditions are taken on 0, word 0 of register A1. Exits for abnormal conditions are taken to IHABFR with one of the following codes set in register A0, S6:

1_8 — Exit from label sentinel routine with no find

2_8 — Exit from label routine with abnormal label

3_8 — Exit from block routine with abnormal block

4_8 — Exit from item routine with abnormal item

5_8 — Exit from EOR sentinel routine with no find

6_8 — Exit from EOR routine with abnormal end-of-record block

7_8 — Exit from EOF sentinel routine with no find

10_8— Exit from routine with abnormal EOF block

Registers A0, S4, and S5 are used to transfer any data pertinent to the user. After interpreting the code in S6, the item handler takes the user's abnormal exit with registers A0 and A1 set as follows:

A0 Length and location of current block or item

A1 H1 – Code pertinent to user

H2 – Address of request that found error

13.5.3.3.1. LABEL RECORD SUBROUTINES

Only three of the four section-names may be used with the label record. These are PREFIX, SUFFIX, and EOFMRK. A list may be specified on the PREFIX and SUFFIX calls to designate the length of the respective section. If not set, the length is computed from the parameters on the subroutine calls. The total length of these sections is the total length of the label block. It must be less than or equal to the maximum data block size allowed for the file.

If the label or FREEWDS (LION format) is specified by the user in the FCT (see 13.5.1), a move routine must be included as a subroutine to move this area into the label block. The location of the start of the prefix within the buffer is specified in word 28,H2 of the FCT. The location of the start of the suffix is specified in word 26,H2 of the FCT.

If EOFMRK is specified as a label section-name, an EOF is written for output files or expected on input files immediately following a label block. No subroutine calls may follow an EOFMRK.

Label output subroutines place data in the label prefix and suffix as specified by the subroutine lists. A subroutine called by the procedure call SENT, must be included as one of the subroutine calls in a label record. Return to format from each subroutine must be made on 0, to register A1 if the label is generated properly and to IHABFR with S6 of register A0 set to 2 if not done properly.

Label input subroutines must validate the label block. The sentinel routine first checks the sentinel. If it is found, the routine must return on 0,A1. If it is not found, return must be made to IHABFR with register A0,S6 set to 1. Since this routine is called on every READ request, make certain it is very concise. All other label input subroutines must check other parts of the label. If the label is accepted as valid, return must be made on 0,A1. If the subroutine wishes to discontinue processing, it must return to IHABFR with S6 of register A0 set to 2.

13.5.3.3.2. BLOCK RECORD SUBROUTINES

Within a block record, PREFIX, TEXT and SUFFIX section-names are used. The length of the prefix and suffix can be computed from the subroutine call. Format subroutines specified as a list on the appropriate section call, or the length for output files, receive control for generating block formats after all items have been moved into the block. Input subroutines receive control before any items have been moved from the block. The location of the first word of the prefix is contained in word 28,H2 of the FCT (see 13.5.1), the text in word 28,H1 of the FCT, and the suffix in word 26,H2 of the FCT.

Return from each subroutine is made on 0,A1 for normal conditions. When further processing of the file is no longer desired or the block was generated incorrectly, the subroutine exits to IHABFR with S6 of register A0 set to 3.

13.5.3.3.3. ITEM RECORD SUBROUTINES

Within an item record, PREFIX, TEXT, and SUFFIX section-names are used. The length of the prefix and suffix is specified as a list on the appropriate section call, or the length can be computed by the subroutines. The location of the beginning of the prefix is specified in word 28,H2 of the FCT (see 13.5.1), the text in word 26,H2 of the FCT, and the suffix in word 26,H2 of the FCT. Control is given at different times.

For output files, control is given to the PREFIX subroutines before the item is moved, unless the user has already moved it himself. Control is then given to the TEXT and SUFFIX subroutines after the item is moved.

For input items processed in the forward mode, control is given to the PREFIX and TEXT subroutines before the item is moved, and to the SUFFIX subroutines after it is moved.

For input files processed in the reverse mode, control is given to the SUFFIX and TEXT subroutines before the item is moved, and to the PREFIX subroutines after it is moved.

If item size is variable, a subroutine is included under both the prefix and suffix. Therefore, if a variable-size item has a prefix, it must have a suffix and vice versa. This subroutine must, on output files, insert the length of the item and formats within the PREFIX and SUFFIX. This length is specified in word 29,H2 of the FCT. On input files, the subroutine retrieves this length and places it in word 29,H2 of the FCT. If neither PREFIX nor SUFFIX is specified, a one-word control word is placed before and after each item. If items are fixed, the lengths need not be saved. If the file is not processed in the reverse mode, the suffix is not needed.

Return from each subroutine is made on 0,A1 except when further processing of the item is no longer desired. Then the subroutine exits to IHABFR with S6 of register A0 set to 4.

13.5.3.3.4. END-OF-REEL SUBROUTINES

End-of-reel processing is similar to processing of label blocks. Three section names, PREFIX, SUFFIX, and EOFMRK, are used. EOFMRK indicates that an EOF mark is to be written on tape before the end-of-reel block. The prefix and suffix lengths are determined from the list on the call line or from the parameters in the subroutine packet. The total length is equal to or less than the maximum block length specified in the FCT (see 13.5.1). The location of the first word of the prefix is contained in word 28,H2 of the FCT, and the suffix in word 26,H2 of the FCT.

For output files, an end-of-reel block is written when an end of tape is encountered. One SENT subroutine is included and the exit is made on 0,A1.

For input files, the SENT subroutine is given control to check for an end-of-reel sentinel. If not found, an exit should be made to IHABFR with register A0 set to 5. If it is found, the exit is made on 0,A1. Control is then given to other end-of-reel subroutines for further checking. If an abnormal end-of-reel block is found and processing is discontinued, an exit is made to IHABFR with S6 of register A0 set to 6. The normal return is made on 0,A1.

13.5.3.3.5. END-OF-FILE SUBROUTINES

EOF processing is similar to the processing of label blocks. Three section names, PREFIX, SUFFIX, and EOFMRK, are used. EOFMRK indicates that an EOF mark is to be written on tape before the EOF block. The prefix and suffix lengths are determined from the list on the call line or from the parameters in the subroutine packet. The total length is equal to or less than the maximum block length specified in the FCT. The location of the first word of the prefix is contained in word 28,H2 of the FCT (see 13.5.1) and the suffix in word 26,H2 of the FCT.

For output files, an EOF block is written when the file is closed. One SENT subroutine is included and the exit is made on 0,A1.

For input file, the SENT subroutine is given control to check for an end-of-file sentinel. If it is not found, an exit is made to IHABFR with register A0 set to 7. If it is found, exit is made on 0,A1. Control then goes to other end-of-file subroutines for further checking. If an abnormal end-of-file block is found and processing is discontinued, an exit is made to IHABFR with control register A0, S6 set to 10_8)

13.6. HANDLING OF LABELS AND SENTINELS

The item handler utilizes a set of procedures to govern the handling of label and sentinel blocks for the input and output files processed. These procedures are directly related to the format definitions given for the file (see 13.5.3) and the type of device on which the file resides.

13.6.1. LABEL AND SENTINEL HANDLING FOR OUTPUT FILES

The item handler generates and records those label and sentinel blocks for which a format definition has been specified. If a format definition has not been specified, the corresponding block is not produced. The manner in which label and sentinel blocks are written for FASTRAND-formatted mass storage and tape output files is described in the following paragraphs.

- FASTRAND-formatted Mass Storage Files

  If specified, labels and EOF sentinels for FASTRAND-formatted mass storage files are recorded as the first and last blocks of the file, respectively. See Figure 13–10 for file organization.

- Magnetic Tape Files

  Labels are written as the first block of each file recorded on a tape reel and at the beginning of each reel on a multireel file. An end-of-reel sentinel, in the case of a multireel file, is recorded at the end of each reel. The EOF sentinel is always recorded as the last block of a file in both a single or multireel file.

  If a label and an EOF definition are specified in a format, the processing of the file produces either a single reel file with a label and an EOF sentinel or a multireel file without end-of-reel sentinel or a multireel file without end-of-reel sentinels but with a label and an EOF sentinel. If label, end-of-reel, or EOF definitions are not specified, the single or multireel tape file will contain only data.

  If the format definition specifies the generation of EOF marks, then one mark is recorded before and two marks are recorded after each end-of-reel or EOF sentinel. In the absence of either sentinel, two EOF marks are written immediately after the last data block in the file. Files recorded on a multifile reel are separated by a single EOF mark.

  See Figures 13–6 through 13–9 for tape file organization. The switching of tape files is accomplished by the item handler upon request from the CLOSE REEL subroutine.

13.6.2. LABEL AND SENTINEL HANDLING FOR INPUT FILES

The existence of label and sentinel format definitions implies that label and sentinel blocks exist in the input files processed by the item handler. When these blocks are encountered, they are validated according to their respective definitions prior to processing. The absence of a definition implies that the corresponding label or sentinel block does not exist in the file. If this is not the case, an error condition could result. Any error condition detected causes the abnormal exit specified in the FCT to be taken. User label information, if any, is presented to the object program for checking. Register A0 is loaded with the location of the user's label area in the form of the following access word:

H1	H2
nbr-wds	addr

If there is no user label information, register A0 is zero filled.

When format definitions for labels have not been specified for a file, the item handler assumes the file to be unlabeled and processes the first block as data.

Files opened in the reverse mode are processed as unlabeled files.

When the input file exists on magnetic tape and an EOF mark is detected, the sentinel block is read before the appropriate exit is taken providing that format definitions have been specified for sentinels. If, however, a sentinel definition was not specified the EOF exit specified in the FCT (see 13.5.1) is always taken. The EOF exit is also taken when two successive EOF marks are detected.

13.7. DATA FILE ORGANIZATION

Data files are organized according to the level at which a file is processed and the device on which the file resides. Files handled at the item level are processed by the subroutines of the item handler and may reside on magnetic tape, or FASTRAND-formatted mass storage. When item level files reside on magnetic tape, they are orginized as shown in Figures 13–6 through 13–9. The examples shown depict item level files as they are organized on single-file reels, multifile reels, and multireel files. The EOF marks shown in the examples are recorded automatically by the item handler. The placement of these marks is based upon the following:

(1) Two EOF marks are recorded after an end-of-reel or EOF sentinel block.

(2) If an end-of-reel or EOF sentinel is not written, two EOF marks follow the last valid data block of the file.

(3) When a reel contains more than one file, the second of the two EOF marks is overwritten, thereby separating files with one mark.

(4) A special separator block is recorded by the item handler when a file is opened and closed on tape without recording any data and for which EOF sentinels have not been specified. This block denotes an empty (void) file which serves to separate the EOF marks.

The organization of item level files residing on FASTRAND-formatted mass storage is shown in Figure 13–10.

The files handled at the block level are processed by the block buffering package. Block level files also reside on magnetic tape or FASTRAND-formatted mass storage. However, these files are assumed to contain pure data. EOF marks, therefore, are not recorded automatically but are placed at the user's descretion. It is suggested that EOF marks be used to eliminate the input of invalid data when performing a read operation.

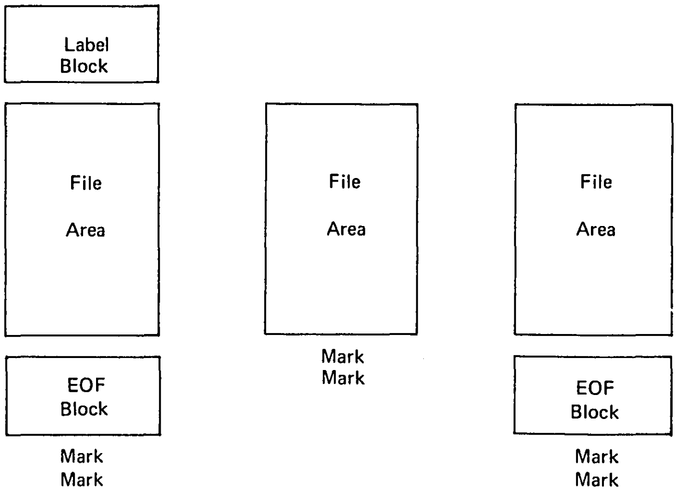


Figure 13-6. Item Level Tape File Organization, Single-File Reels

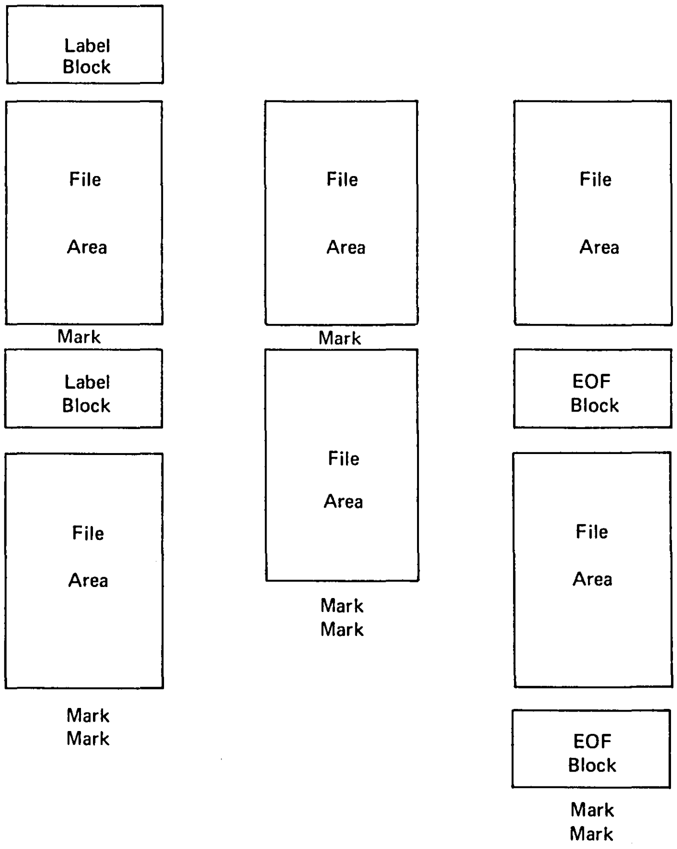


Figure 13-7. Item Level Tape File Organization, Multifile Reels

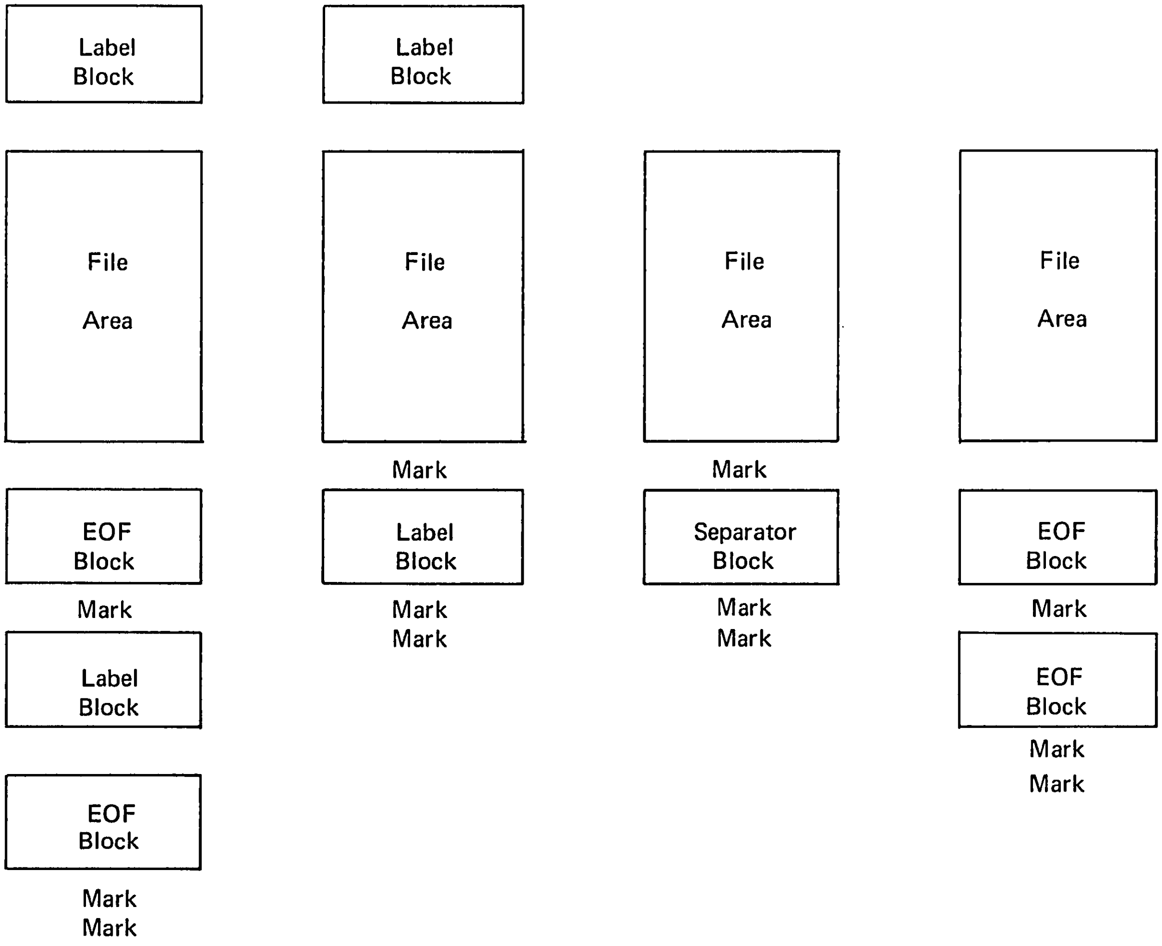
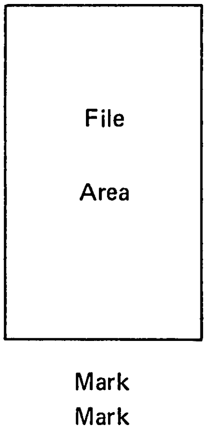


Figure 13-8. Item Level Tape File Organization, Multifile Reels with Void File



NOTE: Organization of first reel is illustrated; subsequent reel layout is the same.

Figure 13-9. Item Level Tape File Organization, Multireel Files

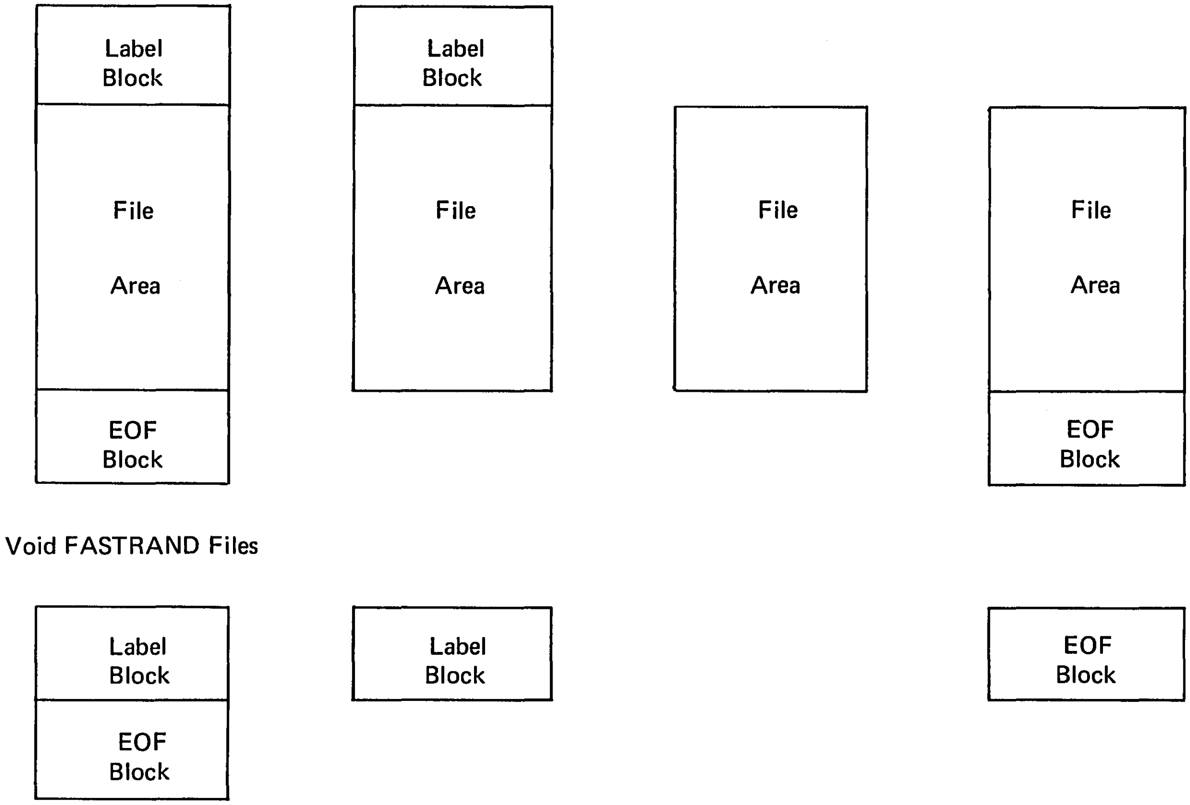


Figure 13-10. Item Level FASTRAND-Formatted Mass Storage File Organization

13.8. ERROR PROCESSING

The executive returns control to the user program when a device, file, or abnormal error condition is detected. The return of control is accomplished through the appropriate error exit control word specified in the FCT for the file being processed. A description is provided for each catagory of error conditions and the respective codes that it returns to the user program.

13.8.1. DEVICE ERROR HANDLING

The occurrence of a device error causes control to be returned to the user program by means of the device exit control word specified in the FCT (see 13.5.1). When control is returned, the status code for the particular device error is in H1 of register A0 and the user's reentry location is in H2 of register A0. If the file is processed in the input mode, register A1 contains the size and data location of the data block that was being read when the error occurred. See Table 13–1 for descriptions of the status codes for device error.

13.8.2. FILE ERROR HANDLING

File errors are the result of a user's request that has violated prescribed block buffering package procedures. For each occurrence of a file error, control is returned to the user program by means of the user's file error exit specified in the FCT (see 13.5.1) of the file being processed. An error code and the user reentry location are returned to the user in H1 and H2, respectively, of register A0. Possible error codes and their descriptions are listed in Table 13–1.

Error Code (Octal)	Description
1	Buffer pool link not provided.
2	Request to close a file already closed.
3	Request to open a file already opened.
4	Request to read or write a closed file.
5	Request to write a block greater than maximum block size specified in FCT or a request to rewrite a block in in/out mode and block size requested to write is greater than the block size read.
6	FASTRAND variable block size specified on block read request larger than maximum block size, or reading variable blocks from fixed block file.
7	Random request and file not assigned to FASTRAND-formatted mass storage.
10	Random request and block size not fixed.
11	Insufficient buffers in pool to satisfy LAF for input or output request.
12	Invalid block number for random read request.
13	Read request for a block greater than block size read.
14	File not assigned to FASTRAND-formatted mass storage for in/out mode processing.
15	Random write request for input file.
16	Random read request for output file.
17	Read request with move-length parameter specified but no move-address specified.
20	Read request for output file.
21	Buffer size less than specified block size.
23	Location of link or buffer area outside of user's assigned area.
24	Block size not fixed for reverse mode for FASTRAND-formatted mass storage file.
25	No I/O facilities assigned or improper equipment type.
26	Write request in input file.
27	Mark request for input file.
64	Invalid mode parameter for open request.

Table 13-1. Device Error Status Codes (Part 1 of 2)

Error Code (Octal)	Description
The following error codes occur only when processing at the item level:	
30	Move address not given on in/out read or write request.
31	Move address not given on read or write request of spanned item.
32	Address of item same as move address given on in/out read or write request.
33	Address of item same as move address given on in/out read or write request.
34	Address of item same as move address given on random read or write request.
35	Random request made in file without fixed items and blocks.
36	Operation attempted on a closed file.
37	Label specified but not found on open input request.
40	Update flag not set on in/out file read request.
41	Highest address written not set on in/out file write extend request.
42	Drain requested on in/out file.
43	A request was made to write a spanned item after reading it.
44	A request was made to write more than the size read on in/out file.
45	Read backwards requested on a spanned item.
46	Format not set in FCT.
47	Tape positioned improperly on open request.
50	Frame count error detected and user did not specify to accept this as normal.
51	An EOF block was detected on a random read request.
52	Fixed item larger than max block size.

Table 13-1. Device Error Status Codes (Part 2 of 2)

13.8.3. ABNORMAL ERROR HANDLING

The occurrence of an abnormal condition causes control to be returned to the user's program by means of the abnormal exit control word in the FCT (see 13.5.1). When control is returned, the abnormal status code for the particular abnormal error is in H1 of register A1 and the user's reentry location is in H2 of this register. Register A0 contains the size and data location of the block read or written when the abnormal condition occurred. The abnormal status codes returned to the user are:

Abnormal Status Codes (Octal)	Description
1	EOF mark or load point has been detected for input tape file; or FASTRAND highest address (EOF) has been detected.
2	End-of-tape mark has been detected for output file.
6	Sentinel block has been detected for file.
10	Block previously read exclusively has been timed out by the executive for an in/out file.

The abnormal exit in the FCT is also used to return control to the user when an error is detected in a sentinel block when using the LION format.

The abnormal status codes and their description are:

Abnormal Status Codes (Octal)	Description
4	Label word error has been detected in label block for an input file.
12	Label block cannot be located, bad tape position.
13	Block size specified in FCT is less than 14 words for an output file; or input file block size of sentinel is greater than block size specified in FCT.
14	Item size specified in FCT not equal to item size in sentinel block for an input file.

Recovery can be made for error codes 4_8, 13_8 and 14_8 by closing the file, modifying the respective fields in the FCT, and initiating the open procedure again. For error code 12_8, the tape must be repositioned.

14. OUTPUT EDITING PACKAGES

14.1. INTRODUCTION

This section describes the two output editing packages: EDIT$ and EOUT$. EDIT$ is the newer of the two and it is more efficient and easier to use than EOUT$. It is recommended that EDIT$ be used; EOUT$ is documented solely as an aid to those users who are using EOUT$ in existing programs.

14.2. EDIT$ (IMAGE COMPOSITION EDITING PACKAGE)

EDIT$ is a set of reentrant subroutines used for composing strings of Fieldata characters in a user-specified area. It is particularly useful in preparing images for:

- ER CSF$ (see 4.8.1)
- ER PRINT$ (see 5.3.1)
- ER PUNCH$ (see 5.3.5)
- ER PRTCN$ (see 5.4.1)
- ER PCHCN$ (see 5.4.5)

The EDIT$ routines work from the following packet where words six through nine are used only for editing floating-point numbers:

	S1	S2	S3	S4	S5	S6
Word 0	[test-and-set]	EMSG$-stop	image-length	image-addr		
1	*char-index*	*word-index*	*EMSG$-char*	*EMSG$-word (index)*		
2	FPS	FPR	0	*return-addr-for-char-store*		
3	*user's-return-addr*			*save-of-original-X1-modifier*		
4	*save-of-original-X2-contents-or-save-of-character-pointer*					
5	*save-of-original-X3-contents-or-save-of-word-pointer*					
6	[DPC]	[SPC]	*digits-before*	*digits-after*	*negative-sign*	*not-normalized*
7	*final-column-position*			*characteristic's-power-of-ten*		
8	*save-area-for-intermediate-floating-point-results*					
9						

Word 0

test-and-set

User test and set flag.

EMSG$-stop

Signal character for EMSG$ and EMSGR$. Each time one or more signal characters are encountered in an EMSG$ or EMSGR$ input image, the routine stops transferring characters. This can serve not only as the final EMSG$ input image stop but also allows the user to insert new information at any predetermined point in the EMSG$ image. The ampersand (&) is often used as the EMSG$ signal character.

image-length

The size of the output image buffer. When using the 132-column printer, 22 words are usually specified.

image-addr

Output image buffer address. When the image is fully formed, this buffer can be referenced directly on a call to ER CSF$, ER PRINT$, and so forth.

Word 1

char-index
Used by the EDITX$ routine to save the character index for the EDITR$ routine.

word-index
Used by the EDITX$ routine to save the relative word index for the EDITR$ routine.

EMSG$-char
Used by EMSG$ and EMSER$ routines to save the character index of the input image.

EMSG$-word (index)
Used by the EMSG$ and EMSGR$ routines to save the input message word index.

Word 2

FPS
Used only with floating point. The scale or number of digits to be placed before the decimal point for scientific floating-point format editing. This is usually set to 1.

FPR
Used only with floating point. Specifies floating-point rounding. When set to nonzero, five is added to the eighth significant digit for single-precision floating-point numbers or to the eighteenth significant bit for double-precision numbers.

return-addr-for-char-store
Used as the return point for the character store vector during calls to other EDIT$ routines.

Word 3

user's-return-addr
Used to save the return point for the user during calls to EDIT$ functions.

Word 6

DPC
Used for floating point only. If nonzero, specifies the character that separates the mantissa and characteristic when editing double-precision floating-point numbers.

SPC
Same as DPC except that it is used for single-precision floating-point numbers.

digits-before
Number of digits to be edited before the decimal point.

digits-after
Number of digits to be edited following the decimal point.

negative-sign
If nonzero, specifies that the floating-point number to be edited is negative.

not-normalized
If nonzero, specifies that the floating-point number to be edited was not normalized.

14.2.1. GENERATING THE EDIT$ PACKET (E$PKT AND E$PKTF)

The following procedure generates a six-word EDIT$ packet: the ampersand (&) is generated for the EMSG$-stop-character if the MSG parameter is omitted.

```
E$PKT   image-length,image-addr   ['MSG','EMSG$-stop']
```

The following procedure call generates a ten-word EDIT$ packet. The following values are assumed whenever the following parameters are omitted:

MSG = &

FPS = 1

FPR = 1

DPC = 0

SPC = 0

```
E$PKTF    image-length,image-addr  ['MSG','EMSG$-stop]  ['FPS',FPS-nbr]  ['FPR',FPR-nbr]
['DPC','DPC-char']  ['SPC','SPC-char']
```

14.2.2. INITIALIZATION AND TERMINATION OF EDITING MODE

Table 14-1 presents the subroutine calling sequences used to initiate and terminate the editing mode.

Routine	Calling Sequence	Description
EDIT$	L,U A0,pktaddr LMJ X11,EDIT$ These instructions can be generated by the procedure call: E$DIT pkraddr	Initiate editing mode. The contents of registers X1, X2, and X3 are saved. The image is space filled and the column pointer is set to the start of the image. EDIT$ uses, but does not save or restore registers X11, R1, and A0 through A3.
EDITX$	LMJ X11,EDITX$ Instruction can be generated by the procedure call: E$DITX	Terminate the editing mode. The column pointer is saved in the packet. Registers X1, X2, and X3 are restored to their original contents. The address of the packet is returned to A0.
EDITR$	L,U A0,pktaddr LMJ X11,EDITR$ These instructions can be generated by the procedure call: E$DITR pktaddr	Reestablish the editing mode to its previous status at the time of the call to EDITX$. The column pointer saved by EDITX$ is restored.

Table 14–1. Editing Routines for Initiation and Termination of Editing Mode

14.2.3. GENERAL PURPOSE EDITING ROUTINES

Table 14-2 describes the nonfloating-point editing routines. These routines are used for such purposes as converting numbers into Fieldata format, inserting strings of characters into the image, and manipulation of the column pointer. Please note that in all cases where one or more characters are inserted into the image, they are inserted beginning at the current column pointer location. The column pointer location is initially set to column zero by the initiate editing mode subroutine and is always advanced to the column following the last inserted character.

Routine	Calling Sequence	Description
ECHAR$	Two calling sequences are provided: L,U A0,'char' LMJ X11,ECHAR$ These instructions can be generated by the procedure call: E$CHAR 'char' or L A0,addr-of-char LMJ X11,ECHAR$ These instructions can be generated by the procedure call: E$CHAR addr-of-char Where addr-of-char may contain any of the x,h,i,u,j instruction word fields in the standard *u,*x,j form.	Insert the character in S6 of register A0 into the image.
ECOL$	L,U A0,col-nbr LMJ X11,ECOL$ These instructions can be generated by the procedure call: E$COL col-nbr	Position the column pointer to the column number in register A0. A previously inserted character or string can be backed-up to and overwritten.
ECOLN$	LMJ X11,ECOLN$ This instruction can be generated by the procedure call: E$COLN	Obtain the current column in register A0.
ECOPY$	L,U A1,char-count L,U A0,addr-of-char LMJ X11,ECOPY$ These instructions can be generated by the procedure call: E$COPY char-count,addr-of-char	Insert into the image the number of characters specified in register A1. The image starting address is in register A0 and all characters are transferred including spaces.

Table 4-2. General Purpose Editing Routines (Part 1 of 4)

Routine	Calling Sequence	Description
EDAY1$	L A0,TDATE$-addr LMJ X11,EDAY1$ These two instructions can be generated by the procedure call: E$DAY1 TDATE$-word-addr or for today's date: ER TDATE$ LMJ X11,EDAY1$ These instructions can be generated by the procedure call: E$DAT1	Convert the date portion of the TDATE$ word format (see 4.5.2) in register A0 to an eight-character Fieldata string in the form of mm/dd/yy and insert this string into the image.
EDAY2$	Calling sequence formats are the same as for EDAY1$. The corresponding procedure calls are: E$DAY2 E$DAT2	Convert the date portion of the TDATE$ word format (see 4.5.2) in register A0 to a nine-character Fieldata string in the form of dd mmm yy and insert this string into the image.
EDAY3$	Calling sequence formats are the same as for EDAY1$. The corresponding procedure calls are: E$DAY3 E$DAT3	Convert the date portion of the TDATE$ word format (see 4.5.2) in register A0 to a variable string length of 11 to 17 Fieldata characters in the form of: mmmmmmmmm dd yyyy and insert this string into the image.
EDECF$	L,U A1,char-count L A0,addr-of-nbr LMJ X11,EDECF$ These instructions can be generated by the procedure call: E$DECF char-count,addr-of-nbr	Convert the number in register A0 to Fieldata decimal digits and insert it into the image. The result is set right-justified and space filled into the fixed-length field specified in register A1. If the number of Fieldata digits (including the minus sign if the content of register A0 is a negative number) exceeds the specified field size, overflow to the next field.
EDECV$	L A0,nbr-addr LMJ X11,EDECV$ These instructions can be generated by the procedure call: E$DECV nbr-addr	Same as EDECF$ except that the field size is varied in length according to the converted Fieldata decimal number.

Table 4-2. General Purpose Editing Routines (Part 2 of 4)

Routine	Calling Sequence	Description
EFD1$	L A0,('Fieldata-name') LMJ X11,EFD1$ These instructions can be generated by the procedure call: E$FD1 'Fieldata-name'	Insert the contents of register A0 into the image excluding any sixth-word whose value is zero, that is, the Fieldata master space (@), or Fieldata space (' '). This command can be used to insert a Fieldata name of one word or less where a partial word fill of zeros is not desired in the image.
EFD2$	DL A0,('Fieldata-name') LMJ X11,EFD2$ These instructions can be generated by the procedure call: E$FD2 'Fieldata-name'	Same as EFD1$ except that it is for a Fieldata name of two words or less which is contained in registers A0 and A1.
EMSG$	L,U A0,input-string-addr LMJ X11,EMSG$ These instructions can be generated by the procedure call: E$MSG input-string-addr	Insert the characters starting at the address in register A0 into the image. This process stops when the character in S2 of register A0 of the EDIT$ packet is encountered in the EMSG$ input string. The EMSG$ input string pointer is saved in the packet for further possible use with the EMSGR$ call.
EMSGR$	LMJ X11,EMSGR$ This instruction can be generated by the procedure call: E$MSGR	Reenter the EMSG$ subroutine and begin copying from the EMSG$ input stream following the point of previous interruption. With the EMSG$ and EMSGR$ subroutine it is possible to copy a string into the image and interrupt this action to perform other EDIT$ functions at selected points in the string.
EOCTF$	L,U A1,char-count L A0,nbr-addr LMJ X11,EOCTF$ These instructions can be generated by the procedure call: E$OCTF char-count,nbr-addr	Convert the number in register A0 to Fieldata octal digits, and set the result right-justified and zero filled into a fixed-length field of the number of characters specified in register A1. If the number of Fieldata octal digits (including a leading 0, if the magnitude of the number is greater than 7) exceeds the specified field size, truncate one or more low order digits. Insert this field into the image. EOCTF$, unlike EDECF$, does not assume that the number in register A0 is necessarily a representation of a single positive or negative qualtity, and therefore EOCTF$ does not take special action to complement the number and add a leading minus sign if the high order sign bit is set. If, for example, register A0 had all bits set to 1 except bit 0, EOCT$ would convert this to the 13-character string 0777777777776_8, while EDECF$ would convert it to a two-character string of –1.

Table 4-2. General Purpose Editing Routines (Part 3 of 4)

Routine	Calling Sequence	Description
EOCTV$	L A0,nbr,addr LMJ X11,EOCTV$ These instructions can be generated by the procedure call: E$OCTV nbr-addr	This is the same as EOCTF$ except that the field size is variable and only as large as is necessary to hold the converted octal number including the leading zero if the magnitude of the number is greater than seven.
EPACK$	L,U A1,char-count L,U A0,char-addr LMJ A11,EPACK$ These instructions can be generated by the procedure call: E$PACK char-count,char-addr	This is the same as ECOPY$ except that sixth word whose value is the Fieldata master space (@), although included in the input character count, is not inserted into the image.
ESKIP$	L,U A0,column-count LMJ X11,ESKIP$ These instructions can be generated by the procedure call: E$SKIP column-count	Advance the column pointer by the column count given in register A0. To back up the count, register A0 may be loaded with a negative number.
ETIME$	L A0,TDATE$-word-addr LMJ X11,ETIME$ These instructions can be generated by the procedure call: E$TIME TDATE$-word-addr or to obtain the real time clock time ER TDATE$ LMJ X11,ETIME$ These instructions can be generated by by the procedure call: E$TD	Convert and insert into the image the time portion of the TDATE$ word format (see 4.5.2) to an eight-character Fieldata string in the form hh:mm:ss

Table 4-2. General Purpose Editing Routines (Part 4 of 4)

14.2.4. FLOATING-POINT EDITING ROUTINES

Table 14-3 contains the floating-point editing routines. Upon entering any of the floating-point routines, register A0 must contain a number in the form x*/6+y where x is the desired field size and y is the desired number of significant digits. When the field size is larger than necessary, the edited floating-point result is right-justified and space filled.

To avoid overflow to the right, the minimum necessary field size is y+4 plus 1 for each of the following cases:

- a separator character is used in the DPC and SPC packet fields;
- the number is negative and requires a leading minus sign; or
- the number is double precision and requires a three-digit characteristic.

Routine	Calling Sequence	Description
EFLF1$	L,U A0,x*/6+y L A1,nbr-addr LMJ X11,EFLF1$ These instructions can be generated by the procedure call: E$FLF1 x*/6+y,nbr-addr	Convert the floating-ppoint number in register A1 to a fixed-point Fieldata string consisting of: a leading minus sign if the number is negative; a period to represent the decimal point; the number of digits specified by y; a separator character if one was specified in the SPC packet field; a plus or minus sign indicating if the two-digit characteristic has a positive or negative exponent. The string is set into a field of x characters and insert the field into the image.
EFLG1	L,U A0,X*/6+y L A1,nbr-addr LMJ X11,EFLG1$ These instructions can be generated by the procedure call: E$FLG1 x*/6+y,nbr-addr	This is the same as EFLF1$ except that an attempt is made to shift the decimal point among the significant digits so that the characteristic goes to zero, in which case the characteristic is set to blanks. If the attempt fails, the decimal point is placed following the number of significant digits specified in the EDIT$ packet FPS field.
EFLS1$	L,U A0,x*/6+y L A1,nbr-addr LMJ X11,EFLS1$ These instructions can be generated by the procedure call: E$FLS1 x*/6+y,nbr-addr	This scientific floating-point format subroutine operates in the same manner as the EFLF1$ routine, except that the the decimal point is always placed following the number of significant digits specified the EDIT$ packet FPS field.
EFLF2$	L,U A0,x*/6+y LD A1,nbr-addr LMJ X11,EFLF2$ These instructions can be generated by the procedure call: E$FLF2 x*/6+y,nbr-addr	Same as EFLF1$, except the floating-point number is in registers A1 and A2.

Table 14–3. Floating-Point Editing Routines (Part 1 of 2)

Routine	Calling Sequence	Description
EFLG2$	L,U A0,x*/6+y DL A1,nbr-addr LMJ X11,EFLG2$ These instructions can be generated by the procedure call: EFLG2$ x*/6+y,nbr-addr	Same as EFLG1$ except that the floating-point number is in registers A1 and A2.
EFLS2$	L,U A0,x*/6+y DL A1,nbr-addr LMJ X11,EFLS2$ These instructions can be generated by the procedure call: E$FLS2 x*/6+y,nbr-addr	Same as EFLS1$ except that the floating-point number is in registers A1 and A2.

Table 14-3. Floating-Point Editing Routines *(Part 2 of 2)*

14.3. EOUT$ (GENERALIZED OUTPUT EDITING ROUTINES)

EOUT$ is an interpretive routine which performs editing functions for output produced on the line printer, the card punch, and the display console. The interpretive instructions performed by the routine are constructed along much the same lines as are machine language instructions:

f	t	d	x	m
35 31	30 24	23 18	17 16	15 0

where:

f — Function code

t — Type wheel (printer) or character position (display console)

d — Decimal point location

x — Specifies indirect address and use of the simulated index register

m — Address (main storage location of data)

EOUT$ is called by:

```
LMJ X11,EOUT$
```

There are two entry points to this subroutine. The normal entry point is EOUT$. The other, EOUTR$, is the point for reentry after E$TERM (terminate) function and is discussed in 14.3.4.

The addressed word in the m field may be either in control register or main storage. Any word, even a volatile register, is permissible; but if register X11 is addressed, the location of the interpretive word which references X11 is put out. All registers, including volatile ones, are saved and restored. The x field is used to specify indirect addressing and the use of the single simulated index register. Its permissible values are:

0_8	–	No action
1_8	–	Use address indirectly
2_8	–	Apply simulated index register
3_8	–	Apply simulated index register then use address indirectly

Indirect addressing is permitted to one level only, and the x, h, and i fields of the indirectly addressed word are ignored. It is possible, however, to indirectly address control storage. All modes may be used with indirect addressing.

The various functions are described in the following paragraphs. They are all callable as procedures. Each of the procedure calls generates one word in the proper format. The parameters of these procedures are interpreted differently depending on the number written. A single parameter is taken as m; two parameters as m and x; three parameters as t, d, and m; and four parameters as t, d, m, and x. Any missing parameters are assumed to be zero.

Entry to EOUT$ may be obtained by the procedure E$OUT or E$OUTR, depending on the entry point desired. No parameters are required.

14.3.1. EDITING FUNCTIONS

These functions actually convert the information to be outputted. In all cases, except E$A (alphanumeric words), the t field specifies the type wheel at which the rightmost digit, bit, or character is to be printed. The number given in parentheses following the procedure call is the octal function code.

E$D(01) – Decimal: The address word is treated as if it were a signed decimal integer and is edited without a decimal point unless a set function (see E$PNT – 14.3.3) is in effect. Leading zeros to the left are suppressed and a minus sign, if any, is printed immediately to the left of the number (also see E$OVRP – 14.3.3). If the value is zero, a single zero is printed. If a set point is in effect, the decimal number is assumed to have the stated point specified by the set point, and the d field specifies the number of decimal digits to be printed to the right of the decimal point. If a set field function (see E$FLD – 14.3.3) with D=0 is in effect, the specified field is treated as an unsigned decimal integer.

E$O(02) – Octal: The d low-order bits of the addressed word are edited and printed as (d+2)/3 octal digits, unsigned. For a full octal, binary, or alphanumeric character word, d must always be given as 36.

E$B(03) – Binary: The d low-order bits of the addressed word are edited as d binary digits unsigned.

E$C(04) – Alphanumeric Characters: The d low-order bits of the addressed word are edited and printed as (d+5)/6 alphanumeric characters in Fieldata code.

E$A(05) – Alphanumeric Words: The d words beginning with the addressed word are edited as 6*d characters in Fieldata code. For this editing function only, the t field specifies the print position at which the left-most character is printed.

E$E(06) – Floating Point (FORTRAN E): The addressed word is edited as a floating-point number with d significant digits. Normally these are all printed to the right of the decimal point (see E$SCL – 14.3.3). A decimal exponent consisting of a sign and two digits is inserted immediately to the right of the significant portion. If the floating-point number is negative, a minus sign is inserted immediately to the left of the number (see E$OVRP – 14.3.3). If the addressed word is minus zero, no effect will occur, and the field is left blank.

E$F(07) – Floating to Fixed (FORTRAN F): The addressed word is assumed to be a floating-point number and is edited to fixed point with d places following the decimal point. Negative numbers, including minus zero, are treated as in E$E.

E$DE(26) – Double-Precision Floating Point: This editing function is the same as E$E with the addressed word and the addressed word plus one edited as a double-precision floating-point number. A decimal exponent consisting of a sign and three digits are inserted immediately to the right of the significant portion.

E$DR(27) – Double–Precision Floating to Fixed: This editing function is the same as E$F with the addressed word and the addressed word plus one edited as a double-precision floating-point number.

14.3.2. OUTPUT FUNCTIONS

The output functions serve to transmit the edited line to an output device; the printer, the card punch, or the display console. The device to be used is determined by the d field:

Printer	D=0
Card Punch	D=1
Display Console	D=2

The word or character count is given in the t field.. This count must be given. (It is not assumed maximum if it is given as zero.) For the printer, the word count is normally 22; for the card punch, normally 14. For the display console, the t field is a character count and cannot be more than 60. For the printer, the m field serves to specify the number of lines to be spaced. A value greater than the length of a logical page results in printing on the first line of the next page. For the punch and display console, the m field is ignored. The number given in parantheses following the procedure call is the octal function code.

E$WT(10) – Write and Terminate: The edited image is transmitted to the specified device, and the routine returns control to the next instruction in machine language mode. The image is not reset to blanks.

E$W(11) – Write: The edited image is transmitted to the specified device and the routine continues in the interpretive mode. The image is reset to blanks.

E$WS(12) – Write and Save: The edited image is transmitted to the specified device and the routine continues to the next instruction in the interpretive mode. The image is left available for use by further output functions or further editing.

14.3.3. MODAL FUNCTIONS

The modal functions serve to enter information which affects the interpretation of one or more of the instructions which follow. The number given in parentheses following the procedure call is the octal function code.

E$SCL(13) – Set Scale: The contents of the address field are treated as a signed power of 10 to be applied to any floating-point or floating-to-fixed function which follows the set scale function. For floating point, the scale is the number of digits to be printed to the left of the decimal point. The exponent field is reduced accordingly, so that the resulting value is the same as if no set scale function were in effect. Negative values of the address (the 16-bit ones complement) introduces leading zeros after the decimal point and increases the exponent field accordingly.

For floating-to-fixed conversion, the actual value of the resulting number is altered by multiplying it by the power of 10 indicated by the address. The set scale function remains in effect until it is countermanded by a new set scale. Upon initial entry to EOUT$, the scale is assumed to be 0.

E$PNT(14) – Set Point: The set point function specifies the position of the binary point for the next editing function to be encountered (presumably a decimal editing function). It remains in effect only for the single edit. The address of the set point gives the number of bits following the binary point. Negative values are permitted (see E$FLD – below).

E$FLD(15) – Set Field: The set field function is used to specify a subfield of the next word to occur (presumably a decimal, octal, binary, or alphanumeric characters function). The t field specifies the lefthand margin and the m field the righthand margin. The bits of the machine word are numbered, for the purposes of this function, from left (00) to right (35). The d field specifies extension of sign; if it is nonzero, the field is treated as signed. A set field function with d=0 and t=0 may be used to treat fields, including the sign bit, as unsigned unless m=35 (that is, a whole word must always be signed in the event a sign is applied).

The set field function remains in effect only for the next function encountered. If both a set field and a set point function are in effect when editing occurs, the set field function is applied first. In this case, the set point function specifies the binary point counting from the righthand end of the specified field.

E$INDX(16) – Set Index: The set index function is used to address a quantity in main storage which is to be loaded into the single simulated index register. For any function which addresses storage (including this one), the presence of a 1 bit in the increment (h) portion of the address causes the simulated index to be added to the specified address before access is made. The left half of the index register word is ignored. If the d field is nonzero, the contents of the m field (with sign extension) are loaded into the simulated index register. The set index function remains in effect until it is countermanded by another set index function.

E$OVRP(17) – Overpunch: The overpunch function specifies that any minus signs produced by the editing functions are to be removed from their positions in front of the edited numbers and placed as 11-punches over the low-order digits. In the case of floating-point editing, the sign of the mantissa is placed over the low-order digit of the mantissa and the sign of the exponent over its low-order digit. The space that would normally contain the sign of the exponent is omitted.

The overpunch function is initiated by its occurrence with address 1. It is countermanded by its occurrence with address 0. Upon initial entry to EOUT$, the overpunch mode is assumed to be off.

14.3.4. CONTROL FUNCTIONS

The control functions serve to introduce into the interpretive language some of the control operations available in machine language. The number given in parentheses following the procedure call is the octal function code.

E$TERM(20) – Terminate: The terminate function causes the routine to return to the next instruction in machine language. Upon reentry at point EOUTR$, all counters, modes in effect, interpretive subroutines, and any partial image are left undisturbed; control is returned to the next instruction in machine code. If reentry is made at EOUT$, these are all cleared; control is returned to the interpretive mode. Entry at EOUTR$ is made by:

```
LMJ    X11,EOUTR$
```

E$LINK(21) – Link: The link function is used to form subroutines in the editing language. Its effective address specifies the location of the entry to a subroutine. Subroutines may be nested to a depth of 10.

E$JUMP(22) – Jump: The jump function with a nonzero effective address causes an interpretive transfer of control to the designated location. If the address is zero, the jump function serves as a subroutine exit. Transfer is to the interpretive function following that link control most recently executed for which no exit has been performed.

E$RPT(23) – Repeat: The repeat function causes the next single interpretive function to be repeated the number of times specified in the d field of the repeat word. A repeat function preceding E$LINK is meaningless; for multiple execution of E$LINK, the routine EOUT$ itself should be called within a machine language loop. The t and m fields contain increments to the t and m fields of the instruction to be repeated for each execution. Any modes set by the modal functions which would be in effect for the first execution of a repeated instruction remain in effect for all executions.

E$CLR(24) – Clear: The clear function sets the image to blanks.

14.3.5. EXAMPLES

Several examples* of typical calling sequences to EOUT$ follow:

Example 1:

The FORTRAN instruction

```
    PRINT 100, A, I, N, B, C

100 FORMAT (6X, E20.7, I20, O20, 1P.2F20.6)
```

**The use of FORTRAN formats here is merely to indicate the format desired. The I/O functions in FORTRAN employ an editing scheme peculiar to themselves.*

is equivalent to the interpretive sequence:

```
E$OUT
E$E          26,7,A
E$D          46,0,I
E$0          66,36,N
E$SCL        1
E$F          86,6,B
E$F          106,6,C
E$WT         22,0,1
Next machine language instruction
```

Example 2:

If this line were to be put out also on the card punch, whose output code is 1, then the last interpretive instruction would be replaced by:

```
E$WS         14,1,0
E$WT         22,0,1
```

Only the first 80 columns of the image would be punched.

Example 3:

The FORTRAN instruction

```
    PRINT 100 (J (I), K (I), L (I), M (I), I=1,4)
100 FORMAT (20I6)
```

is equivalent to the following interpretive sequences:

```
E$RPT        30,4,1
E$D          6,0,J,2
E$RPT        30,4,1
E$D          12,0,K,2
E$RPT        30,4,1
E$D          18,0,L,2
E$RPT        30,4,1
E$D          24,0,M,2
E$WT         22,0,1
```

15. COMMUNICATIONS HANDLER

15.1. INTRODUCTION

The communications handler provides the interface between the multitude of available remote terminal devices and the user programs. The diversity of hardware dictates a general routine upon which the variances of each application can be built.

Worker programs to which communication devices are assigned must be operated as real time programs, because of the high priority which must be given to communication interrupt processing.

Each worker program to which communications devices are assigned should register an error routine with the executive so that the worker program may be properly notified concerning operating contingencies. The error routine is registered by means of an IALL$ request (see 4.9.3.1) for the error mode entry. If the error routine is omitted, a contingency causing entry into the error routine causes program termination.

15.1.1. EQUIPMENT

Communication devices may be connected to UNIVAC 1100 Series channels through three types of subsystems:

- Communications Terminal Synchronous (CTS)
- Word Terminal Synchronous (WTS)
- Communications Terminal Module Controller (CTMC)

15.1.1.1. THE CTS AND WTS

The CTS and WTS operate in the internally-specified index (ISI) mode with only one remote terminal connected to an I/O channel at any one time.

15.1.1.2. THE CTMC

The CTMC line terminals operate in the externally-specified index (ESI) mode. Each character transfer is accompanied by an address which identifies to the CPU the external line to or from which the transfer is directed, and each address has a distinct I/O access control word association. One CTMC is capable of multiplexing 64 line terminals: 32 for input and 32 for output.

In order for the system to utilize the main storage overlap feature, the system base address for all ESI access control words is 002100_8 for main storage. A group of 64 consecutive addresses (one for each line terminal) is set aside for each CTMC defined at systems generation time. Any ESI channel may be configured to operate in either quarter- or half-word mode. The mode is selected by hardware modification of the CPU's or IOC's ESI channel.

The communications handler assumes that the user program is written to interface with a particular type of hardware and that the buffers are organized accordingly, such as, packing characters into words for the WTS, setting up code conversion, providing start, end, and parity characters for the CTS, and so forth. The amount of time available to process characters dictates that executive action be kept to a minimum and thus disallows character manipulation. Other than for buffer format and time considerations, the user need not be aware of the hardware arrangement being employed.

15.1.2. MODES OF OPERATION

The real time user, while interfacing with the communication handler, may operate in the following modes:

- Single Mode:

  Used when one message, with a known maximum length, is to be received or sent.

- Pool Mode:

  Used primarily when multiple messages are to be received or sent, or when a message of variable and unknown maximum length is to be received. There are three types of pool mode operation:

  (1) Open Pool — A multiple number of buffers chained so that the chain can be exhausted. The last buffer in the chain indicates the end of the chain by a zero value in H2 of the link word.

  (2) Closed Pool — For input operations only. Contains a multiple number of buffers chained in a continuous manner, where the last buffer is chained to the first; that is, the pool is never exhausted. The last buffer in the chain points back to the first by means of H2 of the link word. Use of closed pool mode requires extreme user care.

  (3) Dual Pool — For input operations only. Contains an initial buffer pool for status checking, and an additional buffer pool (open or closed) for the receipt of data.

The user must weigh many factors before choosing a mode of operation. Such factors are devices (CTS/WTS are limited to single mode), carrier speed, type of acknowledgement required or not required in operating the device, and the manner in which the data is to be processed. The mode of operation need not be the same for both input and output.

15.2. ASSIGNING LINE TERMINAL (LT) DEVICES

At system generation time, each channel must be completely defined. For communications devices, this includes specifying the subsystem type and characteristics of the LT to which the remote terminal is connected (bits per character, speed, fixed or common carrier line, unit type). At that time, various devices connected to a single line, and programmed as a unit (that is, one input and one output and/or one dialing unit), are given a line terminal group (LTG) identity. This identity can be used in assigning communications devices. The arbitrary subsystem and unit assignment for these devices can also be used as specified in the @ASG control statement (see 3.7.1).

15.3. THE LINE TERMINAL TABLE

The user program controls each LTG identified by means of the line terminal table (LTT) constructed in the user-program D bank. All user operations on an LTG must reference that group through a single LTT (in contrast to other I/O devices which may have any number of active packets). The LTT enables the user program to control each LTG from the execution of the initialize request to the execution of terminate request (see 15.4.1).

The format of the LTT is divided into four parts: internal filename (words 0 and 1), output area (words 2 through 5), input area (words 6 through 9), and dial area (words 10 and 11). The user can omit any part simply by not coding it; however, omission of a part must be accounted for by a zero-filled area. The format of the LTT is:

Word	S1	S2	S3	H2	
0	internal-filename				
1					
2	*output-status*	quarter-word-indicator	output-usage	output-completion-activity-addr	output area
3	output-character-count (0 for pool mode)			output-buffer-or-pool-start-addr	
4	*end-of-output-backup-queue-addr*			*start-of-output-backup-queue-addr*	
5	*partial-buffer-character-count*			buffer-transfer-time	
6	*input-status*	end-of-input-action	input-usage	input-completion-activity-addr	input area
7	input-character-count (0 for pool mode)			input-buffer-or-pool-start-addr	
8	*end-of-input-backup-queue-addr*			*start-of-input-backup-queue-addr*	
9	partial-buffer-character-count-or-dual-pool-addr			buffer-transfer-time	
10	*dial-status*		dial-usage	dial-completion-activity-addr	dial area
11	dial-access-control-word				

- Internal filename

Words 0 and 1

Contains the identity used to reference the LTG. If the internal filename is not the same as the external filename, the internal filename must be equivalenced by a @USE control statement (see 3.7.5).

- Output Area

Word 2

output-status

An octal code denoting the completion status of the last buffer transferred to the remote terminal. For pool mode, this code is stored in S3 of word 0 of each buffer. Values for this field are:

0_8 – The number of characters specified in H1 of word 3 has been transferred to the remote terminal. For pool mode, T1 of the first buffer word has been transferred to the remote terminal.

2_8 – The line has been declared down by the operator. H1 of word 5 contains the number of characters transmitted. No further action is taken on queued buffers.

5_8 – Output was terminated before the specified number of characters was transferred. Termination is detected by the timeout of the communications handler's timer. The number of characters transferred is placed in H1 of word 5 and the output has been turned off. This mode of operation can be used to avoid output monitor interrupts on the CTMC by specifying a full buffer, setting the EOT bit before the end of the buffer, and allowing the buffer to timeout.

10_8 – An ESI activity associated with this line terminal is in a contingency state and as a result the line terminal is terminated.

20_8 – Same as 0_8, except that the end of the buffer queue is reached and output is turned off. If a buffer is added to the queue after the communications handler makes the check, the worker program must restart the output.

25_8 – Same as 20_8, except that the output buffer has timed out.

quarter-word-indicator

For ESI activity processing, either quarter- or third-word execution is permitted. If quarter-word mode is desired, a 1 is placed in this field. A 0 (or if the SGS for quarter-word states that quarter-word execution is not allowed) indicates third-word execution. In either case, the mode is applicable to all ESI activities created with the initialization request. Once the mode is established by the real time program, the only method to alter the mode is to terminate the line (by means of a CMT$ request – see 15.4.1.10) and reinitialize (by means of a CMS$ request – see 15.4.1.1) with the new mode. For each subsequent initialization request, the mode must be reestablished. This is required because the communications handler uses the field as a link to the program control table (PCT) item associated with the LTT after the initialization request.

output-usage

Denotes the condition required to start the output completion activity. Values for this field are:

0_8 – No activity is to be initiated.

1_8 – Give control to completion activity upon completion of each output buffer only if activity is not executing. The communications handler detects exiting from this activity after the check, and restarts it.

2_8 – Give control to the output completion activity only if the output backup queue is exhausted or a nonzero status is returned for a buffer.

output-completion-activity-addr — Contains the starting address of the output completion activity, which is a routine to be given control upon completion of an output buffer transfer and within the conditions specified by the output usage field (S3). This activity is given control, with register A0 containing the address of the LT, and register A1 containing the buffer address. For pool mode operations, register A1 contains the address of the first buffer transferred since more than one buffer may have been transferred prior to the activation of the completion activity. The activity specified must be within the bounds of the user program.

Word 3

This word defines the single output buffer or the output buffer pool. For the CTMC (half-word operation only), output characters are transmitted in ascending order within a word starting at the lowest portion of the word. For CTMC quarter-word transfers, see 15.4.2. For WTS, output characters are transmitted in descending order within a word starting at the highest portion of the word. CTS output characters are transmitted one character per word, right-justified. Both single mode and pool mode individual buffer sizes are limited to 4095 characters.

output-character-count — Contains the number of characters for single mode; contains zero for pool mode.

output-buffer-or-pool-start-addr — Contains the single buffer address, or the address of the buffer pool control word which is the first word of the buffer control packet.

Word 4

For pool mode, this word contains the starting and ending address of the queue of buffers already filled for output to the terminal device. H1 contains the ending address, and H2 contains the starting address. H1 and H2 must specify a buffer currently removed from the output queue and not currently in another output queue. The communications handler updates only H2. The user program can detect the end of the output queue by finding a start field with a value of zero in his LTT, or by finding a start field (H2) which is equal to the end field (H1).

Word 5

partial-buffer-character-count — Contains the number of characters transferred as output when the output transfer of a buffer is completed before the specified count is transferred.

buffer-transfer-time — Contains the number of basic time intervals to be used as the maximum time between buffers. If a buffer does not transfer in this time interval, a fault is suspected unless the previous character denoted end of output. If a time value of zero is specified, no timing check is performed by the communications handler. It is not intended that the communications handler perform extensive buffer timing checks, but merely that it provide a means of detecting a stalled or inactive condition. Any extensive timing checks would be excessive overhead which reduces system throughput, and would be of no appreciable value to the communications handler users.

The basic time interval used by the communications handler is 600 milliseconds for the CTMC and six seconds for the CTS/WTS to conform to ISI timing by the day clock because:

(1) the dayclock interrupts at this time interval

(2) the automatic calling unit dial time is approximately 600 milliseconds per digit

(3) some medium speed modems require a certain amount of time between transmissions, which can be measured in 600-millisecond increments.

■ Input Area

Word 6

input-status — Contains a value which denotes the completion status of the last buffer transferred from the remote terminal. For pool mode, this value is stored in S3 of word 0 of each buffer. Values for this field are given in Table 15-1.

Octal Code	Description
0	Normal acknowledgment was returned. For single mode, the number of characters transferred is in H1 of word 7; for pool mode, T1 of the first buffer word was transferred from the remote terminal.
1	Input was terminated by an external interrupt, The number of characters transferred is in H1 of word 9 or, for pool mode, in T1 of word 0.
2	The operator has declared the line to be down by using a DN keyin. H1 of word 9 or T1 of word 0 of the buffer contains the number of characters transmitted, if any. No further input is accepted.
3	Input was terminated before the specified number of characters were transferred, due to an input request (ER CMI$) before the previous request was completed by the handler. This applies only to pool mode; T1 of word 0 contains the partial character count, if any. Further action is specified in the end-of-input-action field which provides the user with the capability of terminating pool mode input.
5	A timeout was detected by the communications handler buffer timer.
6	A ring indicator external interrupt was received.
7	An external interrupt was received with a status word indicating either character or block parity error, or late input acknowledge.
10	An ESI activity associated with this line terminal is in a contingency state and as a result the line terminal is terminated.
11	A carrier off external interrupt was received.
12	A timeout external interrupt was received.
13	A space-to-mark transition external interrupt was received.
20	Same as for 0_8 except that the end of the input buffer pool has been reached; input is terminated.
21	Same as for 1_8 except that the end of the input buffer pool has been reached.
23	Same as for 3_8 except that the end of the input buffer pool has been reached.
25	Same as for 5_8 except that the end of the input buffer pool has been reached.
27	Same as for 7_8 except that the end of the input buffer pool has been reached.
30	Same as for 10_8 except that the end of the input buffer pool has been reached.
31	Same as for 11_8 except that the end of the input buffer pool has been reached.
32	Same as for 12_8 except that the end of the input buffer pool has been reached.
33	Same as for 13_8 except that the end of the input buffer pool has been reached.

NOTE: Ring indicator and carrier off interrupts may occur whether or not input is active. If either of these interrupts occurs and input is active, normal processing of the interrupt is performed with the buffer or data address corresponding to the interrupt passed in register A1 to the ESI activity along with the appropriate status code. If either of the above interrupts occurs and input is not active, register A1 is passed to the ESI input activity. Set to zero to indicate the abnormal condition. The appropriate status code is provided in the input status field of the corresponding LTT for the inactive input condition.

Table 15–1. LTT Input-Status Codes

end-of-input-action

For pool mode, denotes action to be taken by the communications handler when an end-of-message characters or external interrupt received. Values for this field are:

0_8 – Turn input off

1_8 – Reinitiate input using next buffer from pool.

input-usage

Contains a value which denotes the condition required to start the input completion activity. Values for this field are:

0_8 – No activity is to be initiated.

1_8 – Give control to the completion activity only if the activity is not executing. (The communications handler detects exiting from this activity after the check and restarts it.)

input-completion-activity-addr

Contains the address of a routine to be given control upon completion of an input buffer transfer as indicated by the input usage field. Control is given to this routine with register A0 containing the address of the LTT and register A1 containing either the address of the first buffer transferred (pool mode) or the address of the user's data area (single mode). For pool mode, more than one buffer may have been transferred prior to giving control to the completion activity (see word 8).

Word 7

Defines the single input buffer or the input buffer pool. For CTMC half-word transfers, input characters are transmitted in ascending order within a word starting at the lowest portion of the word. For CTMC quarter-word transfers, see 15.4.2. For WTS, input characters are transmitted in descending order within a word starting at the highest portion of the word. CTS input character are transmitted one character per word, right-justified. For both single mode and pool mode, individual buffer sizes are limited to 4095 characters.

input-character-count

Contains the number of characters for single buffer mode; must not be a value that would cause the buffer to extend beyond the upper boundary of the user's D bank; must contain zero for pool mode.

input-buffer-or-pool-start-addr

Contains the single buffer address (single mode), or the address of the buffer pool control word, which is the first word of the buffer mode).

Word 8

For pool mode, this word contains the starting and ending addresses of the queue of buffers already filled as input from the terminal device. This word is not used in single mode. H1 contains the ending address, and H2 contains the starting address. After each buffer is processed, the user program is expected to update H2 with the link portion of that buffer. To accomplish this update, the user program can determine the end of the input backup queue by finding a start field with a value zero in his LTT, or by finding a start field (H2) which is equal to the end field (H1).

Word 9

partial-buffer-character-count-or-dual-addr

For single mode, H1 contains the number of characters transferred as input when the input transfer of a buffer is completed before the specified count is transferred. For input pool mode, this field must be zero unless dual pool mode is desired; for dual pool mode, H1 contains the dual pool address. Dual pool mode is used primarily for polling operations, whereby a small input buffer can be initially set up so that an immediate switch to a pool of larger buffers occurs when the poll response is received. Thus, larger buffer areas can be used for the input data stream initiated by the polling operation.

buffer-transfer-time Contains the number of basic time intervals to be used as the maximum time between buffers. If a buffer does not transfer in this time interval, a fault is suspected unless the previous character denoted end of input. If a time value of zero is specified, no timing check is performed by the communications handler (see word 5).

- Dial Area

Word 10

dial status Contains completion status of the last dial operation. Status codes are:

1_8 – Successful

2_8 – Unsuccessful

3_8 – Leased line assigned

10_8 – An ESI activity associated with this line terminal is in a contingency state and as a result the line terminal is terminated.

40_8 – In progress

dial-usage Denotes action to be taken upon completion of the dial operation:

0_8 – No dial completion activity

1_8 – Give control to dial completion activity when dial operation is completed

dial-completion-activity-address Upon completion of the dial operation, control is passed to the address specified in this field.

Word 11

This word contains the count of characters in H1, and in H2 the buffer address at which the number to be dialed is stored in BCD format. The CTMC dial characters are transmitted in ascending order within a word starting at the lowest portion of the word. The communications handler uses the buffer timer to verify that a connection has been established. This verification is contingent upon the user setting the EOT bit (2^9) in the last dial digit.

15.4. COMMUNICATIONS HANDLER OPERATIONS

15.4.1. SUPPORT OPERATIONS

The available operations are:

Initialize	ER CMS$ (see 15.4.1.1)
Terminate	ER CMT$ (see 15.4.1.10)
Dial	ER CMD$ (see 15.4.1.2)
Input	ER CMI$ (see 15.4.1.3)
Output	ER CMO$ (see 15.4.1.4)
Input and Output	ER CMSA$ (see 15.4.1.5)
Hangup	ER CMH$ (see 15.4.1.9)

The references to these operations are made with H2 of register A0 loaded with the address of the LTT defining the LTG.

15.4.1.1. INITIALIZATION (CMS$)

Purpose:

To initialize one or more LTG's.

Format:

```
L    A0,(lttcount,lttaddr)
ER   CMS$
```

Parameters:

lttcount — The number of LTT's (see 15.3)

lttaddr — The address of the first LTT

Description:

When the CMS$ request is executed, the user program must be in real time mode, and H1 of register A0 must contain the number of LTG's minus one; H2 must contain the location of the first LTT. The format of register A0 is:

H1	H2
LTT-count-minus-one	first-LTT-addr

where:

LTT-count-minus-one — Specifies the number (minus one) of the contiguously located LTT's which are to be initialized and which must not exceed the number of communications LTG's (minus one) currently assigned to the user.

first-LTT-addr — Specifies the address of the first of the contiguously located LTT's to be initialized.

Each LTT in the user's D bank must be formatted for the initialize operation. Refer to 15.3 for the format of the LTT.

When a CMS$ request is made, the LTT's specified by the user are initialized, in turn, beginning with the LTT specified in H2 of register A0. Thus, when an error condition is detected and control is returned to the user's error contingency routine, all of the LTT's preceding the one being initialized when the error condition is detected have been initialized. The user may elect to use the initialized LTT's or terminate them by executing a CMT$ request (see 15.4.1.10).

15.4.1.2. DIALING (CMD$)

Purpose:

Initiates a communications handler dialing operation.

Format:

```
L,U  A0,lttaddr
ER   CMD$
```

Description:

The user program must be in real time mode, and register A0 must contain the location (lttaddr) of the LTT (see 15.3).

The LTT in the user's D bank must be formatted for the dial operation (see 15.3).

The dial completion activity operates as a high priority interrupt routine and is allowed a limited time period to perform analysis of the interrupt. When this routine is entered, register A0 contains the location of the appropriate LTT.

The dial operation initiates the buffer specified in word 11 of the LTT addressed by register A0. The telephone number to be dialed must be in BCD. After the dial operation is completed, the user gets control at the dial completion activity starting point, with the dial status in S1 of word 10 of the LTT. If input or output was initiated before dialing, the character timing is not done before either the first character transfer or the dial times out. In the case of initiating output, the output terminal requires an enable from the dial unit, so transfer starts immediately upon dial completion. A dial request on a leased line is given a unique status code and subsequent input or output is honored. The dial completion activity operates as an ESI completion activity.

If an automatic calling unit does not exist for an LTG, the following message is displayed on the operator's console:

```
run-id   DIAL   NUMBER .......... Sxxx/Uxxx Y OR N?
```

where Sxxx/Uxxx is the subsystem and unit to be connected. The operator must respond with an N or Y to indicate completion.

15.4.1.3. INPUT (CMI$)

Purpose:

Initiates a communications handler input operation.

Format:

```
L,U  A0,lttaddr
ER   CMI$
```

Description:

When a CMI$ request is made, the user program must be in real time mode, and register A0 must contain the location (lttaddr) of the LTT (see 15.3).

Input on line terminals can be initiated in one of the following modes:

- Single buffer
- Pool
- Dual pool

15.4.1.4. OUTPUT (CMO$)

Purpose:

Initiates a communications handler output operation.

Format:

```
L,U  A0,lttaddr
ER   CMO$
```

Description:

When a CMO$ request is made, the user program must be in real time mode, and register A0 must contain the location of the LTT (see 15.3).

Output on line terminals can be initiated in one of the following modes:

- Single buffer
- Pool
- Dual pool

15.4.1.5. SEND AND ACKNOWLEDGE (CMSA$)

Purpose:

Initiates a communications handler input and output operation.

Format:

```
L,U  A0,lttaddr
ER   CMSA$
```

Description:

When a CMSA$ request is made, the user program must be in real time mode and register A0 must contain the LTT (see 15.3) address (lttaddr).

15.4.1.6. SINGLE BUFFER MODE FOR INPUT/OUTPUT OPERATIONS

If a request to CMI$ (see 15.4.1.3), CMO$ (see 15.4.1.4), or CMSA$ (see 15.4.1.5) is for single buffer mode, the LTT (see 15.3) must be currently initialized for either operation by setting up the appropriate word of the LTT. Word 7 is set up for input operations; word 3 is set up for output operations. If the LTT is initialized for single buffer mode input, the CMI$ request must be for single buffer mode input. Similarly, if the LTT is initialized for single buffer mode output, the CMO$ request must be for single buffer mode output.

For single buffer mode input, a CMI$ request causes initiation of input according to the input access control word (IACW) in word 7 of the LTT; the IACW points to word 0 of the input buffer.

For single buffer mode output, a CMO$ request causes output according to the output access control word (OACW) in word 3 of the LTT; the OACW points to word 0 of the output buffer.

Completion of an I/O operation is indicated by an external interrupt for the CTS and WTS subsystems. For the CTMC, the completion of an I/O operation is indicated by a monitored or external interrupt or a timeout. The timeout method is also allowed for the CTS and WTS input operations.

After the input interrupt is detected, the input line terminal is turned off. The completion status of the input buffer is stored in S1 of word 6 of the LTT. The number of characters accepted as input is stored in H1 of word 9. For synchronous line terminal devices working on a CTMC, there may be extraneous characters in the buffer following the last data character, if the message received is shorter than the buffer.

The number of output characters transferred in the single buffer mode is stored in H1 of word 5 of the LTT before the output completion activity routine is activated.

The completion activity for either input or output operations is activated with register A0 set to the starting address of the LTT controlling the buffer, and with register A1 set to the address of the buffer just transferred. The address of the input completion activity is specified in H2 of word 6 of the LTT; H2 of word 2 of the LTT contains the address of the output completion activity routine. This routine is given control as an ESI completion activity. No output completion activities are given control for CTS and WTS subsystems.

H1 of word 9 of the LTT contains the number of input characters before the input completion activity is activated. H1 of word 5 contains the number of characters transferred as output if the output of a buffer is completed before the specified count is transferred.

H2 of word 9 of the LTT contains the number of basic time intervals to be used as the maximum time until the input buffer is filled. For output buffers, H2 of word 5 contains a similar value. The basic time interval used by the system is 600 milliseconds for I/O buffer transfers. If a buffer is not filled in the time interval specified, a fault is suspected unless the previous character indicated end-of-message. The appropriate completion activity is activated.

15.4.1.7. POOL MODE FOR I/O OPERATIONS

If a request to CMI$ (see 15.4.1.3), CMO$ (see 15.4.1.4), or CMSA$ (see 15.4.1.5) is for pool mode input or output, respectively, the LTT (see 15.3) must be currently initialized for pool mode input or output operations. All CMI$, CMO$, or CMSA$ requests must be for pool mode, and there must be at least one buffer currently available in the pool.

For input, pool mode is indicated by a zero value in H1 of word 7 of the LTT. H2 of word 7 of the LTT points to a pool control packet which locates the pool of chained buffers. The value specified in H2 of word 7 is the address of the pool control word returned in H2 of register A0 upon return from a CPOOL$ request (see 15.4.2.1).

For output operations, pool mode is indicated by a zero value in H1 of word 3 of the LTT. H2 of word 3 points to a pool control packet which locates the pool of chained buffers. The value specified in H2 of word 3 is the address of the pool control word returned in register A0 upon return from a CPOOL$ request (see 15.4.2.1).

The output queue may contain any number of buffers between one and the total number in the pool. These buffers must be currently removed from the output pool and not currently in another queue.

In the input pool mode, input is initiated with monitor. As each input buffer is filled, it is added to the backup queue specified in H1 of word 8 of the LTT. The user program is expected to update H2 of word 8 with the link portion of each buffer after that buffer has been processed. The user program can determine the end of the input backup queue by updating H2 of word 8 until either a value of zero is encountered or it matches H1 of word 8. The first word of the buffer that is added to the backup queue is loaded with the completion status in S3 of word 0 and the number of input characters transferred in T1 of word 0. Depending upon the input-usage field, S3 of word 6 of the LTT, the input completion activity is activated. A monitored interrupt causes buffer switching. Upon receiving an end-of-message indication, S2 of word 6 of the LTT is tested whether to set up input into another buffer (nonzero value) or to turn off the input operation until the next request by the user program.

In the output pool mode, each buffer is transferred with monitor, and upon interrupt, the next buffer in the backup queue is initiated. As each buffer is removed from the queue, the communications handler updates the start of the output-backup-queue field (H2 of word 4 of the LTT). The user program adds to the queue by updating H1 of word 4 of the LTT. As each output buffer is emptied, the communications handler stores the completion status in S3 of word 0 of the buffer and the number of characters transferred in T1 of word 0. Depending upon the contents of the output-usage field (S3 of word 2 of the LTT), the output completion activity is activated. The communications handler examines the start of the backup-queue field (H2 of word 5) when the output is first initiated, then works from the link field (H2 of word 0 of the buffer) to determine the end of the chain. The user program must add the completed buffers back to the available pool. When the end of the backup queue is reached, the communications handler turns off the output and returns a status code of 20_8 to denote the caught up condition. If the user program submits new buffers, the output must be reestablished by a CMO$ request (see 15.4.1.4) and the start of the backup queue must be reset.

Both the input and output completion activities are activated with register A0 set to the starting address of the associated LTT and register A1 containing the address of the buffer or user's data area. The completion activity is given control as a high priority interrupt processing routine and is therefore allowed minimum time for analysis of the interrupt. Completion activity timing considerations are discussed in 15.5.

15.4.1.8. DUAL POOL MODE FOR INPUT OPERATIONS

When using dual pool mode for input operations, word 7 and H1 of word 9 of the LTT (see 15.3) are used for dual pool mode input. Dual pool mode for input is specified when H1 of word 7 is equal to zero and H1 of word 9 points to a buffer pool control packet different from the buffer pool control packet specified in H2 of word 7. The first monitor interrupt causes buffer switching with input initiated using the next buffer in the pool, indicated by H2 of word 7 in LTT. From there on, the dual buffer pool mode works exactly like the normal buffer pool mode for input.

15.4.1.9. HANGUP (CMH$)

Purpose:

Initiates a communications handler hangup operation.

Format:

```
L,U  A0,lttaddr
ER   CMH$
```

Description:

When a request is made to CMH$, the user must be in real time mode, and register A0 must contain the address (lttaddr) of the LTT (see 15.3).

The CMH$ request releases the current remote connection. At the time the CMH$ request is made, the user program must have ensured that output operations are completed, and that any input operations that may occur are of no concern. The CMH$ request disregards the current line activity and issues a remote release to the dial and input line terminals. Any further activity after the hangup must be preceded by a CMD$ request (see 15.4.1.2) to activate the terminal.

If no automatic dialing exists for an LTG, a CMH$ request displays the following message on the operator's console:

```
HANGUP    subsystem/unit
```

No response is required by the operator for this message.

15.4.1.10. TERMINATION (CMT$)

Purpose:

Deactivates the input and output line terminals and performs various housekeeping functions associated with an LTG.

Format:

```
L,U  A0,lttaddr
ER   CMT$
```

Description:

When a request is made to CMT$, the user must be in real time mode, and register A0 must contain the address (lttaddr) of the LTT (see 15.3).

The CMT$ request terminates the line terminal associated with the LTT. The assignment for the device is not released from the user program; it can be reinitialized by CMS$ (see 15.4.1.1). A communications device is released either by a @FREE control statement (see 3.7.4) or when the program terminates.

15.4.2. COMMUNICATION POOLS

A pool for use with the communications handler may be established in any portion of the user's main storage area. The system furnishes the subroutines to:

- establish a communications pool (CPOOL$ – see 15.4.2.1)
- remove buffers from pool (CGET$ – see 15.4.2.2)
- return buffers to pool (CADD$ – see 15.4.2.3)
- expanding a communications pool (CJOIN$ – see 15.4.2.4)
- release one pool or all pools (CREL$ – see 15.4.2.5)

The format of each buffer in a pool is:

	T1	S3	H2
Word 0	character-count	input-output-status	next-buffer-addr
1 → n	data characters		

Word 0

character-count

Specifies the number of characters transferred to or from the buffer. For input buffers to be processed by the input completion activity routine, the character count for completely filled buffers is the value which is specified by the pool start information in H2 of word 8 of the LTT (see 15.3).

For partially filled input buffers, which may occur as a result of either a timeout or an external interrupt, this field contains the number of characters in the buffer when the timeout or external interrupt occurred.

For output buffers, the character count represents the number of characters to be transmitted from this buffer. Character count for output buffers can be dynamically supplied by the user so that he may specify either partially or completely filled buffers in any order.

input-output-status

Contains the status code for this buffer. Status codes in this field are identical to those described for S1 of word 6 (for input) and S1 of word 2 (for output) of the LTT (see 15.3).

next-buffer-addr

Specifies the address of the first word of the next buffer in the chain of those currently linked together. A value of zero in H2 of the last buffer in an open chain is interpreted as the end of the chain.

Words 1 to n

Data characters start at word 1. The value in the character count field determines the value of n.

The format for data received or transmitted for CTMC quarter-word mode is to or from successive quarters of the input or output data area as follows:

Q1	Q2	Q3	Q4
character-1	character-2	character-3	character-4
character-5	character-6	character-7	character-8
		⋮	
character-(n-3)	character-(n-2)	character-(n-1)	character-n

The format for data received or transmitted for CTMC half-word mode is to or from successive halves of the data area as follows:

H1	H2
character-2	character-1
character-4	character-3
⋮	
character-n	character-(n-1)

15.4.2.1. ESTABLISHING A COMMUNICATIONS POOL (CPOOL$)

Purpose:

Establishes a pool of I/O buffers for communications usage.

Format:

```
L,U    A0,pktaddr
ER     CPOOL$
```

Description:

When a CPOOL$ request is made, the user program must be in the real time mode. The format of the CPOOL$ packet is:

		S1	S2	S3	H2
Word	0	mode	char-count		addr-of-first-buffer
	1	not-used		method	length-of-area-to-be-used

Word 0

mode

Word mode indicator. A value of 0 indicates half-word mode and a value of 1 indicates quarter-word mode. For half-word mode, the user's data area is established in individual buffers equal in length to the character count divided by two, plus one for the buffer control word. For quarter-word mode, the process is similar with the exception that the character count is divided by four. If any divisions yield a remainder, a value of one is added to the quotient. Quarter-word and half-word buffer formats differ only in the total length required to establish the individual buffers.

char-count

The number of characters to be used for each buffer area to be established. The value specified in this field should be the optimum value for the application. The value specified may be either an odd or even number but it must not exceed the maximum communications buffer length defined at systems generation time. One main storage location is assigned to the individual buffer area for each group of two characters (half word) or four characters (quarter word).

addr-of-first-buffer

Specifies the starting main storage address of the area to be set up as an input/output buffer pool.

Word 1

not-used

Not used by any of the buffer pool control routines and is available to the user for any purpose.

method

Defines the method of pool buffering for which the established buffers are to be used. Two methods of pool buffering are permitted.

The first method is similar to that employed by the block buffering package and is referred to as the open chain method. Each individual buffer is linked to the next buffer in the pool by the value in H2 of word 0 of each buffer except for the last buffer in the chain which has a value of zero in its link field. The zero in the link field causes the pool to be be open ended, hence the name, open chain method.

The second method of pool buffering is referred to as the continuous chain method. Each individual buffer is linked to the next buffer in the pool in the same manner as employed by the open chain method except the last buffer in the sequence is linked back to the very beginning of the pool, thus forming a continuous chain.

The open chain method is the preferred method because each buffer is removed from the pool by the communications handler as input data is received and is not used again until it has been returned to the pool by the user. In the use of the continuous chain method, an individual buffer is never really removed from the pool but rather the buffers in the pool are used in sequentially cyclic manner. This may result in the reuse of a buffer before all of its previous contents had been processed because either the buffer is of an insufficient size for the application or the real time program may be spending excessive time in its buffer processing.

One of the more frequent uses of the continuous chain method is to employ two individual buffers chained to each other, thereby operating in an alternating toggeling manner. When the continuous chain method is employed, the user assumes all responsibility for processing individual buffer contents in the required time interval. For the preferred open chain method, the communications handler ensures that no buffer is reused until directed by the real time program. For either method, the optimum size buffer must be used for the application.

length-of-area-to-be-used — Specifies the length of the main storage area to be used for the pool. The setup routine continues to establish individual buffers of the specified size in the desired method until this value is exhausted.

Upon return from the CPOOL$ request, a pool-id is contained in H2 of register A0. The user is expected to place the pool-id in every LTT (see 15.3) sharing the pool. If the pool is used for output, the pool-id is placed in H2 of word 3 of the LTT. If the pool is used for input, the pool-id is placed in H2 of word 7 of the LTT. A pool may be used for both input and output by placing the pool-id in H2 of register A0 in both word 3 and word 7 of the LTT. Please note that the pool-id returned in H2 of register A0 is an executive linking value to be used only for executive control of the buffer pool. The only circumstance under which the real time program can use this value is in providing it as information for an executive request such as CGET$.

15.4.2.2. REMOVING BUFFERS FROM A POOL (CGET$)

Purpose:

Removes buffers from the pool only when the open chain method is employed. Any number of buffers can be removed from the pool for future exclusive use by the requestor.

Format:

```
ER    CGET$
```

Description:

The program must be in real time mode at the time of the CGET$ request.

To remove buffers from the pool, register A0 must contain the following:

H1	H2
nbr-of-buffers-to-be-removed	linking-value-for-pool

The return of control from the CGET$ request is with the information provided in register A0. H1 of A0 is the actual number of buffers removed from the specified pool; this value is normally the number of buffers requested unless less than that specified number existed in the pool at the time of the CGET$ request. H2 is the starting address of the buffers removed. Each buffer removed is linked to the others by the open chain method.

15.4.2.3. RETURNING BUFFERS TO A POOL (CADD$)

Purpose:

Returns a number of buffers to the pool.

Format:

```
L,U   A0,pktaddr
ER    CADD$
```

Description:

The program must be in real time mode at the time of the CADD$ request and control register A0 is loaded with the address of a packet containing the following information:

		H1	H2
Word	0	nbr-of-buffers-returned	linking-value-to-buffer-pool
	1		addr-of-first-buffer

A buffer can be returned only to the same pool from which it had been previously removed so that buffer size consistency can be maintained within a pool. Each buffer returned to the pool is expected to be linked to the others by the open chain method.

15.4.2.4. EXPANDING A POOL (CJOIN$)

Purpose:

Expands or adds to a previously established pool by joining it to an additional pool area.

Format:

```
L,U  A0,pktaddr
ER   CJOIN$
```

Description:

H2 of A0 must be loaded with the address of a two-word packet whose format is:

		H1	H2
Word	0	name-of-pool-to-be-expanded	starting-addr-of-added-pool-area
	1		length-of-added-pool-area

In addition to loading register A0 with the packet address, the worker program must be in real time mode and should have an error contingency routine registered with the executive (see 4.2.9).

15.4.2.5. RELEASING COMMUNICATIONS POOL (CREL$)

Purpose:

Enables symbionts or real time user programs to release buffer pools obtained through a CPOOL$ request (see 15.4.2.1), a specified buffer pool, or all buffer pools associated with a real time program.

Format:

Two formats are available for ER CREL$.

To release all pools:

```
LXI,U  A0,1
ER     CREL$
```

To release one pool:

```
L    A0,pool-id
ER   CREL$
```

Description:

The buffer pool-id is returned to register A0 on a CPOOL$ request (see 15.4.2.1).

A pool-id need not be specified for the release of all pools.

15.4.3. ALTERING COMMUNICATIONS PATHS (ROUTE$)

Purpose:

Dynamically alters the primary paths of communications LTG's.

Format:

```
L    A0,(mode,lttaddr)
L,U  A1,pointer
ER   ROUTE$
```

Parameters:

mode	1_8—Request alternate input LTG path
	2_8—Request alternate output LTG path
	3_8—Request alternate LTG paths for both input and output
lttaddr	The address of the LTT (see 15.3)
pointer	Specifies a logical alternate of the primary LTG, such as 1, 2, 3, ..., n

Description:

Each primary LTG is defined at system generation time and provides the necessary information for the communications handler and facility inventory to make assignments by means of the @ASG control statement (see 3.7.1). The primary LTG may define up to three terminals: input, output, and dial; therefore, an alternate LTG configuration may define one, two, or three terminals.

An alternate LTG path may be another assignable LTG primary path or may be assignable only by the ROUTE$ request. If a primary LTG path is reassigned (altered) to another primary LTG path, both input and output line terminals should be altered; otherwise, the first primary LTG path remains in an assigned state and cannot be reassigned. Once a primary LTG path has been altered, all dial and hangup operations are initiated using the new LTG path. If, however, the primary LTG path was only partially altered (input or output, but not both), the communications handler provides dialing specified by the output line terminal alternate onlv.

It is the responsibility of the user to perform manual dialing for a LTG not dialed by the communications handler. If both input and output line terminals of a primary LTG path are altered, the handler performs the hangup operation for the primary LTG path.

15.4.3.1. ROUTING PROCEDURES

Before a ROUTE$ request can be made to the communications handler, the user's LTT (see 15.3) must be initialized by using the CMS$ request (see 15.4.1.1). Once the LTT has been initialized, the ROUTE$ request may be referenced as frequently as desired. Once a primary LTG has been routed, the primary LTG is not available for this assignment. Either a @FREE control statement (see 3.7.4) must be initiated or the primary LTG may be an alternate of its alternates and the ROUTE$ request may be initiated to reestablish the original assignment. If the primary LTG being routed has an idle line monitor state and both input and output are routed, the primary LTG is reestablished in the idle line monitor state.

15.5. COMPLETION ACTIVITIES

Completion activities, called ESI activities, are given control at the occurrence of ESI interrupt for CTMC subsystems or ISI input interrupt for the CTS or WTS subsystems. Completion activities are initialized by the communications handler when given a CMS$ request (see 15.4.1.1).

The user can request completion activities for input, output, and dialing line terminals by specifying a usage code and completion activity location in the LTT for each respective mode of operation.

When control is given to an ESI activity, register A0 contains the address of the respective LTT (see 15.3); register A1 contains the address of the first word of the user data area for single buffer mode; register A1 contains the address of the user's buffer location for the buffer pool mode. The complete set of A, X, and R registers may be used in ESI activities, but their contents are not passed on or restored when the activity releases control. Also, the control registers are not passed on between real time and ESI activities.

ESI activities are given control as high priority interrupt processing routines. These activities are interrupted only by the real time clock, the day clock, and by both ESI and ISI interrupts; therefore, it is necessary that these activities be timed to detect closed-loop situations and other excessive computation. The time quantum is variable and can be changed at systems generation. The quantum used should reflect that the ESI activity is interruptable as previously described and should account for the time lost in processing those interrupts. If the ESI activity execution time exceeds the specified amount, the activity is placed in a contingency state, as described in 4.9.5.

The following requests may be used to allow an ESI activity to release control:

- ER EXIT$ (see 4.3.2.1)
- ER ACT$ (see 4.3.3.4)
- ER CADD$ (see 15.4.2.3)
- ER ADACT$ (see 15.5.1)

Reference to any other request causes an ESI contingency condition.

Release of control by an ESI activity should not be confused with activity termination. ESI activities are terminated only by a CMT$ request (see 15.4.1.10) or a @FREE (see 3.7.4) of the LTG.

15.5.1. EXITING FROM AN ESI ACTIVITY (ADACT$)

Purpose:

Used to exit from an ESI activity, return specified buffers, and activate a previously named activity.

Format:

```
L     A1,name
L,U   A0,pktaddr
ER    ADACT$
```

Description:

The name parameter specifies the symbolic name contained in register A0 as a result of a NAME$ request (see 4.3.3.2).

For ADACT$ requests, register A0 must contain the address of a two-word packet whose format is:

H1	H2
nbr-of-buffers-returned	linking-value-to-parent-buffer-pool
	addr-of-first-buffer

The format of this packet is identical to the format of a two-word packet used for the CADD$ request (see 15.4.2.3).

15.6. IDLE LINE MONITOR

At system generation, the input line terminal devices can be given an unassigned status of either off or standby. The standby status causes input to be enabled on the devices when not assigned to a program. Upon receipt of a particular identifying character string the appropriate symbiont is initiated.

Some data sets, with the ring indicator feature, are designed not to release (hangup) after a ring indicator interrupt until a remote release function is received by the data set. If a terminal is initialized for idle line monitoring and after a ring indicator interrupt (no data is received or the received remote-id is invalid), a remote release must be issued or a redial from the remote site is inhibited. If a terminal is configured with these data sets, the release function option bit must be provided in the option field of the CLASS system generation control statement. When this option is specified, the system times the terminal for the amount of time specified at systems generation time for dialing operations; timing is initiated at the receipt of the external interrupt for the ring indicator signal. If no input is received during this timing interval or the remote-id is invalid, the system issues a release function to the terminal which permits a redial from the remote site if desired.

15.7. TIMING CONSIDERATIONS

Within the realm of communications equipment handling, there are three levels of activity which must be considered to determine the communications activity which can be allowed in conjunction with other I/O activity, real time clock activity, and processor usage. These levels are:

- Interrupt response
- Buffer processing
- Information analysis

15.7.1. INTERRUPT RESPONSE

Interrupt response occurs within the communications handler and consists of setting up the next buffer for the interrupting line terminal, queueing the interrupt, and reinitiating the input or output mode. The necessary criteria to be met by the handler are:

(1) Ensuring that the mode be reset within the character availability of the fastest devices on the channel (196 microseconds for 40,800 cps at eight-bit characters)

(2) Ensuring that each interrupt is processed in a time interval such that all active lines could be ready to interrupt at the same time and no information is lost on any line.

If single buffer mode is employed, then the second case is of no concern for the given line (the interrupt terminates activity on that line) but must be considered for lower priority multiple buffer lines. The configuration should be arranged with the highest speed line terminals which may be used in pool mode in the highest priority interrupt position (lowest ESI channel number and highest priority multiplexer position). To ensure no loss of information for each line terminal, higher priority interrupts plus the single interrupt for this line terminal must be handled within the character availability (CA) time of that line.

The count of higher priority interrupts must include:

- one for each ESI external interrupt which can occur in the character availability time
- one for each half-duplex I/O line terminal pair of higher priority
- one for each simplex line of higher priority
- two for each higher priority full-duplex pair
- one for the line of concern.

If any buffer of a higher priority line can fill once or more within the character availability time, one must be added for each occurrence. The character availability divided by 40 microseconds should be greater than the number computed. This value takes into account the interrupt processing instructions plus data transfers. For instance, with the configuration of

- 10 full-duplex UNIVAC 1004 subsystems at 4800 bps (CA = 1.25 milliseconds), and
- 250 half-duplex KSR 35 teletypewriters at 100 words per minute (CA = 20 milliseconds),

the interrupt count for the lowest priority UNIVAC 1004 subsystem is 19 (two for each of the nine higher priority full-duplex pairs plus one for the line of concern), and for the lowest priority teletypewriter is 270 which is less than the limits of 1250/40 = 31 and 20,000/40 = 500, respectively; hence, there would be no loss of information if the entire system was operated simultaneously, in a pool mode (remote UNIVAC 1004 subsystem operation under executive control is normally a single buffer operation). Note that CTS and WTS equipment is operated on ISI mode channels and hence are lower interrupt priority than all ESI channels. In addition, the WTS incorporates character assembly into full words and multiplies the character availability time.

15.7.2. BUFFER PROCESSING

Buffer processing is a user function performed from the input and output completion activities. The most critical buffer processing routines are expected to be those that handle pool mode line terminals. These routines must operate within the constraint that each buffer must be processed within the time it takes to fill the next buffer in the chain (unless the pool is of sufficient length to contain an entire message for each terminal concerned). Each buffer processing routine must share the CPU so that all routines have a chance to process their corresponding buffers. The need to share the CPU among the processing routines dictates that a time interval must be chosen such that switching from one completion activity to another is based on the real time clock. This time interval is set by a system generation parameter (if omitted, no maximum exists). The basis for determining this interval follows.

Each buffer must be processed in the time interval (TD) to fill the largest (in terms of time) buffer (Tmax) divided by the number of buffers which can be filled in the maximum interval. A given buffer can fill Tmax/TI times, where TI is the time required to fill a buffer (number of characters times the character transfer rate). Tmax is the largest value of TI in the system. Hence, the time interval, TD, is determined by the formula:

$$\mathrm{TD} = \frac{\mathrm{Tmax}}{\frac{\mathrm{Tmax}}{\mathrm{T1}} + \frac{\mathrm{Tmax}}{\mathrm{T2}} + \frac{\mathrm{Tmax}}{\mathrm{T3}} + \ldots + \frac{\mathrm{Tmax}}{\mathrm{Tn}}}$$

$$= \frac{1}{\frac{1}{\mathrm{T1}} + \frac{1}{\mathrm{T2}} + \frac{1}{\mathrm{T3}} + \ldots + \frac{1}{\mathrm{Tn}}}$$

$$= \frac{\mathrm{T1\ T2\ T3\ T4 \ldots Tn}}{\mathrm{T1\ T3 \ldots Tn{+}T1\ T2\ T4 \ldots Tn{+} \ldots {+}T2\ T3 \ldots Tn}}$$

The upper and lower limits of the time interval are determined by T1 = T2 = T3 = ... Tn = Tmin and TI = Tmin with all other T1...Tn considerably larger which gives:

$$\frac{\mathrm{Tmin}}{n} \leqslant \mathrm{TD} < \mathrm{Tmin}$$

It is reasonable to assume that all devices are buffered so that the time to fill each is nearly equal; hence, TD = Tmin/n can be used. The value n includes:

- one buffer for each simplex input or output
- one for each half-duplex pair
- two buffers for each full-duplex pair where only multiple buffer mode lines are considered.

The time interval can now be changed as buffers are lengthened or shortened. In the time interval during which the buffers must be processed, the only time which the completion activity does not have control is during cycles taken for data transfers and during time needed to queue interrupts. In the worst case, all active communications lines and all standard lines may interrupt.

Consider the following:

- TD = 10 milliseconds
- 50 active lines at 2400 bps at 7 bits per character
- 1 FH-432 Magnetic Drum Subsystem (240,000 words per second)
- 1 FASTRAND mass storage unit channel (25,150 words per second)
- 1 UNISERVO VIII-C Magnetic Tape Subsystem channel (16,000 words per second)

In the TD (time interval) given above, there could be (240+25.15+16) (1000) (0.01) = 2811 data transfers at 0.75 microseconds per transfer = 2.1 milliseconds. For high speed channels, 50 (10ms) (1.5 microseconds) /2.91 ms = 0.25 ms for communications line terminal data transfers and (53 interrupts) (25 microseconds per interrupt) = 1.33 milliseconds for interrupts, which leaves 10–(2.1+0.25+1.33) = 6.32 milliseconds as the minimum time the program can count on for executing instructions. Since only pool mode processing is critical, the communications handler enters interrupt completion activities into one of two queues controlled by the dispatcher:

(1) Pool mode completion activities

(2) Single buffer mode completion activities

The nature of pool mode processing causes it to be more time critical than single buffer mode processing. As a result, single buffer mode activities are given control only when the pool mode activity queue has been exhausted.

15.8. INFORMATION ANALYSIS

Information analysis is a level of communications activity which is also a user function. This activity is normally expected to be real time with a user-determined response time. This is handled by the executive through the standard dispatching algorithm.

15.9. ERROR CODES FOR LT CONTINGENCIES

Each user program to which communication devices are assigned is expected to have an error handling routine registered with the executive. This error handling routine notifies the user of system and LTG error conditions which do not apply to the transmission or reception of a particular buffer. The user program is notified in the manner and format as defined under program contingency ERR mode condition. A complete list of the possible error codes and their meaning can be found in Appendix C.

16. REAL TIME PROCESSING

16.1. INTRODUCTION

A basic responsibility of the executive is to assist real time programs by providing interfaces that enable real time programs to appropriately influence the executive. Assistance is provided for real time programs in the following areas:

(1) Activity registration

(2) Activity priority manipulation

(3) Program positioning in main storage

(4) Lockout protection from simultaneous record access during program execution

(5) Real time test and set contingency mechanism

(6) Interface with nonstandard peripherals at the hardware level (I/O commands and interrupts).

(7) Priority access to peripheral devices

(8) Communications handler interface to remote devices

The executive design is not oriented toward any one particular type of real time application. Rather, a general environment capable of supporting all types of real time applications is provided.

16.2. PROGRAM LOCATION

The nature of a real time program causes its storage requirements to be handled differently than ordinary demand and batch applications. A real time program is never swapped out of main storage nor relocated within main storage, because it must be accessible at all times. For this reason, the executive system optimally positions all real time programs in main storage, if necessary, when they initially acquire real time status. The use of MCORE$/LCORE$ (see 4.7.1 and 4.7.2, respectively) by real time programs is especially complex due to the "locking in" of the program. A full discussion is beyond the scope of this document and is discussed in other Univac documentation; however, a basic rule may be stated: the program should expand to maximum size before it acquires real time status (see 16.3.2), and afterwards contract to the desired size.

16.3. BUFFER OPERATIONS

One of the most important considerations in a typical real time application is the structure and management of communications buffers. Section 15 provides the details of buffer control interfaces with the communications handler; this section discusses the general aspects of communications buffering in the system environment. Although the system provides both single and pool mode capabilities, the user should employ the method most advantageous for the application. The single mode of operation is the most efficient method, as there is no overhead involved with controls for a pool. Single mode uses less main storage per buffer because pool mode requires additional storage area in the real time program and within the executive for the pool control information. This amounts to three words for each buffer in the pool; one word is maintained with parameter information in the user's area, and two words in the user PCT are used for linking and queuing purposes by the communications handler. The type of transmission is perhaps the best guide as to which should be used.

16.3.1. TRANSMISSION TYPES

Types of transmission may be classified as fixed length, variable length, or indeterminate length. An example of a fixed-length transmission is the use of a UNIVAC 1004 Card Processor as the remote terminal or a similar printing device which employs a print line image of 80 or 132 characters. Also the poll message (not to be confused with a poll response) for a polled network is generally fixed length. The single mode of operation should always be used for fixed-length transmissions.

The maximum length of a variable-length message can be predicted. An example of a variable-length message is a transmission for a CRT operating as a remote inquiry and display device where the remote CRT user may enter any quantity of information up to the maximum size of the CRT screen. If the amount of main storage available for use as buffers for variable-length messages is extremely limited, it is better to use a pool of buffers since not all messages are in progress at once. Less buffer area is needed than if single mode is used and buffers are permanently assigned. Either single mode or pool mode could be used for a variable-length transmission with the most advantageous method being chosen as the application dictates.

Indeterminate-length messages should always be processed using the pool mode of operation. A transmission of indeterminate length is most frequently encountered in message-switching applications where the length of an input message is under the jurisdiction of the remote station, and the system must use segmentation to accept, process, store, and forward the entire message.

16.3.2. MAIN STORAGE AVAILABILITY

Another factor which largely affects whether single or pool mode of buffering is to be used is the quantity of main storage available for buffer areas. The lack of adequate buffering areas dictates that the pool method should be used so that the buffer area may be shared by numerous line terminal (LT) groups. Sufficient buffer area may permit the use of single mode buffering or even the extreme case of a closed pool of buffers for each LT group. However, such an extreme case has an added restraint that each buffer of information must have sufficient staging area and adequate mass storage transfer time so that no data is lost during an overload situation. The open chain pool method ensures that a buffer is never reused by the system until so instructed by the user. If pool mode is chosen, consideration must be given to the size of the main storage area to be used for buffering. If adequate area is available, the desired size can be set aside for the pool.

16.3.3. POOL SIZE

The optimum pool size is determined by the application, but it is also influenced by the system's work load. If the applicattion is such that the loss of any data cannot be tolerated, there is no choice but to fix the size of the pool at some maximum value to adequately handle the peak load. Systems with such stringent requirements do exist, but the real time program has some degree of control over the remote stations. For example, the polling operations can be reduced if an excess work load is encountered, or the remote station can be instructed to retransmit if any portion of a transmission is lost just as though a parity error had occurred.

16.3.4. BUFFER SIZE

Very closely connected with the size of the pool is the consideration to be given to determining the proper size for the individual input and output communications buffer areas (not to be confused with mass storage buffering areas). The size of a communications buffer is normally fixed at a single adequate value and is not dynamically changed or influenced by changes in the system work load or time of day. The individual buffer size is again determined by the application, but it is greatly influenced by such things as the mass storage medium, staging area and working area size, actual line speeds of the communication network, number of circuits in the network, and data packing techniques. With all of these factors taken into account, typical buffer sizes for existing real time installations range from 10 to 100 characters. The mass storage medium is perhaps the largest influence in determining the size for individual buffers. Hardware characteristics such as access time and addressability, as well as software utilization of mass storage in areas such as the amount and method of segment and message linkage, real time directory contents and location, real time repacking principles, and so forth, must be considered. A mass storage device with a high access time dictates that a larger buffer size should be used, while smaller buffers may be successfully used if the mass storage device has a very fast access time and also a high data transfer rate.

Addressability of the mass storage medium determines staging area size which, in turn, controls the size of individual buffers. It is best to establish a buffer which is an integral fraction of the size of the staging area. For example, a mass storage device with either track or sector addressability would need the staging area to be a multiple of the track or sector size. The full range of software techniques for proper use of mass storage is beyond the scope of this manual. It is expected that the size of individual buffers must increase with the line speed (a larger buffer size is more successful for a 4800-baud line than the buffer size chosen for a 75-baud line). The size of buffers should also increase with the number of lines controlled by the system. Note the use of the term *system;* it is possible that several real time programs may operate concurrently. The work load would be divided among various programs, according to type of work, with common linkage between programs existing only in the form of queues, tables, files, and so forth. Multiple real time programs are not time shared or sliced by the executive but are expected to voluntarily share the system resources according to their combined requirements.

Data packing techniques also influence individual buffer size, and cover both hardware characteristics and software methods. Hardware characteristics include such things as whole-word, half-word, or quarter-word communications buffering methods, partial-word addressing capabilities of the CPU, and peripheral subsystem operation such as the reading of bytes by way of the UNISERVO VI-C/VIII-C Magnetic Tape Subsystems, which pack nine-bit bytes in double-word format.

Software methods include use of a common internal code and related packing. Some internal code must be selected as the standard to be used internally by the real time program. The code which is most common throughout the communications network is generally selected although the decision may be influenced by line speed to avoid excessive character translation on high speed lines. This selection is generally not Fieldata (which is used internally in the 1100 series) but either Baudot or ASCII, since the majority of remote stations have hardware design characteristics employing one of these codes.

Any input which is received and which does not conform to the standard internal code is translated before being processed by the real time program. For output, the data in internal code must be translated to the code desired by the remote station as the communications output buffer area is being filled.

Once a particular code is selected as the internal standard for the real time program, various data packing methods by the software can be built around that code. For instance, the selection of ASCII permits only nine characters to be packed into a double word, while the use of Baudot allows 10 characters to be packed into a double word with the added advantage of two bits remaining for use as control information. The proper control techniques and the greater efficiency of packing using a smaller code such as Baudot can provide a larger system capacity when mass storage area is limited. The use of vertical packing rather than horizontal packing can eliminate the output staging area since the communications output buffer area can double as the staging area. Vertical packing may also be advantageous for certain unique hardware characteristics.

16.3.5. DUAL POOL METHOD

The dual pool method is available for accepting input. The primary use for the dual pool input mode is to provide a rapid software response for features otherwise provided by hardware options for polling operations. The response from a poll message may be either a short response or a lengthy transmission of data from that particular remote station. It is most desirable for the real time program to have immediate notification when the first portion of the poll response is received so that the next poll message may be initiated if the response indicates NO BUSINESS. Rather than use a timer, the occurrence of an input monitor interrupt for a small buffer can be used to trigger the real time program's analysis of the poll response. However, if a lengthy transmission was initiated by the poll message, it is not advisable from a system standpoint to continue with small buffers for accepting input data. The dual pool input mode provides the ability to accept a transmission into a small buffer area when the input request (CMI$ request see 15.4.1.3) is initiated, with an immediate switch to a pool of larger buffers for subsequent portions of the data transmission, if any should follow.

16.4. PROGRAM EXECUTION CONSIDERATIONS

The following discussions cover the areas of priorities, priority control, timing constraints, and the use of the Test And Set (TS) instruction to protect common data areas.

16.4.1. PRIORITY CATEGORIES

16.4.1.1. I/O PRIORITY

The executive assigns an I/O request into one of three categories, depending upon the nature of the activity which submits the request.

(1) Real Time

(2) Executive

(3) Demand/Batch

For each subsystem, all requests in a priority category are completed before any request for the next lower priority is honored.

Look-ahead techniques are used within a priority category, when appropriate, so that average time for I/O operations may be reduced. Look-ahead techniques may cause I/O requests to be completed in a sequence different from the order of submission; therefore the program must provide its own protective measures and take the appropriate action if it is concerned with the completion sequence of I/O requests.

16.4.1.2. DISPATCHING PRIORITY

The executive provides several methods to be used by real time programs for changing CPU priority (switching) levels, registering activities, and dispersing the work load among the processors in a multiprocessor environment. The proper use of these executive functions should be understood so that they are not abused, but are properly utilized with minimum system overhead to achieve a desired goal. This discussion is concerned with the priority control executive requests RT$, NRT$, FORK$, EXIT$, and UNLCK$, as well as the priority levels imposed on all activities by the executive. The numerous priority levels should be understood in order to determine the proper use of the previously mentioned executive requests. Deadline and demand activities have a priority below that of real time and are considered to be grouped with batch for the purposes of this discussion. The main categories of CPU priorities have the following order:

Category	Description
A	All interrupt queueing at the time of interrupt occurence.
B	Any ESI completion activity in order of interrupt occurrence.
C	All high priority executive action.
D	Any real time interrupt activity in order of occurrence (level 1)
E	Real time activity levels 2 through 35.
F	All low priority executive actions.
G	All batch activities.

The occurrence of any work in a higher priority category causes the lower category to be suspended until all higher priority work has been completed. All categories except A are controlled by the software, and therefore, are interruptable and subject to suspension.

- Category A

  Interrupt queueing is the process performed at the occurrence of a hardware interrupt and requires approximately 45 microseconds of instruction execution time for an ESI interrupt as discussed further in 16.6.2.1.

- Category B

  Category B has the highest software imposed priority and is used for the processing of ESI completion activities.

  In order to detect program errors and excessive loops, the executive always times ESI completion activities. The amount of time permitted for these activities is fixed at system generation time.

- Category C

  Used for all high priority operations that the executive must perform, such as I/O control, interrupt post processing, dispatching, and clock control.

- Category D

  Used for real time I/O interrupt activities at level 1 (see Section 6). This category is always activated with a limited register set (X11,A0-A5,R1-R3).

- Category E

  Used for real time priority levels 2 (highest) through 35 (lowest). All activities at a particular real time level are serviced before any service is given to lower level activities. There is a maximum of 34 real time levels available for use by all real time programs. The user is expected to assign the real time levels appropriately for the application. Either a limited set or a full set of registers may be used for real time levels 2 through 35.

- Category F

  Used for low priority executive processing such as storage management, function control, symbiont processing, and so forth.

- Category G

  Batch activities are normally subject to suspension by the executive for purposes of swapping. Batch activities of a program having real-time status (see 16.4.2), however, are never suspended for swapping. This is the only preference given to batch activities of a real time program; all in-main-storage programs are treated equally as far as batch activity switching is concerned. This means that background batch programs which do a lot of computation might delay the completion of batch activity execution within a real time program, especially if such activities also perform extensive computation (I/O-oriented batch activities are treated preferentially – see 25.5.6.2 for a full discussion of the switching algorithm).

  The implication of the preceding discussion is that care must be taken in assigning time-critical work to batch activities within a real time program.

16.4.2. PRIORITY CONTROL

The following executive requests can be used by the real time program to control the priorities and to distribute the work load in a multiprocessor environment:

ER RT$ (see 4.3.4.1)

ER NRT$ (see 4.3.4.2)

ER FORK$ (see 4.3.1.1)

ER UNLCK$ (see 6.3.8)

ER EXIT$ (see 4.3.2.1)

16.4.2.1. CHANGING ACTIVITY PRIORITIES (RT$ AND NRT$)

The RT$ request (see 4.3.4.1) is provided for changing priority levels within the real time program. It can accomplish either of the following:

- change of priority level for a real time activity; or

- allow a program to acquire real time status.

The RT$ request may be executed by any activity: batch, demand, or real time at levels 2 through 35. A check is performed by the executive to determine if the run's account number permits real time activities and the real time level specified is allowed for the account number.

Closely associated with the RT$ request is the NRT$ request (see 4.3.4.2) which provides the ability for a real time activity to return to its original status (that is, batch or demand). The level for a nonreal time activity is determined by the executive and cannot be specified by the user. A program has real time status whenever it has one or more real time activities.

16.4.2.2. APPLICATION OF MULTIPROGRAMMING TO REAL TIME

Although the multiprogramming concept (use of multiple activities) is applicable to many types of processing, certain aspects are particularly pertinent to real time programs. Some important real time applications of multiple activities are:

- Distribution of the work load among several CPUs (multiprocessing).

- Use of real time interrupt activities to ensure top priority for turnaround of critical I/O operations.

- Registering multiple activities at varying real time priority levels to perform tasks of varying criticality. (See FORK$ and TFORK$ requests – 4.3.1.1 and 4.3.1.2, respectively.)

- Registering batch activities to perform noncritical tasks in the background.

While the above techniques are useful and powerful, the real time programmer must take care not to overuse the activity concept. Important points to consider include:

- Dispatching overhead. The time required to switch from one activity to another is not negligible and can become excessive if the work load is distributed among too many activities. In many cases, a single activity can perform several tasks in serial fashion more efficiently than several activities performing single tasks in parallel (the RT$ request – see 4.3.4.1 – can be used to adjust activity priority according to the priority of the task to be done).

- Forking overhead. Each new activity requires both time and space for registration. Time is also required for activity termination. Again, a single activity may do as well as several in certain cases. Note that activity control space is taken from the PCT and thus unlimited forking is not possible.

- Register set. Extra time and space are required for saving and restoring the major register set. Therefore, heavily executed activities should use the minor set if feasible. Note that an activity cannot change its register set; however a FORK$ request followed by a EXIT$ request has the effect of changing set (note that activity name and id are not retained in this sequence).

16.4.2.3. INTERRUPT ACTIVITY PRIORITY REDUCTION (UNLCK$)

An interrupt activity is initiated with a limited register set at a priority level above all other activities for that type of program. If no other executive request is to be used by this activity, a UNLCK$ request should be added to return the activity to the real time priority level of the activity that issued the I/O request (see 6.3.8).

16.4.2.4. ACTIVITY TERMINATION (EXIT$)

Normally the EXIT$ request (see 4.3.2.1) causes termination and deletion of an activity. The EXIT$ request works differently, however, when executed by an activity that was originally initiated as an ESI completion activity. Rather than the activity being deleted, it is placed in a wait condition so that it is again eligible for execution as an ESI completion activity when a communications interrupt occurs. When the activity is in a wait condition, it can be given control almost immediately since the area used to hold information related to the activity is permanently maintained for the ESI completion activity.

16.4.2.5. TIMED WAIT CONSIDERATIONS

The real time programmer using timed wait requests (TFORK$ – see 4.3.2.1; TWAIT$ – see 4.3.5) should understand that the wait time for real time acitivities is an elapsed time by the clock, during which the activity is suspended. When the specified time has elapsed, the activity becomes available for execution at its priority level; this means that execution does not actually resume until there are no higher (or possibly equal) priority activities requiring CPU time.

It should also be noted that timed waits do require time to process. A large number of activities in short wait loops may effectively lock out lower priority activities.

16.4.2.6. CONSOLE INTERRUPT HANDLING

Real time programs which accept unsolicited console input usually benefit from using II$ requests (see 4.6.2) rather than the II contingency (see 4.9). The reason is that the II$ request allows a particular activity to get control at a priority which can be programmer specified, whereas, for II contingency, the executive may select any activity to process the interrupt (this might be a low-priority real time or a batch activity which could be delayed for an undesirable period by high priority execution). In addition, the II$ request receives some input, which may eliminate the need for a type and read sequence to find out what the operator wants.

16.4.3. EXCEEDING MAXIMUM TIME LIMITATION

CPU time used by real time activities is charged to the run. To reduce overhead, the check for exceeding maximum run time (from the @RUN control statement) is not made for real time activities. However, the check is made for any batch activities in a real time program, and the entire program is aborted if the limit is exceeded.

This means that when debugging a real time program, a loop is not caught by the maximum time check unless batch activities are executed. Conversely, a fully operational real time program which uses batch activities may be mistakenly aborted during a long run unless a sufficiently large maximum is specified on the @RUN control statement (see 3.4.1) or unless system standard is not set to abort a run upon exceeding maximum time.

16.4.4. TEST AND SET USAGE

The Test And Set (TS) instruction is available in all systems (simulated by way of software if not part of the hardware) to protect common data in a user program. For batch and demand activities, the executive automatically resolves conflicts that may occur in a user program because of activities at different switching priority levels for these types of activities and automatically degrades by one level any activity that experiences a TS failure (TS interrupt). After level degradation, the executive forces the activity to the end of the list of activities operating at that level.

For real time programs, the executive does not change the activity switching level on the TS interrupt, except for interrupt activities which are dropped to the level of the activity that issued the I/O request, (this means that for TS purposes, interrupt activities have an implied level equal to that of the issuing activity). Instead, the user activity is directed to a contingency routine if one is registered (see 4.9). If a contingency routine is not registered, the activity is placed at the end of the list of activities operating at the activity's current level. It is obvious that the real time program can hang the system if the situation does not automatically resolve itself at the original switching levels, or if the proper action is not taken in the contingency routine to change the switching level in relation to other activities that reference the data. The contingency routine enables the activity to change its level (by an RT$) if a potential lockup exists, or to perform lower priority work for some period of time. Note that a contingency notification on a TS interrupt does not occur unless the activity specifically requests such action, independent of other types of contingencies.

A real time program may have nonreal time activities. For activities other than real time, the TS contingency feature is not available. The real time program may be delayed indefinitely if activities both real time and nonreal time attempt to reference common data. The nonreal time activity should cause itself to be raised to real time status before executing a TS protecting the common data; the activity can revert to its original status after the need for the common data reference has ended.

A real time activity should never have a TS cell set when calling an ER. Although this procedure may work in some cases, it is highly vulnerable to lockup (since ERs commonly require executive processing that runs at a CPU priority level below real time), and is strongly discouraged. Furthermore, this practice violates a basic system philosophy concerning TS usage, namely: that TS's are to be used to protect short critical sequences and not as a long term logical interlock.

It is illegal for ESI activities to reference common data that is also referenced under a TS condition by real time activities. No action, other than a return of control, is taken by the executive when a TS interrupt occurs in an ESI activity.

Real time programmers working on unit-processor 1108's which lack TS hardware should be aware of the extra time (about 35 microseconds) required to software simulate TS (see 2.4.5).

16.5. PROGRAMMER'S GENERAL RESPONSIBILITIES

The programmer, prior to interfacing with the communications handler must:

(1) Use options on the @RUN control statement to assure sufficient PCT size (see 3.4.1). (All control packets are built and maintained by the handler in the user's PCT). Once the program becomes real time, it is locked in main storage and incapable of being moved for dynamic expansion of the PCT. Note that PCT space is also required for activities created by a FORK$ request (see 4.3.1.1).

(2) Assign the line terminal group (LTG) to be controlled by an @ASG control statement (see 3.7.1) or dynamically using a CSF$ request (see 4.8.1). The assignment must be made using the arbitrary device format.

(3) Declare the program as real time using an RT$ request (see 4.3.4.1).

(4) Prepare the LTT (see 15.3) for each LTG, and the input and output buffers to be used for the data transfers.

The user also should register, by an IALL$ request (see 4.9.3.1), a program contingency routine as well as an ESI contingency routine. By doing this, the user can obtain error information should an error occur during his interface with the communications handler.

The communications handler/user interface, therefore, should include:

- One or more real time activities which request, by means of the LTT and ER's, the communications handler to perform input and output.

- ESI completion activities which are given control by the communications handler when the data transfer is accomplished.

- A program contingency routine which is given control by the communications handler when invalid requests are made to the communications handler.

- An ESI contingency routine which is given control by the communications handler when errors are detected during the execution of an ESI completion activity.

16.6. ESI CONSIDERATIONS

16.6.1. ESI ACTIVITY CONCEPT

An ESI activity is generated when the real time activity makes a reference to CMS$ (see 15.4.1.1). A nonreal time activity is not permitted to make this reference. At this point in time, the executive establishes an entry point at which control is given upon occurrence of an interrupt for the line associated with the LT group. The ESI activity is never suspended, either voluntarily or involuntarily, and hence, no control register or CPU state preservation is required for ESI activities. An ESI activity is given top priority by the executive such that it is never interrupted except for interrupt queuing. Executive operation can be interrupted at any time to allow ESI processing. The philosophy adopted for ESI activities is that of extending system interrupt processing to the user. For this reason, the amount of work done by an ESI activity is restricted to a minimum, allowing a nonESI activity to do the remaining work. This implies certain restrictions on programs using the communications handler:

(1) Only real time activities can establish ESI activities.

(2) ESI activities are limited to four executive requests (EXIT$ – see 4.3.2.1; ACT$ – see 4.3.3.4; CADD$ – see 15.4.2.3; and ADACT$ – see 15.5.1) and may only use one request per activation. Violation causes an ESI contingency.

(3) ESI activities must be completed within a limited time frame defined at system generation.

(4) TS protected common data can exist among ESI activities but not between ESI and real time activities. In other words, a real time activity that blocks the path of an ESI activity is in error.

Since the ESI activity is not allowed forking requests, any real time activity which is to process incoming data must already have been established with a 'name' (see 4.3.3.2). This name is then used by the ESI activity to initiate processing. When processing is complete, the real time activity must execute a DACT$ request (see 4.3.3.3). DACT$ differs from EXIT$ in that the activity is not terminated but rather put into an inactive state.

16.6.2. ESI TIMING

ESI interrupt processing and switching times under various conditions have been calculated for real time programs by counting the actual instructions involved. These counts must be considered as estimates because coding is subject to modification.

16.6.2.1. ESI INTERRUPTS

When an ESI interrupt occurs, various functions must be accomplished before interrupts can be enabled. The amount of time required before interrupts are allowed depends upon the type of interrupt (input monitor, output monitor or external) as well as the type of operation (single or pool mode). A maximum of 45 instructions are required before interrupts are enabled.

(1) If an ESI activity has been interrupted, control is returned to the activity within the 45-instruction limit.

(2) If control is to be given to a new ESI activity, approximately 120-130 instructions are required between the time the ESI interrupt occurred and the new activity receives control. The time required depends upon the number of control registers to be saved if an activity has been interrupted as well as the amount of postprocessing required for the activity in the communications handler.

16.6.2.2. REAL TIME ACTIVITIES

If a real time activity has been interrupted to process an interrupt, approximately 230 instructions are executed between the time the interrupt occurs and the real time activity receives control (the actual time depends upon the type of interrupt being processed). If a new I/O function is available for initiation before the real time activity receives control, approximately 800 instruction executions are required between interrupt occurrence and return of control to the activity.

17. CHECKPOINT/RESTART

17.1. INTRODUCTION

Checkpoint/restart provides the user with the ability to save the status of a run at a given point (checkpoint), and to restore the run to the previous state at some future time (restart).

There are two basic types of checkpoints: complete and partial.

The complete checkpoint saves the complete status of a run at a given point; a restart of this checkpoint acts like a start of a new run. The partial checkpoint saves only the status of a program at a given point; it saves only the user's program and activities. A partial restart initializes and executes the checkpointed program in the run requesting the restart. The partial checkpoint and restart is actually a program save and restore.

17.2. COMPLETE CHECKPOINT/RESTART

The checkpoint and restart operations may be initiated by three kinds of requests:

(1) an executive request;

(2) a control statement; or

(3) an unsolicited console keyin.

Any checkpoint dump is capable of being reestablished by any restart request regardless of the kind of checkpoint request used to generate the dump. For example, an internal (program) checkpoint request may be restarted by either an internal request, a control statement, or unsolicited console request. The dump may be recorded on magnetic tape or mass storage files.

The complete checkpoint/restart feature cannot be used when operating in the demand mode.

17.2.1. COMPLETE CHECKPOINT – RUN SAVE

When a checkpoint request is encountered, the checkpoint routine suspends any current activities and saves all the pertinent data needed to restart the run. Checkpoints are not taken on runs with active ESI activities, from a keyin when a program within the run has acquired real time status, or when reentrant processors are active for the run.

The checkpoint routine automatically provides all the information necessary for restart except for currently catalogued or to be catalogued mass storage files. Checkpoint copies all temporary mass storage files, but it copies only those catalogued files on which the B option was specified on the @ASG control statement (see 3.7.1). If a file is dumped, the status of the original file remains unchanged.

Some of the operations involved when a checkpoint is taken are as follows:

- A checkpoint sentinel block is written.
- The user main storage area is saved.
- The program's registers are saved.
- Executive control tables, switch lists, and so forth are saved.
- The remaining portion of the run stream is saved. The current position within @ADD files (see 3.9.1) is also saved. These @ADD files must remain catalogued until the restart or be assigned with a B option at checkpoint time.
- All temporary mass storage files are saved. If a temporary file does not need to be saved, it should be released before the checkpoint using a CSF$ request (see 4.8.1).
- Catalogued mass storage files are saved if the B option was specified on the @ASG control statement. This also applies to files that are about to be catalogued.

17.2.1.1. CONTROL STATEMENT (@CKPT)

Format:

```
@label;CKPT,options    filename
```

All parameters on the @CKPT control statement are required except label and options.

Parameters:

options	The options are: P – Displays completion and error messages on the operator's console; these messages are also entered in the master log. T – Terminates run after the checkpoint is taken.
filename	Specifies the assigned magnetic tape or FASTRAND–formatted file into which the checkpoint information is to be dumped. The file which is to contain the checkpoint dump must be catalogued, public file.

17.2.1.2. EXECUTIVE REQUEST

The CSF$ request (see 4.8.1) is used to request a checkpoint from within a user program. The user packet contains a character string identical in format to the @CKPT control statement (see 17.2.1.1).

17.2.1.3. UNSOLICITED CONSOLE REQUEST

Format:

```
CK,options    filename,run-id
```

Options is the only optional parameter.

Parameters:

options — Same as for the @CKPT control statement (see 17.2.1.1).

filename — Specifies a magnetic tape file which is assigned to the executive by checkpoint and catalogued so that it is available for restart.

run–id — Specifies the run–id of the run to be checkpointed.

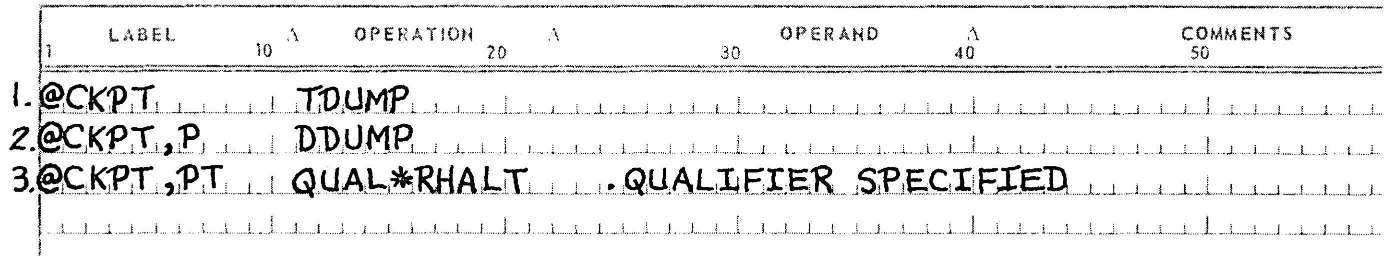
Description:

The unsolicited keyins can be used to checkpoint any active batch run not having real time or ESI activities. The checkpoint must be taken on a magnetic tape file which will be assigned to the executive by checkpoint and catalogued so that it is available for a restart. The operator should ensure the availability of a tape unit prior to initiating an unsolicited keyin. Unsolicited keyins are not permitted for mass storage files or for any file assigned to the user. If successive checkpoint requests are keyed in for checkpoints on the same file, the tape remains assigned until all checkpoints are taken.

When no additional checkpoint requests are queued for a particular file, the message SHOULD 'filename' BE RELEASED: ANS Y OR N is displayed on the operator's console. If the operator expects to take further checkpoints on this file, an N should be keyed in; if not, a Y response is required.

NOTE: The Y response is essential following the final checkpoint on a file in order to free the tape unit and catalogue the file.

17.1.4. EXAMPLES OF CHECKPOINT



```
   LABEL     ∧  OPERATION    ∧          OPERAND      ∧               COMMENTS
  1         10            20          30            40               50
1.@CKPT         TDUMP
2.@CKPT,P       DDUMP
3.@CKPT,PT      QUAL*RHALT      .QUALIFIER SPECIFIED
```

1. TDUMP is specified as the file into which the checkpoint information is to be recorded. TDUMP must be an assigned file, either magnetic tape or FASTRAND–formatted mass storage. Since no options are specified, messages are not displayed and run termination does not occur when the checkpoint is taken.

2. Basically the same as for example 1, except that the P option specifies that error and completion messages are to be displayed when the checkpoint is taken.

3. This example includes the features included in both example 1 and example 2. In addition, the T option is used to terminate the run when the checkpoint is taken.

17.2.2. CHECKPOINT FILE FORMAT

The technique by which the checkpoint routines perform a dump varies with the media on which the checkpoint is to be recorded (magnetic tape or mass storage). A checkpoint on mass storage is limited to one active dump per file in order not to strain storage capacity; only the last checkpoint taken is available as a restart point. Therefore checkpoints on mass storage should be called primarily for those runs where hardware malfunctions are the expected contingencies and where an immediate restart of the run is desirable.

When checkpoint information is recorded on magnetic tape, it may be interspersed with data on a data output tape or may be recorded separately on a nondata tape. In either case, the tape may contain many checkpoints, of which any of them can be used as a restart point for the run.

The checkpoint routine writes two end-of-file (EOF) marks on the tape at the end of each checkpoint, and backs up one EOF mark. Thus checkpoints stacked on a tape are separated by an EOF mark; the last checkpoint is followed by two EOF marks.

The checkpoint can be bypassed on a data tape by skipping the initial and final bypass blocks, one EOF mark, and all tape blocks in between. The 18—word bypass blocks take the form:

S1	S2	S3	S4	S5	S6
74	75	74	75	74	75
40_8					
40_8					
74	75	74	75	74	75

If an end-of-reel condition is encountered on the checkpoint tape while a dump is being recorded, the routine automatically writes an end-of-reel block and two hardware EOF marks, swaps reels, writes a sentinel block and then continues dumping on the alternate reel. The format of the 18—word end-of-reel block is as follows:

S1	S2	S3	S4	S5	S6
74	75	74	75	74	75
20_8					
20_8					
74	75	74	75	74	75

Control is not returned to the user until the checkpoint is complete. Because of this, a data file tape should not be designated as the checkpoint file if the user must handle end-of-reel processing at the end of every reel.

17.2.3. CHECKPOINT FILE IDENTIFICATION MESSAGE

The checkpoint routine produces a message to indicate that a checkpoint was completed successfully. This message is recorded in the master log. If the P option is used in the checkpoint request, however, it also appears on the operator's console. The format of the message is

```
     run-id      ckpt-nbr      filename,reel-nbr
```

where:

run-id Identity of the run.

ckpt-nbr Number of the checkpoint which must be specified in the restart request.

filename Specifies the file that contains the checkpoint dump.

reel-nbr Number of the reel or reels of the file that contains the checkpoint if it is on tape.

In addition, a list of all files assigned to the run at checkpoint time is placed in the master log along with the normal completion message just discussed. This list takes the forms

Form 1:

```
filename-1 - reel-nbr-1, *reel-nbr-2,...,*reel-nbr-n

filename-2 - reel-nbr-1, *reel-nbr-2,...,*reel-nbr-n
     .                                      .
filename-n - reel-nbr-1, *reel-nbr-2,...,*reel-nbr-n
```

Form 2:

```
filename-1 - equip-type

filename-n - equip-type
```

where:

filename Specifies the files currently assigned to the run.

reel-nbr Reel numbers presently assigned. An asterisk before the reel-nbr indicates the reel that was actually in use at the time of the checkpoint dump.

equip-type See Appendix E.

If the checkpoint dump must be error terminated, a Fieldata status code is generated (see Table 17–1).

17.2.4. COMPLETE RESTART – RUN RESTORE

When a restart request is received, the restart routine schedules a new run with the information specified on the restart request. When the run to be opened is selected for execution, the restart routine assigns the checkpoint file, finds the checkpoint, and restores the run to its original state at the time of the checkpoint. Some of the tasks performed by the restart are:

- The program's registers and tables are restored.
- The user's main storage is restored.
- All facilities are reassigned at restart. Repositioning of arbitrary devices, however, is not done by restart but must be handled by a restart contingency routine.

- All tape files are repositioned.
- All temporary mass storage files are restored.
- Those catalogued mass storage files which were saved because of the B option on the @ASG control statement (see 3.7.1) are reloaded according to the other options (E, H, or M) specified on the @ASG control statement at checkpoint time.

17.2.4.1. CONTROL STATEMENT (@RSTRT)

Format:

```
@label:RSTRT,priority/options   run-id,acct-id,filename,ckpt-nbr,reel-nbr
```

All parameters in the @RSTRT control statement are optional except run-id, filename, and ckpt-nbr.

Parameters:

priority — Specifies the priority under which the run is to be reestablished and restarted (see 3.4.1). If omitted, the priority of the initiating run is used.

options — The options are:

P — Error messages are displayed on the operator's console as well as in the master log. If omitted, the error messages appear only in the master log.

T — Terminates the run requesting the restart.

run-id — Specifies the run to be restarted.

acct-id — Specifies the account to which charges are made for reloading the run. Once the reestablished run receives control, charges revert to the original account for the run. If omitted, charges are made to the account specified for the initiating run.

filename — Specifies the catalogued file containing the dump. Entry must agree with that specified in the filename parameter of the appropriate @CKPT control statement (see 17.2.1.1).

ckpt-nbr — Specifies the checkpoint number at which the run is to be restarted. This is the last checkpoint taken for a dump recorded on mass storage.

reel-nbr — Identifies the specific reel of the file containing the dump. If omitted, each reel of the file, beginning with the first reel, is searched. All the information necessary to read the checkpoint file is taken from the master file directory.

17.2.4.2. EXECUTIVE REQUEST

The CSF$ request (see 4.8.1) is used to restart a run from within a user program. The user packet contains a character string identical in format to the @RSTRT control statement (see 17.2.4.1).

17.2.4.3. UNSOLICITED CONSOLE REQUEST

Format:

```
RS,priority/options   run–id,acct–id,filename,ckpt–nbr,reel–nbr
```

Parameters:

Identical to @RSTRT control statement (see 17.2.4.1).

Description:

The checkpoint being restarted by a keyin may have been requested by either an unsolicited keyin, a @CKPT control statement, or an internal program request. Restarting a run, which was checkpointed, by using an unsolicited keyin results in the checkpoint file being assigned to the executive. At the end of a restart involving such a file, the operator is given the option of releasing the file by means of a Y response to the message SHOULD 'filename' BE RELEASED? ANS Y OR N. If the file is not released at this time, it remains assigned to the executive until another restart of a run which is contained on this file is attempted and the operator is again given the opportunity to release the file.

When a run checkpointed by a control statement or an executive request is restarted by an unsolicited keyin, the checkpoint tape is assigned to the restarted run.

17.2.4.4. EXAMPLES OF RESTART

```
     LABEL           OPERATION                 OPERAND              COMMENTS
1         10              20             30             40             50
1. @RSTRT,C/P     DRUM,03212,DDUMP,4          . FASTRAND FILE
2. @RSTRT         TAPE,,TDUMP,8,N432          . TAPE FILE
```

1. C priority is assigned to the reestablished run identified as DRUM. Charges for loading the run are made to account 03212. The checkpoint at which the restart is to begin is checkpoint 4 on the catalogued file DDUMP. Since this file is not on tape, the reel-nbr parameter is not specified. Error messages, if they occur, are displayed on the operator's console (P options).

2. The run identified as TAPE is to be reloaded and restarted at checkpoint 8 of the file TDUMP. The dump for this file is located on tape reel N432. Since no priority is specified, the restarted run uses the priority of the initiating run. Error messages which might occur are recorded only in the master log and are not displayed on the operator's console since the P option is omitted. Charges for loading the run are made to the account of the initiating run.

17.2.5. RESTART CONTINGENCY ROUTINE

A restart contingency routine may be specified by the user before a program requests either a complete or partial checkpoint. This routine will receive control upon completion of the load, when the run is activated at restart time, and may do whatever is necessary before giving control to the normal return. The request is:

```
           L      A0,ADRS
           ER     IALL$
           .
           .
           .
      ADRS +      04000,PKT
       PKT+       0
           +
     START .                              .Start of contingency routine
           .
```

When the contingency routine receives control, the packet (pkt) is set up as follows:

	S3	S4	S5	S6
	contingency-type (6_8)	return-addr		
			ckpt-type	ckpt-nbr

where:

contingency-type Specifies the type of contingency (6_8).

return-addr Specifies address to which control is returned upon completion of the contingency routine.

ckpt-type Specifies what initiated the checkpoint; where:

0_8 — Initiated by an executive request

1_8 — Initiated by an unsolicited keyin

ckpt-nbr Specifies checkpoint number and type; where:

0_8 — Partial checkpoint

1_8 thru 77_8 — Complete checkpoint

After the contingency routine is completed, the user should return control to the address placed in the packet by the restart routine. This address is either the instruction immediately following the point at which the internal checkpoint was taken or the return address for an externally caused checkpoint.

17.3. PARTIAL CHECKPOINT/RESTART

A partial checkpoint may be initiated by either an internal program request or by a control statement. The same is true of the partial restart feature. While useful in the batch mode, this program save and restore feature is most useful in saving and later executing a program while operating in demand mode.

17.3.1. PARTIAL CHECKPOINT – PROGRAM SAVE (@CKPAR)

Purpose:

The partial checkpoint is requested when only the current portion of the user's program is desired for checkpoint. This checkpoint saves only user's program, activities, and registers. The checkpointed program is recorded and named in a user program file in absolute format.

The checkpoint is initiated by the @CKPAR control statement.

All parameters in the @CKPAR control statement are required except label and options.

Format:

```
@label:CKPAR,options    filename.element
```

Parameters:

options — Same as for @CKPT control statement (see 17.2.1.1).

filename — Specifies the name of a program file currently assigned to the run.

element — Specifies the name to be given to the element created by the checkpoint. Only one checkpoint per element name is permitted.

Description:

The @CKPAR control statement has no meaning within a batch run stream because there is no program executing at the time the statement is encountered.

A partial checkpoint can also be initiated from within a user program by a CSF$ request (see 4.8.1). The user packet contains a character string identical in format to the @CKPAR control statement.

17.3.2. PARTIAL RESTART – PROGRAM RESTORE (@RSPAR)

Purpose:

A partial checkpoint may be initiated by either an internal program request or by a control statement. It is used to restart a program whose checkpoint was taken by a partial checkpoint control statement or executive request (see 17.3.1). The checkpoint may have been taken within the same run or earlier in another run.

The partial restart is initiated by the @RSPAR control statement.

All parameters in the @RSPAR control statement are required except label and options.

Format:

```
@label:RSPAR,options   filename.element
```

Parameters:

options — The only option is the P option. When specified, error messages are displayed on the operator's console in addition to being printed in the master log.

filename — Specifies the name of the assigned program file containing the checkpoint.

element — Specifies the name of the element which contains the checkpoint.

Description:

When an internal partial restart request is received, the present program and activities are terminated and the checkpointed program is executed as the next operation in the run requesting the restart. When the request is submitted in the run stream, the same is true with the exception that no program is currently in execution, thus there is no termination. In either case, no files are assigned, tapes are not positioned, and any facility manipulation which might be needed must be accomplished using a restart contingency routine (see 17.2.5).

A partial restart can also be initiated from within a user program by a CSF$ request (see 4.8.1). The user packet contains a character string identical in format to the @RSPAR control statement.

17.4. CHECKPOINT/RESTART ERROR CODES

Tables C–6 and C–7 (see Appendix C) list the checkpoint and restart error codes which are returned when a complete or partial checkpoint or restart is error terminated. The requesting run is not terminated as a result of the inability to take a checkpoint. The error code is stored in register A0 on the return from a CSF$ request, and is listed in the master log. If the P option was specified, an error message is displayed on the operator's console.

18. SYSTEM SYMBOLIC PROCESSORS

18.1. INTRODUCTION

This section describes various system symbolic processors that are available to the user. All of the processors can be called from within a user program by coding the appropriate control statement.

The processors discussed are:

- CON78 – Accepts CUR-formatted symbolic elements and converts them to FURPUR-formatted symbolic elements. (See 18.7.)
- CULL – Generates a listing of symbols cross-referenced to the element and line in which they are found. (See 18.5.)
- DATA – Inserts, updates, and corrects data files from within the run stream. (See 18.3.)
- ED – Permits conversational editing of symbolic files and elements. (See 18.4.)
- ELT – Inserts and updates symbolic elements in a program file from within the run stream. (See 18.2.)
- LIST – Generates an edited listing of any type of element. (See 18.6.)

In addition to the system symbolic processors, the @END control statement (see 18.2.1), which operates in conjunction with the DATA and ELT processors, is also described.

18.2. ELT PROCESSOR

Purpose:

Introduces an element into a particular program file or makes corrections to a symbolic element in a program file from the run stream. The @ELT control statement is used to call the ELT processor and it must precede the element or correction images in the run stream. With the exception of the @ELT,D control statement, the ELT processor is terminated by the first control statement encountered in the run stream. ELT processor operation is terminated by an @END control statement (see 18.2.1) whose sentinel matches the sentinel on the @ELT control statement.

All parameters in the @ELT control statement are optional except @, ELT, and eltname-1.

Format

@label:ELT,options eltname-1,eltname-2,sentinel

Parameters:

options

See Table 18-1.

The A, R, and S options (element type options) identify the element type, while the I, L, and U options principally outline element (image handling) options. Those elements identified as type S are considered symbolic elements and may be corrected by using the processor control statements (see 9.4). The format of symbolic elements in a program file is that of a SDF-formatted file (see 24.2.3).

The S option is assumed when the element type options are not specified. The L option is assumed when the element handling options are omitted.

eltname-1

Specifies the input element

eltname-2

Specifies the new output element generated

sentinel

Specifies, when the @ELT,D control statement is used, the character code which terminates the flow of data images into the element being created. This parameter may consist of one to six characters and must agree exactly with the sentinel code appearing on the @END control statement (see 18.2.1) used to terminate the ELT processor.

Option Character*	Description
Element Type Options	
A	Identifies element as an absolute element; used only with the I option.
R	Identifies element as a relocatable element; used only with the I option.
S	Identifies element as a symbolic element. This option is assumed when no element type option is specified.
Image Handling Options	
D	Indicates that the symbolic input images following the @ELT control statement may include control statements which are to be transferred as data. All control statements are transferred until a @FIN or @END (with matching sentinel) control statement is encountered (see 3.4.1 and 18.2.1, respectively).
L	Requests a listing of the complete symbolic element. Since this option cannot be specified for absolute or relocatable elements, it cannot be used with the A and R options. This option is assumed and eltname-1 is listed when the I, L, and U options are omitted.

*Source input routine options (see Table 9–3), except the W option, also apply.

Table 18–1. @ELT Control Statement, Options

Description:

The ELT,D control statement allows insertion of control statements into a program file as elements which may represent @RUN and @ADD runs streams (see 3.4.1 and 3.9.1, respectively) that can be called later by the @ADD or @START control statements (see 3.9.1 and 3.4.3, respectively).

When an element is punched by the FURPUR processor (see Section 8), the element is always preceded by an @ELT control statement. The filename punched into the @ELT control statement is the name of the file from which the element was punched. These decks can simply become part of the input to subsequent runs. If the element is to be added to a file other than the one from which it was punched, the filename punched must be changed.

Examples:

```
     LABEL          OPERATION                    OPERAND                  COMMENTS
1            10              20          30               40             50
1. @ELT,U PF1.ELEMENT1
2. @ELT PF1.ELEMENT1,PF2.ELEMENT2
3. @ELT,L PF1.ELEMENT2
4. @ELT PF1.ELEMENT2
5. @ELT,I PF1.ELEMENT3
6. @ELT,IA PF1.ELEMENT4
7. @ELT,IR PF1.ELEMENT5
8. @ELT,IS PF1.ELEMENT6
9. @ELT,ID PF1.ELEMENT7,,STOP
```

1. The correction images following this control statement update ELEMENT1 of program file PF1. The updated element replaces the old ELEMENT1 in PF1. Since the element type is not specified, the S option is assumed, and the element is considered to be a symbolic element.

2. The correction images following this control statement are applied to ELEMENT1 of program file PF1 to produce a new symbolic element (ELEMENT2) for program file PF2.

3. The correction images following this control statement are applied to ELEMENT2 of program file PF1. The new symbolic element is listed but no new element is produced because the U option is omitted.

4. This control statement lists ELEMENT2 of program file PF1.

5. The images following this control statement are inserted as a new symbolic element (ELEMENT3) into program file PF1.

6. The images following this control statement are inserted as a new absolute element (ELEMENT4) into program file PF1.

7. The images following this control statement are inserted as a new relocatable element (ELEMENT5) into program file PF1.

8. The images following this control statement are inserted as a new symbolic element (ELEMENT6) into program file PF1.

9. The images following this control statement are inserted as a new data element (ELEMENT7) into program file PF1. The data stream is terminated when an @END control statement having the matching sentinel STOP is encountered.

18.2.1. INPUT TERMINATION SENTINEL (@END)

Purpose:

Marks the end of a data file or element. It follows the data images introduced by either an @ELT,D or @DATA control statement (see 18.2 and 18.3, respectively).

For the @END control statement, only the sentinel parameter is optional; all other parameters are required.

Format:

```
@END    sentinel
```

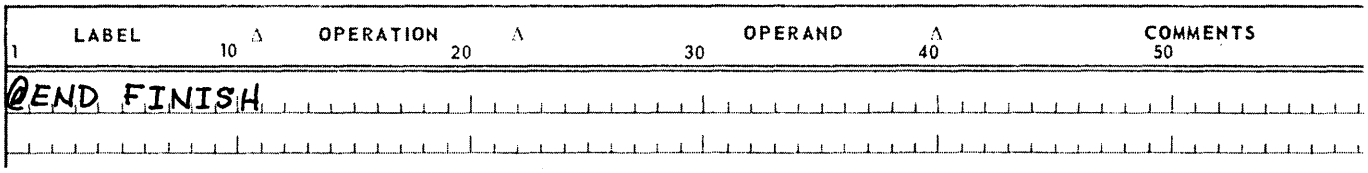
Parameters:

sentinel — A one- to six-character code corresponding exactly to the sentinel contained in the @DATA or @ELT,D control statement introducing the data images. The end of the file or element is determined when the character string in this parameter matches the character string of the sentinel parameter specified in the associated @DATA or @ELT,D control statement.

Description:

@END control statements cannot have labels and cannot be continued. The @END control statement must be coded exactly as shown (punched into first four columns of the card).

Examples:



```
   LABEL        Δ    OPERATION      Δ              OPERAND        Δ              COMMENTS
1            10               20                30             40                 50
@END FINISH
```

When the @END control statement is encountered, it ends the data file or element introduced by the @DATA or @ELT,D control statement that has its sentinel parameter coded FINISH.

18.3. DATA PROCESSOR

Purpose:

Introduces, updates, and corrects data files from the control stream. DATA processor operation is terminated by an @END control statement (see 18.2.1) whose sentinel matches the sentinel in the @DATA control statement. The @DATA control statement is used to call the DATA processor and it must precede the data or correction images in the run stream. Any control statement, with the exception of the @FIN and @ADD,D control statement (see 3.4.2 and 3.9.1, respectively) appearing between the @DATA control statement and the @END control statement is treated as data.

All parameters in the @DATA control statement are optional except @,DATA, and filename-1.

Format:

```
@label:DATA,options    filename-1,filename-2,sentinel
```

Parameters:

options
See Table 18-2.

If the I option is omitted, but both filename-1 and filename-2 are specified, the data images following the @DATA control statement are interpreted as corrections to filename-1. A new updated file identified by filename-2 is generated.

If the options and filename-2 parameters are omitted, the L option is assumed. Filename-1 is listed but no new file is generated.

filename-1
Specifies the file to which the data images and correction images in the run stream apply. If the file is to be catalogued it must have been previously assigned by an @ASG control statement (see 3.7.1).

filename-2
Specifies the updated file to be generated. If this file is to be catalogued, It must have been previously assigned by an @ASG control statement.

sentinel
Specifies a character code of one to six characters used for comparison purposes in determining the proper terminating @END control statement (see 18.2.1) for the data mode.

Option Character*	Description
L	Generates a complete listing of the file, including sequential item numbers which are used when making corrections to the file. If this option and filename-1 are the only parameters specified, filename-1 is listed.
U	Use relative F-cycle -0 as input to produce relative F-cycle +1 of the file.

* Source input routine options (see Table 9–3), except the U and W options, also apply.

Table 18–2. @DATA Control Statement, Options

Description:

The difference between the operation of the DATA processor and the @FILE control statement (see 3.8.1) is that the DATA processor handles data as it is presented in the run stream at run time, whereas the @FILE control statement builds the file as the data is being initially inputted into the system. In short, the DATA processor operates identically to a language processor control statement. The file built by the DATA processor is in SDF format (see 24.2.3).

The DATA processor allows the user to build data files which are an entire or partial run stream. These files can then be called on by the @START control statement (see 3.4.3) to start an independent run or by the @ADD control statement (see 3.9.1) for inclusion in a current or subsequent run. The DATA processor enables the user to make corrections to an independent run stream and then start it using the @START control statement, or make corrections to a partial run stream and add it to the run using the @ADD control statement. The DATA processor can also be used as a convenient means of generating and maintaining a user's data file rather than a control stream type file.

All files referenced in the @DATA control statement must be assigned to the run before the @DATA control statement is executed.

The U option is used only to update from relative F-cycle 0 to relative F-cycle +1, and only one filename can be specified. Relative F-cycle +1 of the file must be assigned prior to implicitly referencing it by means of the U option on the @DATA control statement (see 2.5.3).

Examples:

```
   LABEL       ∧   OPERATION       ∧           OPERAND        ∧          COMMENTS
1           10                 20          30              40              50
1. @DATA FILEA,FILEX
2. @DATA,I FILEB
3. @DATA,L FILED
4. @DATA FILEY
5. @DATA,IL FILEZ
```

1. The images following this control statement provide the corrections for FILEA. The updated version of this file is stored into the newly created file FILEX.

2. The images following this control statement are inserted into FILEB.

3. The images following this control statement are applied as corrections to FILED. FILED is listed but a new file is not created.

4. Because the options and filename-2 parameters are omitted, the L option is assumed, and a complete listing is provided for FILEY.

5. The images following this control statement are inserted into FILEZ and listed.

18.4. ED PROCESSOR

Purpose:

The ED processor is a text editor which allows the user to coversationally edit a symbolic file or element. It allows insertion, deletion, and replacement of text. The ED processor is called by the @ED control statement.

All parameters of the @ED control statement are optional except name-1.

Format:

@label:ED,options   name-1,name-2

Parameters:

options	See Table 18–3.
names	Specifies an input or output files or elements (see Table 18–3).

Description:

Table 18-3 lists the available options, the input/output files as specified by name-1/name-2, and functions. Unless otherwise specified, name-1 and name-2 may be either files or elements.

Option Character	Name-1	Name-2	Description
Not I, R, or U	Input	Output	Input is taken from name-1 and the resultant text is placed in name-2.
Not I, R, or U	Input	None	R option assumed for symbolic element. U option assumed for data file.
A	Input	Output	Attempt auto recovery – see AUTO command (Table 18–4).
B	Input	Output	Batch mode when using a demand terminal – ED run will not solicit input from user.
C	Input	Ignored	Enter input mode if element does not exist.
D	Input	Output	Demand mode when using a batch terminal – output listing of the ED run will contain solicitation messages.
I	Output	Ignored	Initial insertion of symbolic input from the run stream which causes the ED processor to enter the input mode. The images following the @ED control statement are inserted into the file named in name-1. The I option takes procedence over the R or U option.
L	Input	Output	Print all lines following the @ED control statement. The lines printed are indented and preceded by three asterisks.
N	Input	Output	Suppresses printing of changed or relocated lines. This option serves the same purpose as the ON BRIEF command.
R	Input	Ignored	Input is taken from name-1 of the @ED control statement; no output text is produced (read-only mode).
U	Input/ Output	Ignored	Update the symbolic element by applying corrections and create a new symbolic element cycle. The output element name is the same as name-1 but with a cycle number one greater. For SDF-formatted files the original images will be replaced with the updated ones.
X	Input	Output	Take an ERR$ exit (see 4.3.2.2) upon a fatal or nonfatal error (batch mode only).

Table 18–3. @ED Control Statement, Options

The ED processor operates in two modes: input and edit. In input mode, all lines entered are directly inserted into the text. In edit mode, various commands may be used to modify existing text. Changing between modes is accomplished by entering a blank line. Most editing commands implicitly reference a particular part of the text. This is accomplished by an internal cursor maintained by the ED processor. This cursor may be positioned directly by some commands (number, +number, -number) and indirectly by others (LOCATE, FIND, CHANGE).

18.4.1. EDIT MODE COMMANDS

Table 18–4 lists alphabetically the commands available to the ED processor while in edit mode (permits manipulation of images in an element or file). Some of these commands may be abbreviated to one or two characters as specified. All may be abbreviated to three characters.

When using CHANGE, INSERT, RETYPE, and the corresponding abbreviated commands, only one blank should be left between the command and the parameter image. In all other commands of more than one parameter, at least one blank must be left between each parameter. When errors are detected in a command, the command is not executed, an error message is given, and the next command in the run stream is executed, except in the case of the X option in batch mode.

Command	Description
ADD name or ADD name num1 num2	This command is used to add all or portions of a file to the current file. The first form adds the whole file, and the second form adds lines 'num1' through 'num2' to the current file. The lines to be added are inserted at the end of the file unless a + immediately follows the command in which case the lines are inserted following the current position within the edit file. The 'name' is the element or filename (see 2.6).
APPEND	Go to the end of the element or file and enter input mode thereby allowing new images to be inserted.
AUTO <num1	This command specifies that an automatic save of the current file is to be performed in case of processor or system failure. The 'num1' specifies that the auto save is to be performed for every 'num1' input transaction. When the auto save is performed, the word 'AUTO' will be typed out. If a failure does occur, the user should type in the @ED control statement exactly as he did just before the failure occurred with the addition of the A option. When the file is recovered, the words 'AUTO RECOVERY' will be typed out, and the user may proceed. If the I option was used, the user must enter a carriage return to change to edit mode before the auto recovery will take place. Caution: This should be used sparingly as it involves a large amount of I/O and computation. Some sites may choose to set a minimum for the line count for this reason. An AUTO 0 terminates the auto save mode. Entering the AUTO command with no operand causes an immediate auto to be performed without affecting the auto counter.
CCHAR char	This command sets the continuation character. When an input line to the editor has this character in it, the editor assumes that the next line or input is a continuation of this current line. This next line will be solicited in the normal manner except that a + will precede the solicitation. The character is initially set to a character which cannot be typed in. The character can be reset to this non-enterable character by using this command with no 'char'.
CHANGE/string-1/string-2/m G or C /string1/string2/m G	This command searches a specified number of text lines for 'string1' (as with LOCATE command except that CHANGE starts with the current line and LOCATE starts with the following line). When and if the desired string, 'string1', is found, 'string2' is substituted for it. The number of lines to be scanned is indicated by 'm'. The global indicator, 'G', tells whether to change all occurrences of 'string1' in the range of lines or just the first occurrence in each line. A 'G' means all occurrences and any other character (or no character) means just the first. The '/' may be any character which does not occur in 'string1' or 'string2' except for a blank. 'm' may be omitted in which case 1 is assumed. Instead of using 'm' and 'G', a user may change all subsequent occurrences in the file by using the word 'ALL' (abbr. A) where 'm' is usually specified.

Table 18–4. ED Processor Commands (Part 1 of 7)

Command	Description
CLIMIT column number	This command allows the user to set a limit on the number of columns which will be searched during performance of the change command. This is useful for protecting areas of data in a file. The default value is 132.
CSF executive control statement	This command is used to submit a control statement via CSF$ (4.8.1). Only statements valid for CSF$ may be submitted. The control statement must start in column 5.
CPT	This command prints out the CPU time used so far in the present run. The form of the message is (minutes)M (seconds)S.
CPUNCH num1 num2 CPUNCH num1 CPUNCH	This command is used to punch parts or all of a file at an onsite card punch. The syntax has the same meaning as with the SITE command. After the command is entered, a message as follows will be typed out: MSG? The line typed in here will be sent to the system console before the cards are punched.
DELETE num1 num2 D num1 num2 DELETE num1 D num1	This command is used to delete lines from the text. The first form deletes lines 'num1' through 'num2'. The second form deletes the next 'num1' lines starting with the current one. This command allows duplication of other lines in the file.
DITTO num1 DITTO num1 num2	The duplicated lines are inserted at the present position in the file. The first form results in the one line at 'num1' being inserted in the present position. The second form results in all lines 'num1' through 'num2' being duplicated at the present position. Care must be exercised to be sure the most current line numbers are used. At the completion of the DITTO, the pointer is positioned at the next line following the lines inserted.
EXCH char octal number	This command is used to allow input of characters not represented in the keyboard character set. 'char' is the character which is to be used to stand for the number whose internal representation is 'octal number'. When 'char' occurs anyplace in an input line it will be replaced by this character. An EXCH with no parameters disables this feature.
EXIT	This is the command used to take a normal exit from the ED processor. All the corrections will be applied to the designated file and a normal exit will be taken.
FIND mask	FIND searches for an image which corresponds exactly column for column starting at column 1 with the 'mask'. Transparent characters may be in the mask which will test successfully with any character in the column. The normal transparent character is a blank, but alternate ones may be designated with the TCHAR command. The search begins with the line following the current one and proceeds until a match or end-of-file is detected.

Table 18–4. ED Processor Commands (Part 2 of 7)

Command	Description
FC mask	The FC command behaves in the same way as the FIND command except that all occurrences are flagged in the remainder of the file.
IB string	This command behaves exactly the same as the INSERT command except that the line is inserted before instead of after the current line.
INLINE number term-sub	This command allows inline editing of a given line. If 'number' is blank, the current line is assumed to be the one to be edited. Otherwise the editor proceeds to line 'number'. The line will be printed out. The user can then enter editing information directly below the line to modify it. Following are the editing characters to be used. I – The string following this command is inserted following the character immediately above the I. The string is delimited on the right by the termination character '!'. R – The characters following the R will replace the characters immediately above them. A ! is required to terminate replacement. D – The characters in the line above are deleted between the D and the !. If one wished to use another character instead of ! for termination (for example if one wished to insert a !) then an alternate symbol may be specified as 'term-sub'. This will remain in effect for this INLINE only.
INSERT string I string	This command is used to insert a line following the line presently pointed at by the editor. This new line will then be the point at which the editor is positioned. The string to be inserted starts after the first blank following INSERT. If the command with no image is entered when not in EOF mode (see 'ON' command) the editor will switch to input mode. In EOF mode this simply results in the insertion of a blank line.
INPUT	This command directs the editor to enter a special input mode. In this mode everything which is typed in is inserted in the file until an exit from the mode is taken. This is especially useful when large volumes of input are to be entered. Exiting from this mode is accomplished by typing an @EOF when in EOF mode (see ON and OFF commands) or a carriage return when not. Tabs are recognized in this mode.
LAST	This command directs the editor to move to the last line in the test and stay in edit mode. The last line may not be altered at this time.
LNPRINT num1 num2 LNPRINT num1 LNPRINT!	This command behaves like the PRINT command except that each line is preceded with its line number. Syntax is the same as the PRINT command.

Table 18–4. ED Processor Commands (Part 3 of 7)

Command	Description
LNQUICK num1 num2 LNQUICK num1 LNQUICK!	This command behaves like the QUICK command except that each line is preceded with its line number. Syntax is the same as the QUICK command.
LNSITE num1 num2 LNSITE num1 LNSITE!	This command behaves the same as the SITE command except that each line is preceded with its line number.
LOCATE string LOCATE quote char string quote char	This command is used to search the text for a given string of characters. The search begins at the line following the current line and proceeds sequentially through the text until a find is made or the end of file is encountered. The first form ignores multiple blanks in the images. The second form requires that the text image be exactly the same as the string within the two quote characters.
LC string LC quote char string quote char	LC behaves as LOCATE except that all occurrences of the string in the remaining text are located. Just before each line containing an occurrence is typed out, the line number is typed out.
LCHAR char	This command sets the quote character for the LOCATE command. The default character is quote ('). A non-inputtable character will be assumed if 'char' is a blank.
MAIL userid	This allows the user to send messages to another user. The 'userid' is the original runid of the person to whom the message is directed. The editor will then solicit 10 lines of input with: MAIL** If the desired message is to be less than 10 lines the mode can be ended by entering an @EOF. After the message is received by the designated person it will be deleted.
MAXLINE number	This sets the maximum length to which a line may increase. If it is exceeded, the line will be truncated. The default is 80.
MOVE num1 MOVE num1 num2	This command performs the same operation as the DITTO command except that the original lines are deleted after the duplication has taken place. The syntax is the same as for the DITTO command. Care must be exercised to be sure the most current line numbers are used.
MSCHAR char	This command sets a character which will be translated to a masterspace when it is input in column one. If 'char' is a blank, no masterspace translation is available.
OMIT	This is the command to be used if the user does not want his corrections to be applied to the file on exit. The input file will remain as it was at the beginning of the editing session, and the output file, if any, will not be produced.

Table 18–4. ED Processor Commands (Part 4 of 7)

Command	Description
ON special mode ,..., special mode OFF special mode ,..., special mode	This command is used to define some special modes within the editor. ON turns the mode on, and OFF turns it off. The special modes are: QUICK – compress extra blanks out of all output to device. BRIEF – do not echo corrected images for CHANGE and DITTO. NUMBER – precede each line printed out with its line number. PCNTRL – recognize and print print control images DSPLIT – delete lines transferred by SPLIT command. XBRIEF – do not echo lines transferred by SPLIT or ADD commands. SEQ – print sequence numbers when soliciting input. LOOK – look for mail after each command is executed. EOF – special mode where blank lines may be entered. INP command enters input mode and @EOF exits from input mode to edit mode. While in input mode blank lines may be entered. Also the INSERT command with no image following will enter a blank line. MEMORY – remember modes on successive executions. All of the modes may be abbreviated to one letter.
OPR string OPR* string	This command is used to send a message to the system console. The first form sends the message 'string'. The second form does the same, but also solicits an answer. The string may not be more than 50 characters or they will be truncated.
PCN	This command is used to enter a print control image into the file being edited. When the command is entered, the editor will solicit the image with: CONTROL IMAGE- This image can only be read when in a special mode set by the ON command.
PLIMIT column number	This command is used to set a limit on the number of colums which will be printed out by the PRINT command.
PRINT num1 num2 PRINT num1 PRINT!	This command is used to print out lines of text. The first form prints lines 'num1' through 'num2'. The second form prints the next 'num1' lines. If the command is immediately followed with a + the printing starts with the next line instead of the current one (example: PRINT + 3). The third forms prints the entire file from the top. If no number or recognizable symbol follows the command, a 1 is assumed; that is, the present line will be printed out. This command may be abbreviated with a P.

Table 18–4. ED Processor Commands (Part 5 of 7)

Command	Description
PUNCH num1 num2 PUNCH num1 PUNCH	This command is used to punch paper tape for form II paper tape input (see 12.2.1.2.2) at a terminal which has punch and read hardware. The syntax for this command is the same as that for the PRINT command. When the command is entered, the following response will be given: DEPRESS PUNCH ON The processor will then pause to allow the user to push the punch on button on the paper tape punch hardware. It is suggested that the user previously punch several rubouts on the tape to make a leader for later reading the tape. This is accomplished by switching the paper tape punch on and holding the repeat key (REPT) and the rub out key down simultaneously. This may be safely done even while executing a run at the terminal since rubouts are ignored by the operating system. After pausing the designated lines will be typed out which will cause the paper tape to be punched at the same time. When the typing is finsihed, the editor will again pause to allow the user to switch the punch off, rubouts should also be used at the end of the tape. The tape so produced can be used as normal form II input.
QUICK num1 num2 QUICK num1 QUICK!	This command prints lines with all nonsignificant blanks omitted. This provides a fast method of examining areas of the file. 'num1' and 'num2' are the same as on the PRINT command. Plus (+) may also be used on the second form with the same meaning. This command may be abbreviated with a Q.
RETYPE string	This command is used to completely replace the current line with the string following the first blank after the command. A + may be used after the command with the same meaning as with the INSERT command.
RP number	This command is used to set a repeat counter for the INSERT command. Any insertion will be repeated 'number' times.
SCALE number	This command causes a line to be printed out which can be used for column sensitive operations. The form of the line is: 1234567890123456789012345678901234567890... starting in column 'number'.
SET tab1 tab2 tab3 ... tabn	This command is used to set the tabs for the commands which allow them as explained above. As many tabs as desired may be designated. Each SET command redefines all previous tabs, and so a SET with no tabs clears the tabs. If no SET has been performed a default of 11,21,39,73 is assumed.

Table 18–4. ED Processor Commands (Part 6 of 7)

Command	Description
SITE num1 num2 SITE num1 SITE!	This command is used to direct output to an onsite printer (PR). The meanings of 'num1' and 'num2' are the same as for PRINT except that if no numbers are given, the third form is assumed. After this command is entered, a message as follows will be typed out: HDG? The line typed in here will be used to head the onsite output. Periods must not be used in this header as anything beyond the period will not be printed. After the output is done, the following will be typed: MSG? The user should enter the information here necessary to indicate where and to whom the output should be returned.
SPLIT name SPLIT name num1 num2	This command is used to build new elements or files from portions of a current file. The first form causes all the lines preceding the line currently pointed at to be reproduced as the designated file. The second form causes lines 'num1' through 'num2' to be reproduced. An '!' immediately after the SPLIT command causes the whole file to be copied.
STATUS special mode ,..., special mode	This command is used to request the status of special modes set by the ON and OFF commands. If no special modes are specified, the status of all will be listed.
TAB tab char	This command is used to specify which character is to be used as a tabulator character. This character is recognized on the INSERT, IB, and RETYPE strings and is recognized on all input when in the input mode. The character is not transmitted to the file and behaves just as a tab on a typewriter. If no character has been specified, a semicolon (;) is the tab character.
TIME	This command prints out the date, time, and cycle information.
TYPE processor-mnemonic	Sets the processor type for symbolic element output. The processor mnemonics are: ALG, ASM, COB, DOC, ELT, FOR, MAP, and SEC.
UP	This command is used to cause an element or file to be saved as if the U was specified on the CONTROL statement. This is used if the entry to the editor was made with an R option.
number + number – number	These commands are used to position the editor at a desired line in the text. The first form directs the editor to line 'number'. The second form directs the editor to move to the position current line plus number. The third form directs the editor to move to the position current line minus 'number'. When the specified line is located, it is typed out if in VERIFY mode, and modifications may be made to it. If it is desired to insert lines before line 1, 0 may be typed in. This will position the editor immediately before the first line.

Table 18–4. ED Processor Commands (Part 7 of 7)

18.4.2. USAGE CONSIDERATIONS

The ED processor proceeds sequentially through the text. It is therefore more efficent to perform editing operations in a more or less sequential manner starting at the beginning of the text. Searching commands such as LOCATE and CHANGE require much computation and should be used sparingly.

There are certain processes within the editor which if indiscreminately interrupted can cause the processor to fail. To protect against this, the processor is designed to break only at specified points when it is safe to do so. If the user wishes to interrupt the processor, he may depress the break key at any time. The system will respond with:

```
INTRPT LAST LINE
```

The user should answer with an X if he wishes the processor killed or a carriage return if he wishes only to escape the current command. In either case a few lines of backed-up printout may follow before the interrupt takes place. If for some reason the editor's escape method is not satisfactory the user may enter the break key and carriage return again. In this case the editor will return to edit mode, but integrity is not guaranteed.

Files with names of the form ED$xx (where x is any character) should be avoided since the ED processor uses such files internally.

18.5. CULL PROCESSOR

Purpose:

Scans a collection of symbolic elements and produces a cross-reference listing of the symbols found, the elements, and the lines on which they occur. A list of symbols which are either to be omitted from the sort or the only symbols to be included in the sort may be specified. The CULL processor is called by the @CULL control statement.

All parameters in the @CULL control statement are optional except @ and CULL.

Format:

```
@label:CULL,options   pro/scol(res),name-1,...,name-n
```

Parameters:

options — Specifies available options for obtaining sorted symbolic listings (see Table 18–5).

pro — Specifies the symbolic element being scanned/sorted. The possible values are:

ASM – 1100 series assembly code

DATA – all symbolic elements

ELT – symbolic data element

If omitted, ASM is assumed.

scol — Specifies the column number where the scan automatically stops. If omitted, the following values are assumed:

ASM - 40

DATA/ELT - full card

res — Specifies the number of mass storage positions reserved for scratch files. If omitted, 2 is assumed (approximately 70,000 references).

names — Specifies elements or filenames in standard notation (see 2.6.6).

Option Character	Description
General Options	
C	Produces a 72-column listing (teletypewriter or page size) omits list of elements processed and summary information at end
E	Does not eject the page. Otherwise, page eject for each time symbol begins with different character than previous symbol.
L	Produces a full listing of each element scanned/sorted.
M	Only symbols in data images are accepted.
N	Marked symbols only are printed; otherwise only unmarked symbols printed.
O	Symbolic elements are not sorted
P	Procedures are not examined
S	Symbols in data images are printed
U	Causes used symbols to be marked (and hence not printed in absence of N option, thus giving listing of symbols defined as labels but not otherwise referenced).
Z	Accepts untyped symbolic images
Special Options for Symbolic Data Elements	
A	Does not sort strings beginning with an alphabetic character
D	Does not sort strings beginning with any character from the set: 0–9, $, and period (.)
Special Options for Assembler Language Elements	
A	Does not sort identifiers
D	Sorts numbers and the symbol $
N	Prints only marked symbols (otherwise, prints only unmarked symbols)
I	Any symbol that appears in the special internal table is not sorted. This table contains all symbols that are commonly used as directives, mnemonics, register names, and j designators. If the M option is specified, the internal table is used as a supplement to the special symbol table.
Q	Sorts data items from the set A–Z, 0–9, $, and period (.) which are enclosed in quotes. These references are followed by a Q.
W	Marks a symbol when it appears in the label field.
X	Marks a symbol when it appears in the label field and is followed by an asterisk (*).
Y	Restricts the W and X options to elements explicitly named in the @CULL control statement.

Table 18-5. @CULL Control Statement, Options

Description:

Each file must be assigned or catalogued. If no filename is given, TPF$ is assumed. All files are returned to their original status when the CULL processor terminates. If a file is not a FASTRAND-formatted program file, an error is noted, and the parameter is ignored.

If read/write keys are necessary, they must be given in the first reference to the file. Keys appearing anywhere else are ignored.

The set of elements to be scanned and sorted is formed in one of two ways:

- If an element name is a parameter, it is included in the scan/sort.

- If a filename is a parameter, the most recent cycle of each symbolic and procedure element is included in the scan/sort, unless the O and P options are specified.

Standard system dropout rules apply to both file and element name formats (see 3.2.7).

Data images, following the @CULL control statement, are scanned for strings of character separated by blanks; these strings are the special symbol table. If the M option is specified, only the symbols in this table are accepted; if omitted, the symbols in this table are ignored. If the S option is used, the special symbol table is printed. Processing does not begin until the end of the data is encountered (indicated by encountering a control statement.

Certain options specify that a symbol which is found under special circumstances is to be marked. When a symbol is to be marked, then all occurrences of that symbol elsewhere are to be marked. If the N option is not specified, only unmarked symbols are printed. If the N option is specified, then only marked symbols are printed.

The following special options are available:

- Special Options for Symbolic Data Elements

  The data scanner accepts strings of characters from the set A–Z, 0–9, $, and period (.) up to 12 characters in length. See Table 18–5 for the special options available only to the data scanner.

- Special Options for Assembly Language Elements

  The assembler scanner accepts strings which are valid identifiers/numbers in the 1100 series assembly language. These strings may include up to 12 characters, and the symbol, $, is recognized. The assembler scanner recognizes an identifier when it is a label, directive, or operand; occurrences of labels and directives are followed by * and D, respectively. Labels defined at another level are marked with a double asterisk. The special options available to the assembler scanner are given in Table 18–5.

Examples:

```
     LABEL      Λ   OPERATION   Λ              OPERAND      Λ              COMMENTS
  1          10               20             30              40             50
1.@CULL
2.@CULL,D       A$M/80(30),EXEC1.,EXEC2.,EXEC3.,EXEC4.
3.@CULL,D       DATA/80(10),PRM/REV21
4.@CULL         ,Q*F.A,.B,.C,.D
```

1. Sorts all of the temporary program file (TPF$) using the assembler scanner.

2. Used to obtain a complete scan and sort of the assembly language files EXEC1 through EXEC4 and to reserve 30 positions of mass storage for use by the CULL processor.

3. Data element PRM/REV21 contains images for a document. This control statement is used to generate a glossary of words used.

4. Used to sort elements A, B, C, and D in file Q*F.

18.6. LIST PROCESSOR

Purpose:

Produces an edited listing of any type of element. The LIST processor is called by the @LIST control statement.

All parameters in the @LIST control statement are optional except @, LIST, and eltname-1.

Format:

```
@label:LIST,options    eltname-1,...,eltname-n
```

Parameters:

options

Only one of the A, R, and S options may be specified; if neither A, R, nor S is specified, S is assumed.

A – Absolute elements

R – Relocatable elements

S – Symbolic elements

The O option is used with A, R, or S to dump erroneously-formatted elements in octal. The dump is not edited. This option is generally used for dumpint erroneously-formatted elements.

eltnames

Specifies the elements

Description:

The edited listing contains the following information for each type of element:

- Symbolic elements

  – Every SDF-formatted image in the element, including control images, is printed with the length and relative word address of the image.

  – The line numbers of the symbolic images belonging to the most recent symbolic cycle are printed. The cycle information for all symbolic images is printed.

  – If the symbolic element is an assembler, COBOL, or FORTRAN procedure, the appropriate procedure name table is printed.

- Relocatable Elements

  – Each text word is printed as 12 octal digits. The j field (bits 29–26), a field (bits 25–22), x field (bits 21–18), and h, i fields (bits 17–16) are printed below the text word.

  – The following abbreviations are used when the relocation information is printed:

LA –	Left address (bits 33–18)
LC –	Location counter
LH –	Left half (bits 35–18)
RA –	Right address (bits 15–0)
RH –	Right half (bits 17–0)
XR –	External reference

- Absolute Elements

  – Each text word is printed as 12 octal digits (see first entry under relocatable elements).

  – The following abbreviations are used when the relocation information for the relocatable segments is printed:

L –	Left half relocated
R –	Right half relocated

18.7. CON78 PROCESSOR

Purpose:

The CON78 processor accepts CUR-formatted symbolic elements on magnetic tape and converts them to FURPUR-formatted symbolic elements. The CON78 processor is called by the @CON78 control statement.

The only optional parameter on the @CON78 control statement is the options parameter.

Format:

```
@label:CON78,options   filename-1.,filename-2.
```

Parameters:

options — The options specify the element types. The options are:

C –	COBOL library procedure
P –	procedure
S –	symbolic

filename-1. — Specifies the input tape file containing the CUR-formatted elements.

filename-2. — Specifies the output tape file to contain the FURPUR-formatted elements.

Description:

Both the input (filename-1) and output (filename-2) files must be tape files and must be assigned to the run.

Each call to the CON78 processor converts only one file.

An EOF mark is written after each converted file; neither file is rewound.

COBOL procedures begin with a PROC line and end with an END line. The label of the procedure line is the element name and is not external defined; the converted procedure must be converted by PDP (see 9.7) before it can be used.

If the options parameter is omitted, all types of symbolic elements are converted.

Examples:

```
    LABEL          Δ     OPERATION        Δ                 OPERAND          Δ                  COMMENTS
1                 10                  20                   30                  40                   50
@RUN        N333,TREP,02212,10/23
@ASG,M      INPUT.,8T,332C
@ASG        PFILE.,T
@CON78,C    INPUT.,PFILE.
```

The EXEC II CUR-formatted element in file INPUT is converted into an element file (named PFILE) suitable for use by the FURPUR processor.

19. FILE ADMINISTRATION PROCESSOR (SECURE)

19.1. INTRODUCTION

The SECURE processor protects the physical security of catalogued files, which reside on mass storage, by producing tape backups. At periodic intervals, each file is examined to determine whether any write operations have occurred since its previous backup was saved, in which case a new backup must be made.

The text of files on mass storage may be destroyed either inadvertently through system failure or user error, or purposely to reduce overcrowding of facilities or to remove certain mass storage units from the available facilities pool. In any case of purposeful destruction, the presence of a current backup must first have been assured. Because a file may be inadvertantly destroyed and the latest backup may not be current, a record must be kept of the files memory lapse. Memory lapse is defined as that time period that starts at the first updating after the latest backup was created and ends with the recovery of the file from the backup copy. This is the period of time during which any additions or deletions were not retained.

When a file's text on mass storage has been destroyed and the backup is the only available copy, the file must be marked unloaded, so that an automatically initiated load of text back to mass storage occurs when the next attempted assignment of the file is made. The run which makes the @ASG request that forces this load is held in a wait status until the load is completed.

The process of selecting files as potential candidates for unload, when n number of currently used tracks must be vacated and made available for new allocation, requires a best-guess algorithm to select those files which will probably be the last to be reassigned, in order that the files actually unloaded can be left dormant for as long as possible before they have to be reloaded. Furthermore, there must be sufficient flexibility in the formulation of the unload mechanism to permit qualified onsite personnel to make dynamic adjustments to the individual weight attached to each of the variables which go into this unload eligibility factor determination.

An unload inhibit option is defined for use by certain files which cannot be removed from mass storage due to real time or other needs. There is also an even more restrictive guard option which prevents even the privileged read necessary to make backup copies. The guard option is required for certain special files which are internal to the system, highly transient, or highly classified.

In addition to the basic SAVE, UNLOAD, and LOAD commands, there are supporting commands to register unknown files recorded on backup tapes made at another UNIVAC 1100 series systems site and to list the memory lapses that have occurred for a file. It is also possible to allow all commands to be selectively directed, when desired, towards only certain named files, projects, accounts, tapes, or mass storage units.

Finally, it is possible for the SECURE processor to assist in a sophisticated catalogued file recovery process. This depends on a tape copy checkpoint of the master file directory (MFD) and the entire set of file backup tapes to restore the set of catalogued files to a state at least as current as the time the MFD checkpoint was made. A file is considered restored when MFD items can be retrieved from mass storage and its text can be retrieved from mass storage or tape.

19.2. MAJOR FUNCTION DEFINITIONS

Five major commands have been defined for use with the SECURE processor.

These are as follows:

(1) SAVE creates a duplicate copy on tape of both the MFD information and the text of a catalogued mass storage file, and then updates the MFD to record the reel numbers used and other pertinent information relating to the backup tape.

SAVE, and all other actions of the SECURE processor, are not performed on files which were catalogued with a G (guard) option on the @ASG or @CAT control (see 3.7.1 and 3.7.3) statement. The purpose of the G option is to override the privileged mode capabilities of the SECURE processor for particular files. Several of the exeuctive's internal files, including the scheduling file, the accounting file, and the symboint files use the G option.

(2) UNLOAD implies SAVE unless there already exists a tape backup copy of the file that was made by SAVE after the last write was done on the file. UNLOAD then releases the space occupied by the text of the file on mass storage and updates the MFD to mark the file unloaded. UNLOAD, or any other action dependent upon unloading the file, is not performed if it was catalogued with a V (unload inhibit) option.

(3) REMOVE implies UNLOAD, unless the file is already unloaded. REMOVE then causes the file to be decatalogued from the MFD.

(4) REGISTER scans the tape reels named in source language which contain SECURE-produced backup copies of mass storage files, and causes files that are not currently catalogued and whose complete backups reside on these reels to be catalogued as unloaded files.

(5) LOAD operates only on currently catâlogued files that are marked unloaded. LOAD copies the text of the file back to mass storage and turns off the unloaded indicator in the file's MFD entry.

19.3. @SECURE CONTROL STATEMENT

The name SECURE indicates that the purpose of the UNIVAC 1100 series systems file administration processor is to protect the physical security of data in permanent files.

All parameters on the @SECURE control statement are optional.

The format of the @SECURE control statement is:

```
@label: SECURE,options   eltname-1,eltname-2
```

The eltname-1 and eltname-2 parameters name the symbolic input and output elements, respectively; see 9.4.1 for rules governing their use.

Table 19–1 describes the options that can be specified on a @SECURE control statement. In addition to these options, the source input routine options (see Table 9–3) may also be used.

OPTION CHARACTER	DESCRIPTION
A	Do not take error exit even if errors are detected.
C	Enable the checksum feature in the SECURE processor to compute and write to tape a checksum total for each text block written by a SAVE command and use this value as a check against random I/O errors which may be introduced when data is transferred from tape during a LOAD command.
L	Produce the most comprehensive printed listing.
N	Suppress all listing except error diagnostics.
R	Scan all files marked disabled and print summarized results. Do not process any source language. @SECURE,R is called for within the executive following a recovery bootstrap, and is not of interest to the normal user. The @SECURE,R controls statement may be used only from within a privileged run, that is, the SYS$*DLOC$ file is assigned to the run (see 22.3).
S	Produce a summary printed listing.
X	Take error exit if all specified tasks cannot be processed.

Table 19–1. @SECURE Control Statement, Options

19.4. INPUT AND OUTPUT BACKUP TAPE ASSIGNMENTS

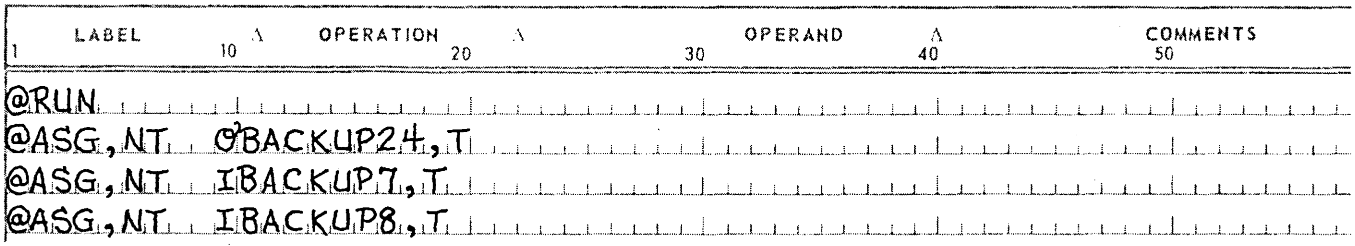
Users calling the SECURE processor should normally first assign adequate tape units for input and output of backup tapes by means of control statements with the following format:

```
@ASG,NT   OBACKUPnn,T

@ASG,NT   IBACKUPnn,T
```

The @ASG,NT control statement causes the tape unit to be assigned temporarily and with the initial tape load message suppressed. nn is an optional one- or two-digit number from 1 to 63 used to distinguish between multiple OBACKUP or IBACKUP names. If nn is omitted, 1 is assumed.

All SECURE processor operations involving either tape reading or writing can be performed on tape units assigned as output backup (OBACKUP[nn]). As a safety feature to help protect the contents of existing backup tape reels, only reads are performed by the SECURE processor on tape units that are assigned as input backup (IBACKUP[nn]).

Some examples of these assignments as they might appear in a single run are as follows:

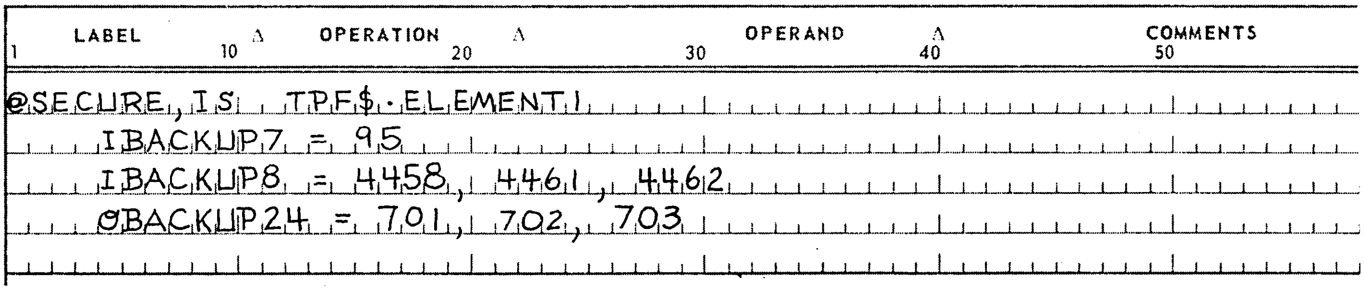


LABEL
OPERATION
OPERAND
COMMENTS
1
10
20
30
40
50

```
@RUN
@ASG,NT  OBACKUP24,T
@ASG,NT  IBACKUP7,T
@ASG,NT  IBACKUP8,T
```

If the operations to be processed do not require the predefinition of specific tape reel numbers that would otherwise be unknown to the SECURE processor, the user need not be concerned with identifying them. For example, the SECURE processor dynamically requests as many new blank tape reels as necessary when creating new backups and records the reel numbers used in both the MFD item of the backed up file and in the user's printed output listing. As another example, the SECURE processor automatically consults the MFD item to get the correct reels necessary to load the text of a file back to mass storage.

There are instances, however, when a specific sequence of numbered reels should be associated with a particular OBACKUP or IBACKUP tape unit. These associations are specified by source language statements of the format IBACKUPnn = REELLIST or OBACKUPnn = REELLIST, as follows:



```
LABEL      OPERATION         OPERAND          COMMENTS
1        10        20        30        40        50
@SECURE,IS TPF$.ELEMENT1
    IBACKUP7 = 95
    IBACKUP8 = 4458, 4461, 4462
    OBACKUP24 = 701, 702, 703
```

As an example, the SECURE processor allows a set of reels in backup format to be registered with the executive, so that any files previously saved to these reels, but which are not currently catalogued, are recatalogued as unloaded, mass storage files. Following a reel number association like that done above for IBACKUP8, the REGISTER command could be invoked by the single control statement: REGISTER FROM IBACKUP8.

When a particular SECURE operation involves the transfer of many text blocks to or from tape, and several tape units are assigned, the SECURE processor initiates multiple concurrent I/O operations to optimize output.

If no usable backup tape units are assigned at the time that the SECURE processor determines that tape I/O must be done, the SECURE processor executes a single dynamic tape assign by supplying an @ASG,TN OBACKUP,C image using a CSF$ request (see 4.8.1).

The most common cases where the SECURE processor will need tapes are the periodic SAVE commands needed to generate a set of new backups and the LOAD of an unloaded file which some run is attempting to assign. It is not necessary for a site to keep a tape unit available at all times just to enable SECURE to handle these two common cases. The SECURE run is placed in a facility wait state until the tape unit becomes available.

System-initiated UNLOAD's to relieve overcrowding of mass storage can sometimes be accomplished without creating new backups (or requiring tape unit assigns). However, if tape assigns are necessary and no tape units are available, the SECURE processor initiates a console message informing the operator that a tape unit must be made available. The operator may make a tape unit available by either restoring a tape unit that is in a reserved or downed state or by terminating another run (by using a checkpoint, E, or X keyin) that has a tape unit assigned.

19.5. CATALOGUED FILE ASSIGNMENTS

The SECURE processor performs no actions on files which are actively assigned to another run at the time that SECURE is executing, unless those files were catalogued as read only. Files to be operated on by the SECURE processor are dynamically assigned with an exclusive-use option to prevent other runs from writing on current write-enabled files. Performing a sequence of reads on a file that is in the process of being changed may cause problems ranging from inaccurate information on a single file to an abort of the entire SECURE execution.

If the number of runs in the system is temporarily reduced at the time a SECURE processor SAVE run is called, the problem of files not getting backed up, due to use by other runs, is minimized. To further reduce this problem, SECURE normally attempts a second or third dynamic @ASG,AX of those files that were busy during the first pass.

19.6. USE OF SYS$*DLOC$

SYS$*DLOC$ is the name given to a mass storage file of any size, which is catalogued with both read and write keys following the initial generation of a system. The keys to SYS$*DLOC$ are chosen at the site and should be known only by trusted personnel, since anyone knowing the keys can assign this file and thereby place his run into what is known as privileged mode. Once in privileged mode, the run may bypass keys to gain control of many other catalogued files.

It is possible for functions within the executive which create and start runs, to call on the SECURE processor to place these runs in privileged mode by using methods that are not available to user programs. All others must have prior knowledge of the keys to SYS$*DLOC$ to initiate a privileged run. The keys to SYS$*DLOC$ can be changed when desired by using the normal FURPUR @CHG control statement (see 8.2.15).

A run in privileged mode indicates that a program within the run (such as the SECURE processor) can access the text and full MFD information for all files catalogued in the system that do not have a G (guard) option inhibit on them, without supplying any of the keys and without regard to whether the file might be catalogued as read only or write only. This permits the SECURE processor to initiate I/O and other operations on a file that are necessary to create backups, or to delete or restore text.

One of the executive interfaces which check to see if a run is privileged is the MSCON$ request (see 22.3). The MSCON$ request allow direct reading and altering of MFD items. The DGET$ request (see 22.3.1), which gets a copy of the entire MFD and writes it into a user-specified file, provides an example of the distinction between privileged and nonprivileged runs: if the run is privileged, the MFD is copied unchanged; if the run is not privileged, DGET$ obscures all project-id's, account numbers, (other than those of the calling run), and all keys.

The SECURE processor operates primarily in the privileged-run mode. The operations which a user can direct SECURE to perform when his run is not privileged are summarized in 19.9.

19.7. SECURE SOURCE LANGUAGE

SECURE employs a source language structure for input which gives the user a simple, but flexible, format for calling on the processor to perform any or all of the allowed functions. Basic source language components are ordered as follows:

```
     command   ALL   limiters   name-list   EXCEPT   name-list;   FROM , equipment-name   TO   equipment-name
```

All parameters are optional except command.

19.7.1. STANDARD COMMANDS

The commands recognized for the SECURE processor are:

```
     SAVE

     UNLOAD

     REMOVE

     REGISTER

     LOAD

     LIST REELS

     UEF +nn namelist

     UEF –nn namelist

     REVERT filename

     LIST LAPSES [namelist]

     CLEAR LAPSES namelist

     END
```

Unless the word ALL is used, the SAVE command causes saves to be done only on those files which do not have a current backup. A backup is current depending on whether any write operations have been performed on the file since the time the last backup was made. However, when the word ALL is used, the SAVE command causes all files in the namelist to be saved, regardless of whether a current backup exists.

A file's disabled status does not prevent a new backup copy from being made, if required. In the case of a disabled file, however, the SECURE processor must also guarantee to preserve a record of the previous backups in the file's MFD items. If more than one backup copy is recorded in the MFD, the user must determine which backup should be retained as the primary copy. Otherwise, by default, the most recent backup becomes the primary copy.

Unless a namelist is given or the word ALL is used, the UNLOAD command does not cause more files to be unloaded than is necessary to free 3000 tracks on mass storage. The particular files chosen for unload, in this case, are selected using procedures explained in 19.8. To change the preset limit of 3000 tracks to some other value, the UNLOAD specification may be stated as: UNLOAD TRACKS = nnnnnn. If a namelist is given, all files so specified are unloaded, regardless of how much or how little space they occupy on mass storage. Finally, if the word ALL is used, all catalogued files are unloaded.

The REMOVE command requires either a namelist or an explicit ALL to specify the set of files which will be removed from the system. This set of files is deleted from the MFD, after the presence of a current backup copy for each has been assured.

The REGISTER command requires a

FROM BACKUPnn equipment-name

to which reel numbers have been associated. Unless a namelist is given, the REGISTER command operates on all files found on the reels associated with IBACKUPnn. Reel number associations are discussed in 19.4.

The command LOAD...FROM IBACKUPnn combines the REGISTER and LOAD actions in one operation.

The LIST REELS command produces a summary listing of the current set of backup tapes sorted in ascending order by reel number. This command must be in a separate SECURE execution with no other source language present.

The UEF +nn or UEF –nn commands may be used to selectively add or subtract a number nn from the computed unload eligibility factor (UEF) for the named files, projects, or accounts. This factor determines the order in which files are unloaded when mass storage space becomes crowded.. The UEF bias remains in effect only for the current execution of SECURE. The larger the UEF is for a given file, the more eligible the file is for unloading.

REVERT filename means revert to a previous backup copy of the named file. This can be used when a user accidentally overwrites the latest copy of the file on mass storage and wants the text in the existing backup to retained instead of the more recent text now residing on mass storage. When the text of files on mass storage is inadvertently destroyed, the backup tape becomes the primary copy. If, however, the backup tape is not current, a record is kept of the time period during which any addition or deletions to the mass storage file were not retained. This record represents a file's memory lapse and any number of these may possibly occur over the life of the file.

The LIST LAPSES command provides a printed listing of the memory lapse entries for all files or for the set of files specified by the namelist.

The CLEAR LAPSES command requires a namelist and erases the record of any existing lapse entries in the set of files specified by the namelist.

END is an optional terminator to mark the end of source language statements.

19.7.2. NAMELISTS AND LIMITERS

Namelists are strings of file, project, or account names which designate particular file sets. They are preceded by an appropriate identifier as follows:

FILE(S) filename-namelist

PROJECT(S) project-namelist

ACCOUNT(S) account-namelist

Names in the list are separated by commas. Actions specified in source language are limited to the set of files specified in a namelist. If no namelist is used with a particular command, a distinction must be made. If the run is privileged, it is assumed the action is to be applied to the entire set of catalogued files. If the run is not privielged, it is assumed that the action is to be applied only to those files under the user's project-id. Exceptions to this general rule are given in the description of each command (see 19.7.1). A limiter may be specified to restrict the file set described in the specified or implied namelist to files within the named category. The limiters that may be used are: PUBLIC, PRIVATE, READ-ONLY, and WRITE-ONLY.

19.7.3. EXCLUSIONS

EXCEPT preceding a namelist, causes those files, projects, or accounts in the list to be excluded from the particular action.

19.7.4. DIRECTION

When an action is to be directed to or from particular units of equipment, a FROM or TO designator is used from the following set, where ss/uu refers to mass storage subsystem/unit numbers:

FROM IBACKUPnn

FROM OBACKUPnn

FROM unit(s) ss/uu-list

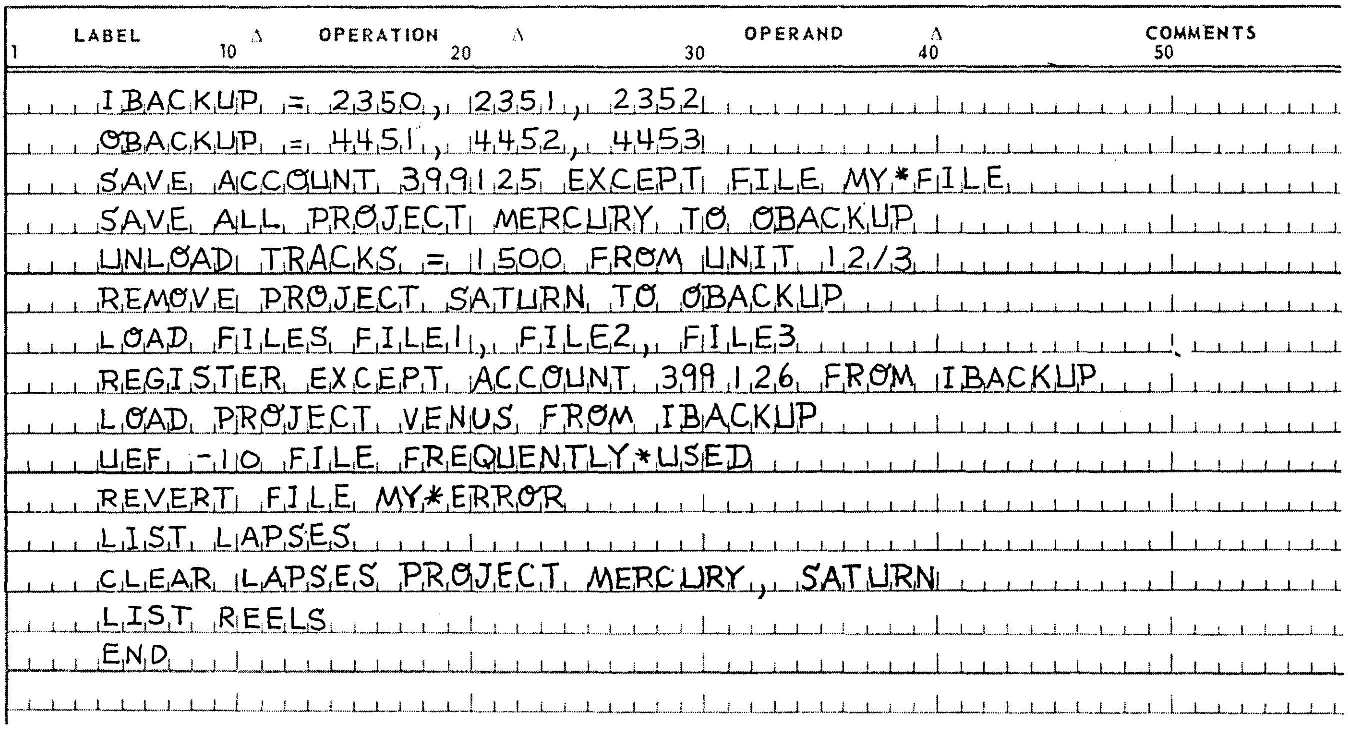
TO OBACKUPnn

TO unit(s) ss/uu-list

The user is cautioned to use these FROM and TO designators only where specific examples on the use of the SECURE processor indicate their use is meaningful or desirable. An attempt to move files from one mass storage unit directly to another using FROM and TO would not produce desirable results.

19.7.5. EXAMPLES OF SOURCE LANGUAGE

The following are examples of source language specifications:


LABEL
OPERATION
OPERAND
COMMENTS
1
10
20
30
40
50

```
    IBACKUP = 2350, 2351, 2352
    OBACKUP = 4451, 4452, 4453
    SAVE ACCOUNT 399125 EXCEPT FILE MY*FILE
    SAVE ALL PROJECT MERCURY TO OBACKUP
    UNLOAD TRACKS = 1500 FROM UNIT 12/3
    REMOVE PROJECT SATURN TO OBACKUP
    LOAD FILES FILE1, FILE2, FILE3
    REGISTER EXCEPT ACCOUNT 399126 FROM IBACKUP
    LOAD PROJECT VENUS FROM IBACKUP
    UEF -10 FILE FREQUENTLY*USED
    REVERT FILE MY*ERROR
    LIST LAPSES
    CLEAR LAPSES PROJECT MERCURY, SATURN
    LIST REELS
    END
```

19.8. SELECTION OF FILES FOR UNLOAD

If an UNLOAD command is given without naming any particular files, projects, accounts, or mass storage units, it is assumed that the SECURE processor has the responsibility to scan the entire set of catalogued files and to select the subset that must be unloaded to acquire the addition assignable mass storage space that is needed.

This differs from the action of the SECURE processor when particular files, projects, or accounts are named for unload, in which case all eligible candidates in the named set are unloaded. This is also in contrast to the action of UNLOAD ALL, where all eligible files in the system are unloaded.

When the SECURE processor is called to do saves to backup tape, drum-to-tape I/O is unavoidable. When the SECURE processor is called to do only an unloading operation, however, it is sometimes possible to avoid tape I/O. This situation results when there exists an adequate reservoir of current backups of unload-eligible files already out on tape to meet the requirements of a general UNLOAD command.

Often, at the same time that mass storage facilities become overcrowded, other system resources become inadequate, making it necessary to temporarily postpone saves. For this reason, those files with current backups are considered first in computing unload eligibility. Note that a SAVE command can be included, prior to the UNLOAD command, if it is intended that all backups be immediately current, prior to unload eligibility factor determination.

The criterion should then be to select a set of files for unload which satisfies the request for space to ensure a maximum amount of time before any one of the files selected is referenced again.

19.9. OWN-PROJECT APPLICATIONS

Although the SECURE processor is designed primarily for privileged-mode runs to guarantee the security and availability of catalogued files, it is recognized that the user does have individual control over those files under his own project-id and for which he has the necessary keys. Since the user can access these files anyway, he should be allowed to use that set of SECURE commands that lend themselves most logically to use by individuals. These commands are: SAVE, REGISTER, REVERT, LIST LAPSES, and CLEAR LAPSES.

Runs that call on the SECURE processor and do not have the SYS$*DLOC$ file assigned automatically fall within this OWN-PROJECT category. The only restrictions placed on the user in referencing SECURE commands from the previously described set are:

(1) files named in source language namelists must be catalogued under the user's own project-id; and

(2) both the read key and the write key, if they exist, must be included with the filename in source language statements.

The SAVE command, when referenced from within an OWN-PROJECT run, produces a backup tape. The reel numbers however, are not recorded in the file's MFD item. This is necessary to maintain control over SECURE backup tapes created by a privileged-run, system-wide SAVE. With the modified or unrecorded SAVE, a user can produce backup tapes at will without destroying the record of the current backup tapes under the site manager's control.

Own-project users of the SECURE processor can also REGISTER any files on backup tape which are under the user's project-id; that is, any user can transfer his own files from one UNIVAC 1100 series systems site to another by doing a SAVE at the old site and a REGISTER at the new site.

Logically, the REVERT command is called by the individual user after he has inadvertently destroyed the text of his file on mass storage. The SECURE processor immediately marks the file as unloaded so that an automatically initiated LOAD of a previous backup copy occurs when the file is next assigned.

The LIST LAPSES command is normally used by the individual user, following the assignment of a file, to see if any new lapses have occurred. The CLEAR LAPSES command may be used when the user is satisfied with the current state of his file and no longer cares to keep information about the file's previous history.

19.10. CATALOGUED FILE RECOVERY APPLICATIONS

No catalogued file should be considered disabled or destroyed as long as a SECURE-produced backup copy exists on tape. With this in mind, three modes of file recovery are possible under SECURE:

(1) REVERT to the backup copy

(2) REGISTER individual files from tape

(3) REGISTER the directory tape

The REVERT command may be called by any user after he has destroyed the text of his file on mass storage to make a previous backup become the primary copy. An automatically initiated LOAD of the primary backup copy text to mass storage occurs when the file is next assigned.

The user is responsible for verifying the present status of a file and its backup tapes. However, the necessary tools are at his disposal to warn him when an interrogation of his file should be made and to give him the complete picture of the alternatives available in terms of the status of the mass storage copy, if one exists, and all presently recorded backups.

For example, warning messages at assignment time signal when a file has been marked disabled. The user can then call @PRT,F (see 8.2.5) to get the status of the mass storage copy and all recorded backups. The Q option or an @ENABLE control statement (see 8.2.17) allows him to probe the mass storage copy and determine whether a REVERT to a previous copy is required. Until the user makes the decision of which copy of the file to retain as the primary copy, the SECURE processor guarantees to preserve all the information it can to give him the greatest possible choice of alternatives.

The third and most important mode of catalogued file recovery under SECURE involves using the MFD tape to recover all catalogued files, including tape files, into an essentially empty system. To initiate this process, perform a special REGISTER of the MFD tape. The SECURE processor copies this tape into a temporary mass storage file, catalogues all files not already catalogued, and remakes MFD items to mark files as unloaded to backup tape. As a result, the system is restored to a condition at least as current as the time the MFD tape was created. This avoids doing REGISTER's of the separate backup tapes. As users attempt to assign files, an executive function initiates an automatic LOAD of text back to mass storage.

19.11. SUMMARY OF SECURE PROCESSOR COMMANDS

Table 19–2 lists the name and function of each SECURE processor command.

Command	Description
CLEAR LAPSES	This command erases existing memory lapse entries for the set of files specified by the namelist.
LIST LAPSES	This command produces a listing of the memory lapse entries for all files or for the files specified by a namelist.
LIST REELS	Reserved for privileged runs, this command produces a summary listing of the current set of backups sorted in ascending reel number order.
LOAD	This command is reserved for privileged runs and causes the text of unload files given in a namelist to be retrieved from backup tape and written on mass storage.
LOAD...FROM IBACKUP	This command is reserved for privileged runs and allows the user to REGISTER files (see REGISTER commands) and to LOAD their text from tape all in one operation.
REGISTER DIRECTORY TAPE IBACKUP	Reserved for privileged runs, this command allows the user to REGISTER an entire system set of catalogued files using only the 'MFD tape' snapshot of the MFD.
REGISTER FROM IBACKUP	This command scans the set of backup tapes associated via source language with the particular IBACKUP unit and restores the MFD items of files found there which are not currently catalogued and whose complete backup copies reside on this reel set. Since only the items are restored with this command, files must be marked as unloaded to the backup tape. This command can also be used to restore the MFD items and granule tables for all files on a removable disc pack following an initial boot or when transferring the disc file information to a new site.

Table 19-2. Summary of SECURE Processor Commands (Part 1 of 2)

Command	Description
REMOVE	This command is reserved for privileged runs and causes all files specified in a namelist to be deleted from the system after first ensuring that a current backup exists.
REMOVE ALL	This command is identical to the REMOVE command except that a new backup copy is produced for each file to be deleted, whether one already existed or not.
REVERT	This command causes a file's backup copy on tape to become the primary copy by releasing its granules on mass storage and marking the file as unloaded.
SAVE	Without further qualification, this command applies to all files in the system not catalogued with a G option if the run is privileged, or to only those files with a project-id matching that of the calling run if the run is not privileged. With the above noted, the SAVE command causes new backups to made only for those files which do not have a current backup. Whether or not a backup is current depends on whether any write operations have been performed on the file since the time the last backup was made. In the case of privileged runs, the location of the new backup copy is recorded in the file's MFD items.
SAVE ALL	This is identical to the SAVE command except that a new backup copy is made regardless of whether a current backup exists. This allows the user to merge all his backup copies on a new, self-contained set of backup tapes. In the case of an unloaded file, SAVE ALL retrieves the text of the file from tape instead of mass storage.
UNLOAD	This command is reserved for privileged runs and without further qualification will not cause more files to be unloaded than is necessary to free up to 3000 tracks on mass storage. Files are chosen for unloading on the basis of their UEF (unload eligibility factor), which is computed by the SECURE processor. Files catalogued with a V option are not unloaded in any case. The UNLOAD command automatically implies SAVE; that is, a new backup copy is automatically produced, if required, before a file is marked as unloaded and its granules on mass storage are released.
UNLOAD TRACKS = nnnn	This command is identical to UNLOAD except that the preset limit of 3000 tracks is changed to some other value.
UNLOAD specific files, projects, or accounts	This command differs from the previous unload commands in that all files, projects, or accounts specified in the name list are unloaded regardless of the UEF.

Table 19-2. Summary of SECURE Processor Commands (Part 2 of 2)

19.12. EXAMPLES OF THE USE OF THE SECURE PROCESSOR

Example 1:

To make the set of backup tapes current (privileged):

```
      LABEL      Δ   OPERATION    Δ              OPERAND      Δ              COMMENTS
1              10            20              30            40             50
@RUN
@ASG,A      SYS$*DLOC$/RKEY/WKEY.
@ASG,TN     0BACKUP,C
@SECURE,IL  TPF$.DUMMY
    SAVE
@FIN
```

Example 2:

To produce backups of user's own files (nonprivileged):

```
      LABEL       Λ      OPERATION      Λ                    OPERAND      Λ                 COMMENTS
1               10                 20                   30                  40                  50
@RUN
@ASG,TN     ØBACKUP,C
@SECURE,IS  TPF$.DUMMY
    SAVE ALL FILES MY*FILE1, MY*FILE2
@FIN
```

Example 3:

To merge all backup copies on a single set of tapes (privileged):

```
      LABEL       Λ      OPERATION      Λ                    OPERAND      Λ                 COMMENTS
1               10                 20                   30                  40                  50
@RUN
@ASG,A      SYS$*DLØC$/RKEY/WKEY.
@ASG,TN     IBACKUP,C
@ASG,TN     ØBACKUP,C
@SECURE,IL  TPF$.DUMMY
    SAVE ALL
@FIN
```

Example 4:

To revert to the backup copy (nonprivileged):

```
      LABEL       Λ      OPERATION      Λ                    OPERAND      Λ                 COMMENTS
1               10                 20                   30                  40                  50
@RUN
@SECURE,IL  TPF$.DUMMY
    REVERT FILE ABC*XYZ
@FIN
```

Example 5:

To load the text of certain files (privileged):

```
@RUN
@ASG,A      SYS*DLOC$/RKEY/WKEY.
@ASG,TN     IBACKUP,C
@SECURE,IL  TPF$.DUMMY
    LOAD PROJECTS SATURN, JUPITER   ;
    EXCEPT ACCOUNT 423055
@FIN
```

Example 6:

To register a number of files from a particular backup tape set (nonprivileged):

```
@RUN
@ASG,TN     IBACKUP,C
@SECURE,IL  TPF$.DUMMY
    IBACKUP = 1201, 1202, 1203
    REGISTER PROJECT MY-OWN FROM IBACKUP
@FIN
```

Example 7:

To register an entire system set of files from the 'directory tape' following an initial boot (privileged):

```
@RUN
@ASG,A      SYS$*DLOC$/RKEY/WKEY.
@ASG,TN     IBACKUP,C
@SECURE,IL  TPF$.DUMMY
    IBACKUP = 1625C               . DIRECTORY TAPE
    REGISTER DIRECTORY TAPE FROM IBACKUP
@FIN
```

Example 8:

To register and load the text for a number of files from a particular backup tape set (privileged):

```
    LABEL      Δ   OPERATION      Δ             OPERAND       Δ              COMMENTS
1           10               20             30              40              50
@RUN
@ASG,A         SYS$*DLOC$/RKEY/WKEY.
@ASG,TN        IBACKUP,C
@SECURE,IL  TPF$.DUMMY
    IBACKUP = 1551, 1552, 1553
    LOAD FROM IBACKUP
@FIN
```

Example 9:

To restore all files on a removable disc pack following an initial boot (privileged):

```
    LABEL      Δ   OPERATION      Δ             OPERAND       Δ              COMMENTS
1           10               20             30              40              50
@RUN
@ASG,A         SYS$*DLOC$/RKEY/WKEY.
@ASG,T         IBACKUP, F17, PACKID
@SECURE,IL  TPF$.DUMMY
    IBACKUP = PACKID
    REGISTER FROM IBACKUP
@FIN
```

20. SYMBOLIC STREAM GENERATOR (SSG)

20.1. INTRODUCTION

The SSG processor generates any variety of symbolic streams, from a file of data to a run stream that configures an executive system. Directions and models for building the desired stream are conveyed to the SSG processor through a skeleton written in SYMSTREAM, an extensive manipulative language. Backus normal form notation is used throughout this section.

20.2. INPUT STREAMS

Five types of input streams are recognized by the SSG processor:

(1) permanent

(2) temporary

(3) stream generation statement

(4) skeleton

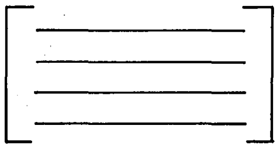
(5) corrections to skeleton streams

The permanent, temporary, and stream generation statement types can have any number of streams. All input streams except the skeleton stream, are optional. When an asterisk is in the first character position, a period terminates the scan of that image; a semicolon continues the scan to the first nonblank character of the next image.

□ Permanent stream

Any number of element/version entries and the corresponding images can be in the permament stream. An asterisk in the first character position and an alphabetic in the second character position defines an element/version entry; all images that follow, until the next entry, are attached to that element/version entry. The format of the element/version entry is as follows:

*<element name>[/<version name>]



where element and version names are 12 characters each. The images attached to an entry can be referenced in a SYMSTREAM program by the CORRECT directive (SYMSTREAM). Each element/version entry must be unique and may be accessed by using the stream generation statement (SGS) reference mechanism (see 20.7.1.4). P is a reserved label for the permanent stream:

[P] – Represents number of element/version entries.

[P,n] – 1 if element name only is present; 2 if element and version name are present.

[P,n,f] – 0

[P,n,1,1] – Represents element name

[P,n,2,1] – Represents version name

- Temporary stream

  Similar to permanent streams, except that the reserved label is T. The permanent and temporary images may have a common entry name between them.

- Stream generation statement (SGS) stream

  Stream generation statements are free form and may contain anything needed by a given skeleton. Any number of SGS's with a given label or any number of different labeled SGS's may exist. The label need not start in column 1. A period terminates the scan; a semicolon continues the scan to the first nonblank character of the next card image. The SGS scan ignores leading blanks, but interprets the first trailing comma as a subfield separator and the first trailing blank as a field separator.

- Skeleton stream

  The skeleton stream, written in SYMSTREAM, contains the directions and models for building the desired output stream images. The directives which cause SSG to perform the operations needed for generating the desired stream have the following format:

  *< directive name > b̶ < remaining image >

  Skeleton images without an asterisk (*) in the first character position are nondirective images and are sent to the output stream. All references in nondirective images must be bracketed. The string that satisfies the reference (up to the first blank character of the string) is substituted for that reference.

- Corrections to skeleton stream

  Corrections may be applied to the skeleton stream. The signal character for the line correction numbers is a plus sign (+).

20.3. OUTPUT STREAMS

Three types of output streams are generated by the SSG processor: revised temporary, revised skeleton, and the desired output stream. Only one revised temporary and one revised skeleton stream can be generated, but any number of desired output streams may be generated. All output streams are optional.

- Revised temporary stream

  By merging the permanent, temporary, and skeleton input streams, a revised temporary stream is generated, correction numbers revised according to the permanent and skeleton stream images.

- Revised skeleton stream

  By merging the skeleton stream and the corrections to skeleton stream, a revised skeleton stream is generated.

- Desired output stream

  Any number of output streams may be generated.

20.4. @SSG CONTROL STATEMENT

Format:

```
@SSG,options    param-1,param-2,param-3,param-4,param-5,param-6,param-7,param-8,...,param-n
```

All parameters are optional.

Parameters:

options — See Table 20-1.

param-1 — Specifies the skeleton stream.

param-2 — Specifies the SGS stream; or program file to have its element entries placed under the SGS label. Format of param-2 is then: < filename >./< label name >

Reference to element information is as follows:

[SGS label,n,1,1]	element name
[SGS label,n,2,1]	version name
[SGS label,n,3,1]	element type
[SGS label,n,3,2]	symbolic type if element type is symbolic

SGS integer references to element SGS's are allowed.

param-3 — Specifies the destination of desired output stream (must be a data file if any BRKPT directives are used).

param-4 — Specifies the destination of revised temporary stream.

param-5 — Specifies the destination of revised skeleton stream.

param-6 — Specifies the corrections to skeleton stream.

param-7 — The first of a variable number of input streams; specifies the type and number of the input stream. The following parameters contain the sources of these streams. This can be repeated any number of times. The format is as follows (must always start at param-7):

$$\left\{\begin{matrix}\text{PCF}\\ \text{TCF}\\ \text{SGS}\end{matrix}\right\}/<\text{number}>,\ \text{name-1},\ldots,\text{name-n},\ \left\{\begin{matrix}\text{PCF}\\ \text{TCF}\\ \text{SGS}\end{matrix}\right\}/<\text{number}>,\ \text{name-1},\ldots,\text{name-n}$$

Option	Description
A	Generates desired output stream regardless of no-find references.
B	Does not @ADD (see 3.9.1) the final output stream to the main run stream.
C	Double spaces all printing.
E	Prints revised skeleton stream.
F	Prints permanent streams.
G	Prints temporary streams.
H	Prints revised temporary stream.
I	Prints stream generation statement streams.
K	Prints desired output stream.
M	Debugs print — account of images in the skeleton being processed.
N	Debugs print — account of information on the directives being processed.

Table 20-1. @SSG Control Statement, Options

20.5. FILE IDENTIFICATION STATEMENTS

The input streams may be defined by file identification statements which are from card input and have the following format:

$$\left\{\begin{array}{l}\text{SGS}\\ \text{SKEL}\\ \text{PERM COR}\\ \text{TEMP COR}\\ \text{SKEL COR}\end{array}\right\}\quad [<\text{filename}>]$$

where filename represents the internal name of a data file.

The file-id statements can be used in conjunction with the parameters on the @SSG control statement. If the skeleton and corrections to skeleton streams are needed, each must be defined one way or the other, but not both. If the filename is not present, the input stream is from cards and must be terminated by an @EOF control statement (see 10.3.3). Any number of file-id statements are allowed and can be in any order. A period on a file-id statement terminates the scan of that image.

20.6. SUPPLEMENTARY INFORMATION

The debugging aids may be set initially on the @SSG control statement (M and N options) and turned on or off by setting or clearing their respective SSG defined global variables - MFLAG for M option and NFLAG for N option.

All input and output streams defined in the @SSG control statements are symbolic elements in program files or data files, unless otherwise noted.

Any number of desired output streams can be generated. However, only the last generated desired stream can be dynamically @ADDed (see 3.9.1) to the main run stream. When there is more than one desired stream, they must be data files.

20.7. FUNDAMENTALS OF SYMSTREAM

A skeleton, written in SYMSTREAM, instructs the SSG processor to produce the desired stream images. Two classes of images exist in the skeleton: directive and nondirective. Directive images perform all operations needed to generate the output stream. Nondirective images are sent to the output stream. The image determines the processor's mode of operation. Directive images are analyzed and executed in the processing mode; nondirective images are passed to the output stream in the output mode. An asterisk (*) in the first character position defines a directive image. The SSG processor dynamically moves between both modes.

20.7.1. ELEMENTS OF SYMSTREAM

The images and referencing techniques of SYMSTREAM are described in the following paragraphs. Unless otherwise stated, all referencing techniques use the left and right brackets to delimit the reference call.

20.7.1.1. VARIABLES

Two types of variables exist: local (increment index) and global. Local variables are created by the *INCREMENT directive and exist as long as the increment-loop block exists for that variable. Global variables are created by the *SET and *CLEAR directives and exist for the remaining processing of the skeleton. A variable must be created before it can be referenced. A reference to the variable, EXAM, would be [*EXAM]. For certain directives, a numeric satisfaction of a variable is expected. The reference then may be EXAM. The priority for satisfying the variable is: take the last created local variable of that name (regardless of the presence of other local variables of the same name); if none exists, try to satisfy from global variables; if no global variable exists with that name, a no-find message is printed and generation is terminated (unless the A option is on).

20.7.1.2. PROCESS PARAMETERS

Process parameters are created on the call line to a named predefined sequence of images in the skeleton. A reference to the third parameter on the last call line would be [#3]. A reference to an upper nested call needs the name of the called sequence of images. The reference [#3,DEFS] would pick up the third parameter of the last call on DEFS (any number of nested calls on the same sequence is allowed). If such a parameter does not exist, a no-find message is printed and generation is terminated (unless the A option is on).

20.7.1.3. INTEGER EXPRESSIONS

Integer expressions include explicit numbers, variables (either bracket or nonbracket references), and process parameters (with satisfaction being either a number or variable name).

Example:

```
23+[*EXAM]–CUR+[# 3,DEFS].
```

20.7.1.4. STREAM GENERATION STATEMENTS

External parameters, switches for alternate processing paths and models, lists or tables of data to be operated upon, and so forth, are provided to the skeleton through stream generation statements (SGS). The SGS's are defined at the processor call level or dynamically created or removed during skeleton processing.

Format:

```
label   param-11[,param-12,...,param-1n]  [param-21,param-22,...,param-2n]...
```

An SGS may contain any number of subfields in any number of fields. The label field cannot be referenced as a subfield. Fields are separated by blanks; subfields are separated by commas. Any number of SGS images with the same label and any number of different labeled SGS's may exist.

The following types of references to SGS's are allowed:

[lab] – Represents the number of SGS's with the label LAB.

[lab,n] – Represents the number of fields on the nth SGS with the label LAB.

[lab,n,f] – Represents the number of subfields in the fth field of the nth SGS with the label LAB.

[lab,n,f,s] – Represents the contents of the sth subfield of the fth field of the nth SGS with the label LAB.

20.7.1.5. NUMERIC EXPRESSIONS

Combinations of integer expressions and SGS references form numeric expressions. These combinations include both elemental references used in integer expressions and SGS references (with satisfaction being either a number or a variable name).

Example:

```
23+[*EXAM]–[#3,DEFS]+[LAB]–[LAB,[#1],3].
```

20.7.1.6. SUPPLEMENTARY BASICS

Any character (including blanks) is allowed in a given string by using quotation marks (".....") which designate a Fieldata string including all characters, excluding the quotation marks. Single quote marks ('.....') designate a Fieldata string which includes all the characters and quotation marks.

NOTE: For SGS's, the sixth, twelfth, and eighteenth characters cannot be blanks.

Reserved SSG words are either explicitly stated or process parameters. All programmer-supplied words or values are explicitly stated, process parameters, variable references, or SGS references. Referencing in the output mode for nondirective images must be bracketed references.

If either # or ¤ are explicit in the first character position of a nondirective image, the output stream contains either @ or *, respectively, in the first character position.

20.7.2. SYNTAX OF SYMSTREAM

An asterisk (*) in the first character position of the skeleton image places the SSG processor in the processing mode. In this mode, the directive image is both analyzed and executed. The directives are:

*BRKPT
*CLEAR
*CORRECT
*CREATE
*DEFINE
*DIVIDE
*EDIT
*ELSE
*END
*IF
*INCREMENT
*LOOP
*MULTIPLY
*PROCESS
*REMOVE
*SET

The following conventions are used in the syntax descriptions of the directives:

(1) b̸ signifies one or more mandatory blanks.

(2) Items enclosed in brackets ([ ]) are optional (not to be confused with bracketed references).

(3) Capitalized parameters are reserved SSG words.

(4) Lowercase parameters are filled in by the programmer.

(5) Braces ({ }) designate a choice of terms.

20.7.2.1. GENERATING OUTPUT STREAMS (*BRKPT)

Purpose:

Generates any number of output streams.

Format:

$$\text{*BRKPT [,K] ƀ} \left[\begin{Bmatrix}\text{<SGS-reference>}\\ \text{<process-parameter>}\\ \text{<string>}\end{Bmatrix}\right]$$

Description:

The K option prints the new output stream. If the external stream is specified on the *BRKPT directive, it must be the name of a data file. The previous output stream is handled according to its initial options. The B option on the @SSG control statement (see 20.4) affects only the last output stream; the other output streams cannot be dynamically executed.

Example:

LABEL 1	10 OPERATION	20	30 OPERAND	40	COMMENTS 50
*BRKPT,K	FILEA				

This directive generates and prints an external stream of the data file named FILEA.

20.7.2.2. ZEROING EXISTING AND CREATED VARIABLES (*CLEAR)

Purpose:

Zeros the value of an existing variable or creates a new variable, if none exists by that name, with the value of zero.

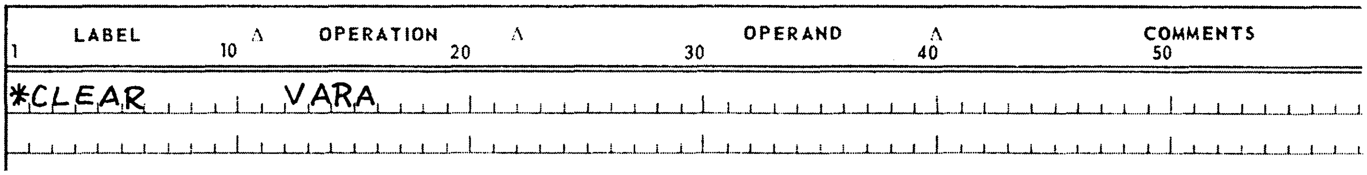
Format:

$$\text{*CLEAR} \begin{Bmatrix}\text{ƀ<SGS-reference>}\\ \text{ƀ<process-parameter>}\\ \text{ƀ<string>}\end{Bmatrix}$$

Description:

The last local variable of the given name is set to zero. If no local variables exist, the global variable of the given name is set to zero. If the global variable does not exist, a global variable of the given name is created with the value of zero. This global variable exists for the remainder of the skeleton processing.

Example:



```
LABEL      OPERATION      OPERAND      COMMENTS
*CLEAR     VARA
```

If VARA is a previously defined local or global variable, this directive zeros the existing value. If VARA did not exist, it is created in the form of a global variable with the value of zero.

20.7.2.3. MERGING INPUT AND SKELETON STREAMS (*CORRECT AND *END)

Purpose:

Selectively merges input streams with the skeleton stream and passes the resultant merge to the output stream.

Format:

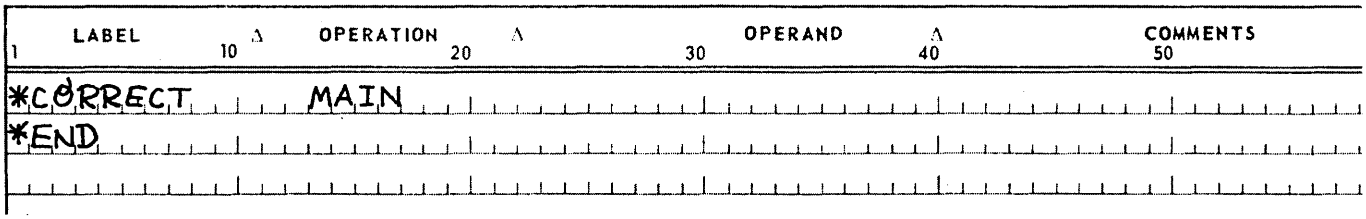
$$\text{*CORRECT } \not b \left\{\begin{array}{l}\text{<SGS-reference>}\\ \text{<process-parameter>}\\ \text{<string>}\end{array}\right\} \left[/ \left\{\begin{array}{l}\text{<SGS-reference>}\\ \text{<process-parameter>}\\ \text{<string>}\end{array}\right\}\right] \; \not b \; \left[\begin{array}{l}\text{PERM}\\ \text{TEMP}\end{array}\right]$$

*END

Description:

The given element/version has its specified input stream merged with the following nondirective skeleton images. If no input stream is specified, both the PERM and TEMP streams are merged with the skeleton images. If the given element/version is not present in a stream to be in the merge, the merge consists of the remaining streams. Any number of merges can be performed for the given element/version. Conflicts in line numbers are flagged, and the stream images are sent to the output stream or bypassed according to the following priority: TEMP images override PERM images which override skeleton images.

Example:



```
LABEL      OPERATION      OPERAND      COMMENTS
*CORRECT   MAIN
*END
```

These directives merge all correction images for MAIN and place them in the output streams following the last record.

20.7.2.4. DYNAMIC EXPANSION OF SGS's OR PERM AND TEMP CHAINS (*CREATE)

Purpose:

Dynamically expands SGS's or PERM and TEMP chains of element/version names for given input streams.

Format 1:

*CREATE ƀ SGS:ƀ < any-number-of-strings-and-bracket-references >

Format 2:

$$\text{*CREATE ƀ}\left\{\begin{array}{l}\text{PERM}\\ \text{TERM}\\ \text{TEMP}\end{array}\right\}\text{;ƀ}\left\{\begin{array}{l}\text{<SGS-reference>}\\ \text{<process-parameter>}\\ \text{<string>}\end{array}\right\}\left[\text{/}\left\{\begin{array}{l}\text{<SGS-reference>}\\ \text{<process-parameter>}\\ \text{<string>}\end{array}\right\}\right]$$

Description:

If the edit mode is on, it is turned off. After the colon, any number of strings and bracketed references can be used to create the secondary image. The original and secondary images can be no longer than one card image each. For the *CREATE SGS, the secondary image is a complete SGS and can be referenced as such (with label, image number, fields, and subfields as placed in the secondary image). For the *CREATE PERM or TEMP, an element/version name entry is made in the chain for the given input stream. The IF directive's test for input stream produces a Boolean false for such an entry, since no input stream image is attached to it.

Examples:

	LABEL	OPERATION	OPERAND	COMMENTS
	1	10 20	30 40	50
1.	*CREATE	SGS:	LAB HAR,3	
2.	*CREATE	PERM:	ELEM/[#1,VERN]	

1. Generates an SGS with a label of LAB. The first field has subfield one as HAR and subfield two as 3.

2. The element, ELEM/[#1,VERSN], is added to the PERM stream.

20.7.2.5. DEFINING SKELETON IMAGE SEQUENCES (*DEFINE AND *END)

Purpose:

Defines a sequence of skeleton images which later may be called upon (*PROCESS) and provided parameters.

Format:

$$\text{*DEFINEƀ}\left\{\begin{array}{l}\text{<SGS-reference>}\\ \text{<process-parameter>}\\ \text{<string>}\end{array}\right\}$$

$$\left[\begin{array}{l}\text{―――――}\\ \text{―――――}\\ \text{―――――}\end{array}\right]$$

*END

Description:

A sequence of skeleton images must be defined before it is called. The parameters generated by the call are referenced within the sequence of images by: [#<number>] ; parameter of higher nested calls are referenced by: [#<number>,<name>]. Nested calls on other predefined sequences and recursive calls upon itself is allowed to any depth.

Example:

See the example given for the *PROCESS directive (see 20.7.2.11).

20.7.2.6. VARIABLE DIVISION (*DIVIDE)

Purpose:

Divides variables.

Format:

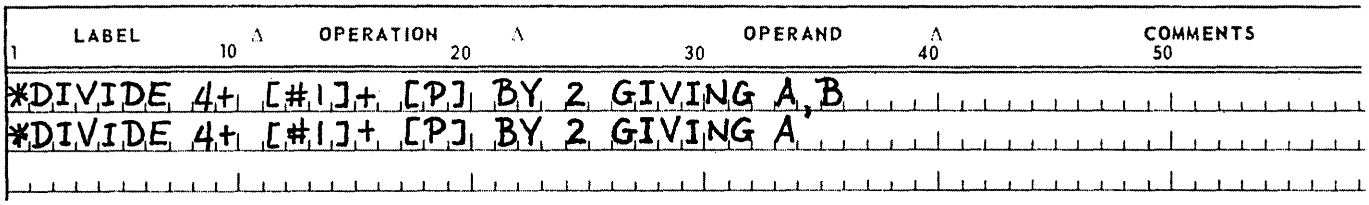
*DIVIDEb̄<numeric-expression>b̄BYb̄<numeric-expression>b̄

$$\text{GIVING }\bar{b}\left\{\begin{array}{l}\text{<SGS-reference>}\\\text{<process-parameter>}\\\text{<string>}\end{array}\right\}\left[,\left\{\begin{array}{l}\text{<SGS-reference>}\\\text{< process-parameter>}\\\text{<string>}\end{array}\right\}\right]$$

Description:

The quotient may be either a global or local variable. The remainder, if desired, is a different variable.

Example:


LABEL
OPERATION
OPERAND
COMMENTS
1
10
20
30
40
50

```
*DIVIDE 4+ [#1]+ [P] BY 2 GIVING A,B
*DIVIDE 4+ [#1]+ [P] BY 2 GIVING A
```

If [* 1] equals 10 and [P] equals 10, then A equals 12 and B equals 0.

20.7.2.7. OUTPUTTING NONDIRECTIVE SKELETON IMAGES AS ONE IMAGE (*EDIT)

Purpose:

Outputs any number of nondirective skeleton images as one image.

Format:

$$\text{*EDIT }\bar{b}\left\{\begin{array}{l}\text{ON}\\\text{OFF}\end{array}\right\}\bar{b}\left[\left\{\begin{array}{l}\text{<SGS-reference>}\\\text{<process-parameter>}\\\text{<string>}\end{array}\right\}\right]$$

Description:

An ampersand (&) is assumed as the edit symbol if none is specified. When edit mode is on, all output images which are terminated by the edit symbol are joined together (a semicolon is inserted if the image needs to continue on the next card. Edit mode is terminated by being specified or after the occurrence of the first nondirective skeleton image which has no edit symbol (when the edit mode is on).

Example:

```
 LABEL     Δ  OPERATION   Δ          OPERAND    Δ          COMMENTS
1        10          20          30          40          50
*EDIT ØN &
  HAR SAR [#1] &
 PAR LAR [#2] &
*EDIT ØFF
```

These lines in the skeleton generate the following image if [#1] is A1 and [#2] is A2:

--HAR--SAR--A1----PAR--LAR--A2--

20.7.2.8. SKIPPING SKELETON IMAGES (*IF, *ELSE, AND *END)

Purpose:

Conditionally tests and skips a sequence of skeleton images.

Format:

```
*IF b̸ <Boolean-expression>

[*ELSE]

*END
```

Description:

Table 20-2 gives the conditional paths available.

	BOOLEAN EXPRESSION	
	TRUE	FALSE
*ELSE PRESENT	Processes images between *IF and *ELSE; skips images between *ELSE and *END	Skips images between *IF and *ELSE; processes images between *ELSE and *END.
NO *ELSE	Processes images between *IF and *END.	Skips images between *IF and *END.

Table 20-2. IF Directive, Conditional Paths

The operands allowed to form Boolean expressions are explained in the following paragraphs.

Either global or local variables may be tested for zero (*CLEAR) or nonzero (*SET). The format is:

$$\left\{\begin{array}{l}\text{<SGS-reference>}\\ \text{<process-parameter>}\\ \text{<string>}\end{array}\right\} \text{ ƀ IS ƀ } \left\{\begin{array}{l}\text{SET}\\ \text{CLEAR}\end{array}\right\} \text{ ƀ}$$

The value of a numeric expression (a plus or minus sign causes evaluation of a numeric expression) may be used in relational tests. The format is:

VALUE ƀOF ƀ numeric-expression

$\left\{ + \right\}$ numeric-expression

$$\left\{\begin{array}{l}\text{< SGS-reference >}\\ \text{< process-parameter >}\\ \text{< variable (bracketed) >}\\ \text{< string > ƀ}\\ \text{value-of-numeric-expression}\end{array}\right\} \left[\text{< relation >} \left\{\begin{array}{l}\text{< SGS-reference >}\\ \text{< process-parameter >}\\ \text{< variable (bracketed) >}\\ \text{< string >}\\ \text{value-of-numeric-expression}\end{array}\right\} \right]$$

SGS's are two-dimensional tables. A row or column search starting from a given label at a given row (a particular image of that label) and a column (a particular field and subfield of that label) attempts to search to the end of that row or column, respectively. A find, on a row search, gives the column of the find. The SSG-defined global variables, FLD and SFLD, then contain the field and subfield (column) of the find. A find, on a column search, gives the row of the find. The SSG-defined global variable, CARD, then contains the image number of that label (row) of the find. If a starting value is not specified, one (1) is assumed. The format for specifying starting values is: LABEL,ROW (image number), COLUMN (field, subfield).

The search format is:

$$\left\{\begin{array}{l}\text{ROW}\\ \text{COLUMN}\end{array}\right\} \text{ ƀ SEARCH ƀ FROM ƀ } \left\{\begin{array}{l}\text{<SGS-reference>}\\ \text{<process-parameter>}\\ \text{<string>}\end{array}\right\} \left[\text{ ,<numeric-expression>} \right.$$

$$\left. \left[\text{,<numeric-expression>} \left[\text{,<numeric-expression>} \right] \right] \right] \text{ ƀFORƀ } \left\{\begin{array}{l}\text{<SGS-reference>}\\ \text{<process-parameter>}\\ \text{<variable (bracketed)>}\\ \text{<string>}\\ \text{value-of-numeric-expression}\end{array}\right\}$$

Two input streams (permanent and temporary) can be merged. A search is performed for the given element/version in either the PERM or TEMP stream. If neither stream is specified, both are searched. The format is:

$$\left\{\begin{array}{l}\text{<SGS-reference>}\\ \text{<process-parameter>}\\ \text{<string>}\end{array}\right\} \left[\text{ / } \left\{\begin{array}{l}\text{<SGS-reference>}\\ \text{<process-parameter>}\\ \text{<string>}\end{array}\right\} \right] \text{ ƀ HAS ƀ } \left[\left\{\begin{array}{l}\text{PERM}\\ \text{TEMP}\end{array}\right\} \right] \text{ ƀ CORRECTIONS}$$

Examples:

```
   LABEL      Δ   OPERATION    Δ              OPERAND        Δ            COMMENTS
1         10              20             30              40              50
*IF B IS SET OR VALUE OF A = -1

```

If B is nonzero (SET) and/or A is -1, this expression is true.

```

*IF ROW SEARCH FROM SGSL, 1,A,B+1 FOR HARI

```

If the following SGS exists:

```
SGSL   LAR,SAR   HAR1,HAR2
```

and A=1 and B=1, then this expression is true and the reserved variable FLD equals 2, and the variable SFLD equals 1. These variables represent the field and subfield where the word HAR1 was found.

20.7.2.9. SKELETON IMAGE LOOPS (*INCREMENT AND *LOOP)

Purpose:

Increments and loops through a sequence of skeleton images.

Format:

```
            ( <SGS-reference>     )   [                          ]   [
*INCREMENTƀ < <process-parameter> > ƀ [ FROM ƀ <numeric-expression>] ƀ [ TO ƀ
            ( <string>            )   [                          ]   [

                       ]   [                          ]   [          ( <SGS-reference>     )
<numeric-expression>   ] ƀ [ BYƀ<numeric-expression>  ] ƀ [ WHILE ƀ  < <process-parameter> > ƀ IS ƀ  >
                       ]   [                          ]   [          ( <string>            )

{ SET   } ]
{ CLEAR } ] ƀ

{ ______ }
{ ______ }
{ ______ }
{ ______ }
*LOOP
```

Description:

Local variables (increment index) are created by the INCREMENT directive and exist as long as incrementing is neeced. Any number of local variables and a global variable may have the same name. The FROM, TO, BY, and WHILE phrases may be in any order or absent. When absent, the BY, TO, and FROM values are assumed to be one (1). The WHILE phrase tests the given variable (global or local) for zero (CLEAR) or nonzero (SET). If the phrase is true, the index is incremented (except the first time). The following algorithm is performed: (TO value - present index) (BY value). If the result is negative or if the WHILE phrase is false, the INCREMENT-LOOP block and its increment index are destroyed. If positive, processing continues until the TO value is satisfied.

Example:

```
    LABEL      Δ   OPERATION   Δ           OPERAND      Δ               COMMENTS
1          10            20            30            40             50
*INCREMENT A FRØM -10 TØ 5 BY 5
*INCREMENT B FRØM 0 BY 2 WHILE A IS SET
[*A],[*B]
*LØØP
*LØØP
```

This produces the ordered pairs:

−10,0   −5,0   5,0

20.7.2.10. VARIABLE MULTIPLICATION (*MULTIPLY)

Purpose:

Multiplies variables.

Format:

$$\text{*MULTIPLY ƀ <numeric-expression> ƀ BY ƀ <numeric-expression> ƀ}$$

$$\text{GIVING ƀ}\left\{\begin{array}{l}\text{<SGS-reference>}\\ \text{<process-parameter>}\\ \text{<string>}\end{array}\right\}$$

Description:

The product can be either a global or local variable.

Example:

```
    LABEL          Δ      OPERATION     Δ                 OPERAND      Δ                 COMMENTS
1              10                   20              30                 40                 50
*MULTIPLY [#1]+ 9 BY 2 GIVING VARA
```

If[# 1] equals 1, VARA equals 20.

20.7.2.11. CALLING A PREDEFINED SEQUENCE OF SKELETON IMAGES (*PROCESS)

Purpose:

Calls a predefined sequence of skeleton images and provides parameters to that sequence.

Format:

$$\text{*PROCESS ƀ}\left\{\begin{array}{l}\text{<SGS-reference>}\\ \text{<process-parameter>}\\ \text{<string>}\end{array}\right\}\text{ƀ}\left[\text{defenition-of-process-parameters}\right]$$

Description:

An extensive repertoire of parameter generation is available to the PROCESS directive. The following syntax is allowed for each parameter, each of which is separated by at least one blank:

$$\left\{\begin{array}{l}\text{<SGS-reference>}\\ \text{<process-parameter>}\\ \text{<variable(bracketed)>}\\ \text{<string>}\\ \text{** <numeric-expression>}\\ \left\{\pm\right\}\text{<numeric-expression>}\\ *\left\{\begin{array}{l}\text{<SGS-reference>}\\ \text{<process-parameter>}\end{array}\right\}\end{array}\right\}\text{ƀ}$$

The presence of a double asterisk (**), plus, or minus sign, triggers the evaluation of a numeric expression. A single asterisk followed by an SGS or process parameter reference directs the evaulation of that string that satisfied the reference. This string can contain any of the features for parameter generation allowed in the PROCESS directive. For indirect parameter generation, the string must be defined by double quotation marks if it contains a space, period, semicolon, comma, left bracket, slash, plus, minus, or right bracket.

Example:

```
LABEL            OPERATION                OPERAND                 COMMENTS
1          10              20          30            40           50
*DEFINE   MAIN
#ASM,SU   [#1],[#2]
*END
*PROCESS  MAIN  EL1  EL2
```

This skeleton generates the following image in the output stream:

```
@ASM,SU  EL1,EL2
```

20.7.2.12. DELETING SGS'S, AND PERM AND TEMP ELEMENT/VERSION NAMES (*REMOVE)

Purpose:

Removes SGS's and PERM and TEMP element/version name entries for a given input stream.

Format 1:

```
                  ⎧ <SGS-reference>     ⎫ ⎡                        ⎡                        ⎤⎤
*REMOVEƀSGS ƀ     ⎨ <process-parameter> ⎬ ⎢ ,<numeric-expression>  ⎢,<numeric-expression>   ⎥⎥
                  ⎩ <string>            ⎭ ⎣                        ⎣                        ⎦⎦
```

Format 2:

```
           ⎧ PERM ⎫    ⎧ <SGS-reference>     ⎫ ⎡  ⎧ <SGS-reference>     ⎫⎤
*REMOVE ƀ  ⎨      ⎬ ƀ  ⎨ <process-parameter> ⎬ ⎢ /⎨ <process-parameter> ⎬⎥
           ⎩ TEMP ⎭    ⎩ <string>            ⎭ ⎣  ⎩ <string>            ⎭⎦
```

Description:

For SGS removal, any number of given labels may be removed. The format specifying the starting image is: label, image number, and number of images to be removed. If the numbers are not specified, 1 is assumed. For the PERM and TEMP removal, the element/version name entry is removed from the chain for the given input stream. For the test of input streams on the IF directive, a removed element/version entry gives a Boolean value of false.

Example:

```
       LABEL         Λ     OPERATION      Λ                OPERAND        Λ               COMMENTS
1                 10                 20               30               40               50
*REMOVE SGS EXAM,[*B]
```

If [*B] equals 3, three SGS's with the common name EXAM are removed from the input stream.

20.7.2.13. CHANGING EXISTING OR CREATED VARIABLES (*SET)

Purpose:

Changes the value of an existing variable or creates a new variable if none exists by that name.

Format:

$$\text{*SET } \flat \left\{ \begin{array}{l} \text{<SGS-reference>} \\ \text{<process-parameter>} \\ \text{<string>} \end{array} \right\} \flat \left[\text{TO } \flat \text{ <numeric-expression>} \right]$$

Description:

If the TO phrase is omitted, 1 is assumed. The last local variable of the given name is changed. If it does not exist, the global variable of the given name is changed. If it does not exist, a global variable of the given name is created with the given (or assumed) value. This global variable exists for the remaining processing of the skeleton.

Examples:

```
*SET A
```

SSG looks for an increment index named A. If none exists, it looks for a global variable named A. If non exists, a global variable named A is created.

This variable is set to 1.

```
*SET B TO A+9-[#1]
```

If A equals 1 and [#1] is Q which equals –4, then B is sought as in the previous example and set equal to 14.

20.8. EXAMPLES OF SSG STREAM GENERATION

The following examples clarify the use of the SSG processor.

(1) On the @SSG control statement (see 20.4), the first specification field (param-7) for the variable parameter fields is divided into two parts. The first part must be either PCF, TCF, or SGS; the second part must be a number n 0. The following n parameter fields designate the source of the input streams.

Example 1:

```
TCF/4, TFL1.,TFL2.,TFL3.,TFL4.
```

TFL1, TFL2, TFL3, TFL4 are four data files, containing temporary corrections, assigned to the runs on either tape or mass storage.

Example 2:

```
PCF/2, PF.ELT1,PF.ELT2
```

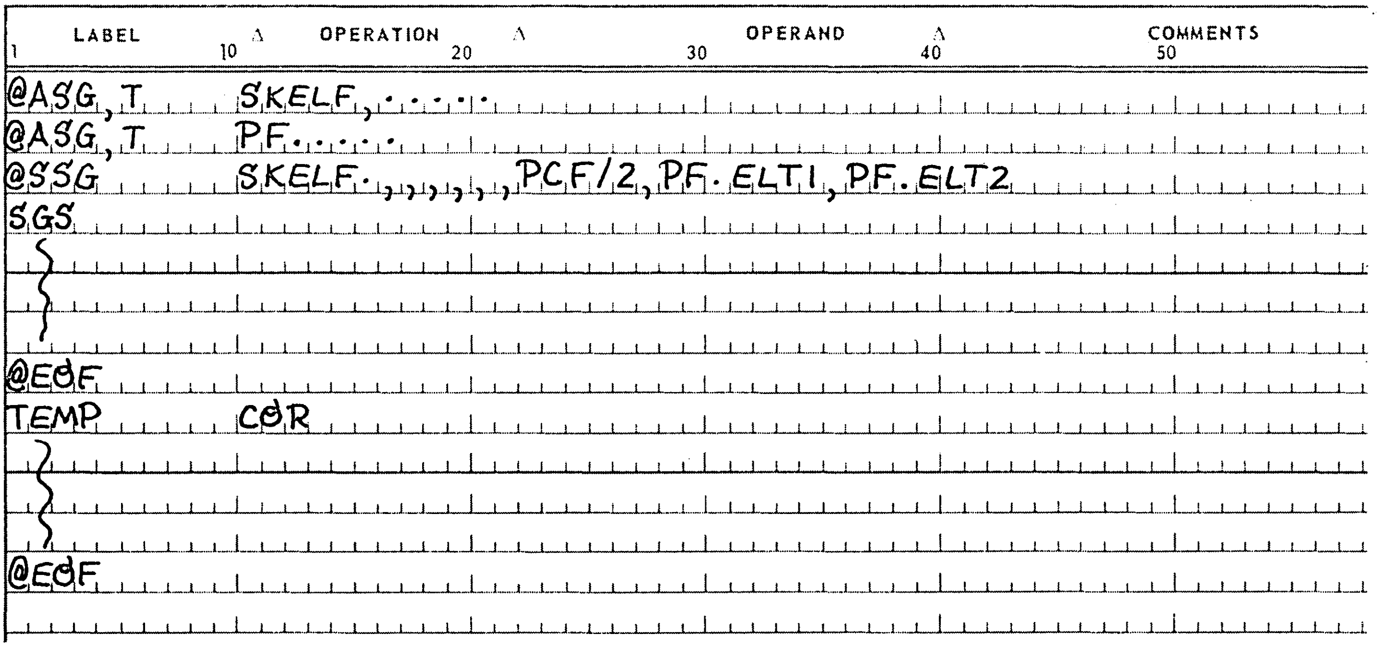
PF.ELT1 and PF.ELT2 are elements containing permanent corrections within the program file PF.

Example 3:

```
SGS/1, CONFD.,PCF/2,FILE1.,FILE2.
```

There is one file of SGS's called CONFD and two data files containing permanent corrections (FILE1, FILE2).

(2) Assume that an SSG run has SGS on cards and two sets of permanent corrections (ELT1 and ELT2), both in program file PF. Further assume that temporary corrections are on cards, and the skeleton is in a file called SKELF in SDF format. The following run stream could be used to execute SSG:



```
LABEL     ∧  OPERATION   ∧             OPERAND   ∧              COMMENTS
1         10            20            30            40            50
@ASG,T    SKELF,.....
@ASG,T    PF.....
@SSG      SKELF.,,,,,,PCF/2,PF.ELT1,PF.ELT2
SGS
 {
 {
 {
@EOF
TEMP      COR
 {
 {
 {
@EOF
```

The following run stream could also be used:

```
     LABEL       Δ     OPERATION      Δ                 OPERAND        Δ                COMMENTS
1            10                  20                30                40                50
@ASG,T    SKELF,....
@ASG,T    PF.......
@SSG      ,,,,,,PCF/2,PF.ELT,PF.ELT2
 SGS
  }
  }
  }
@EOF
SKEL      SKELF
TEMP      COR
  }
  }
  }
@EOF
```

(3) This sample skeleton changes the sequence numbers existing in columns 73-80 of a FORTRAN source element to a new sequence set. The program is compiled and uses the special correction feature of the source input routine (see 9.6) to apply corrections.

The desired run stream is:

```
     LABEL       Δ     OPERATION      Δ                 OPERAND        Δ                COMMENTS
1            10                  20                30                40                50
@FOR,U    A,B
-1,1000-
73/00000000
73/00000100
:
73/00099900
```

The skeleton must generate these sequence numbers.

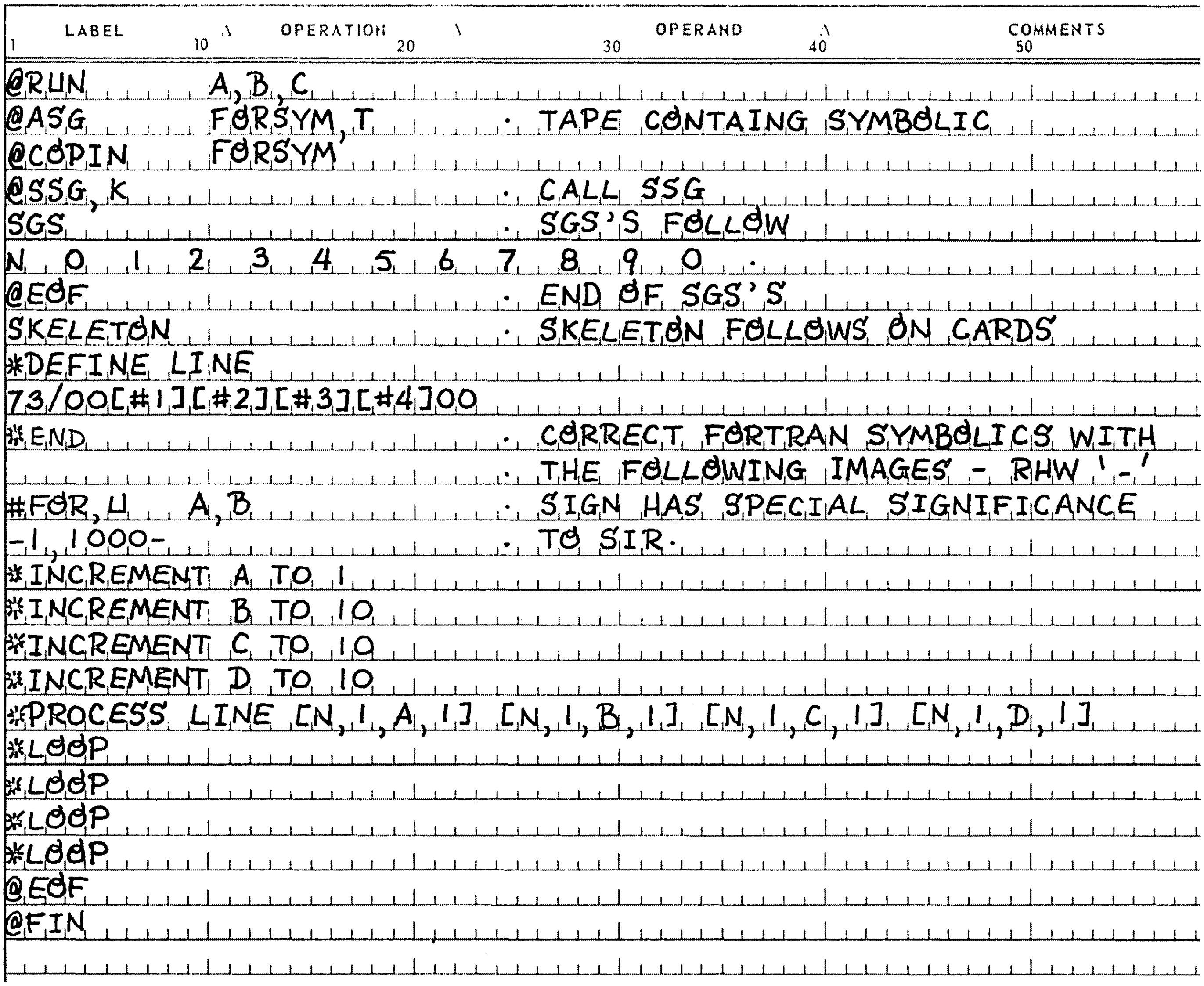
Only one SGS is required and contains the number to be inserted for the sequence numbers. The output image (the referenced process parameters are substituted) is defined for later use. The skeleton images

FOR,U A,B

−1,1000–

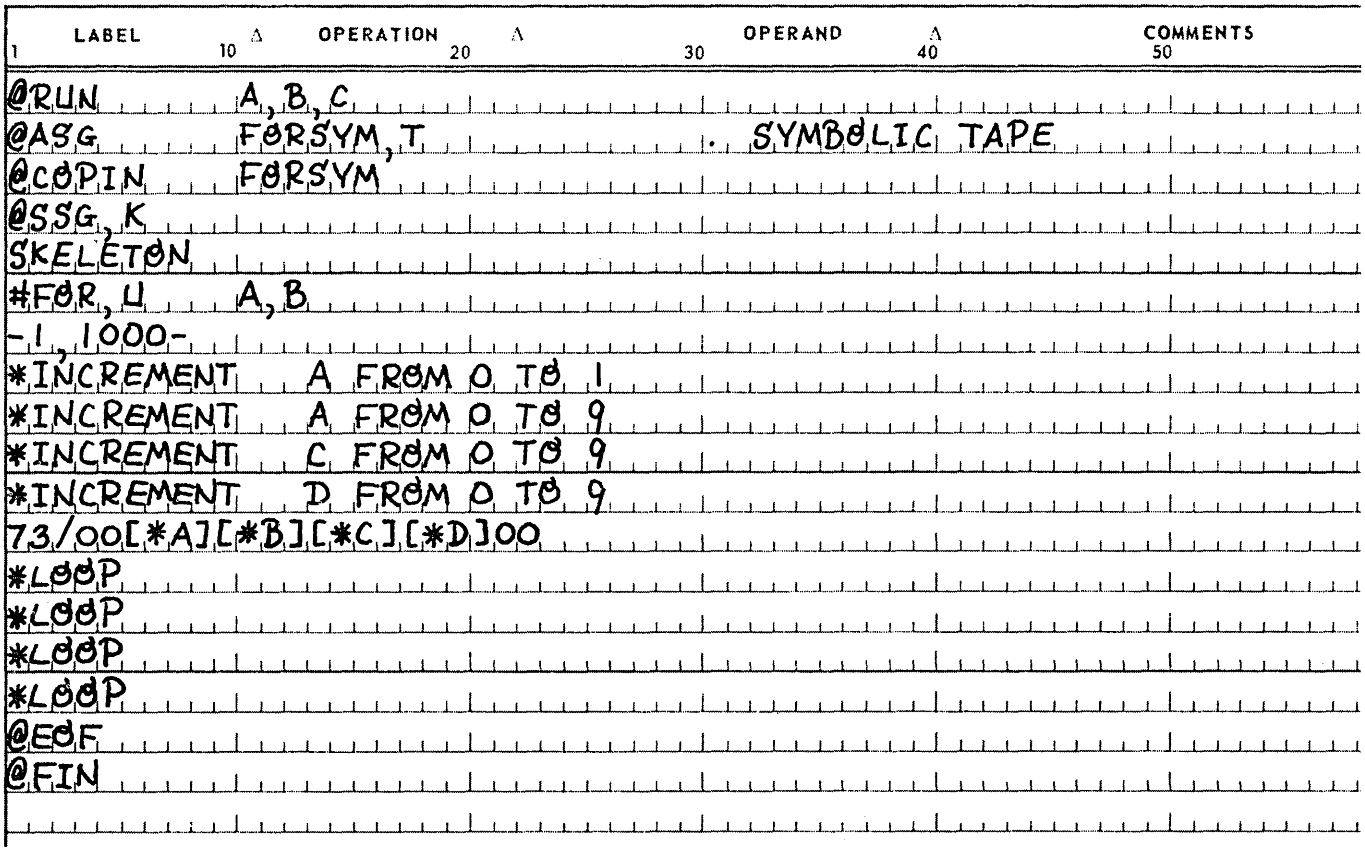
are passed to the run stream with @ substituted for # .

The following run stream could be used to accomplish this:



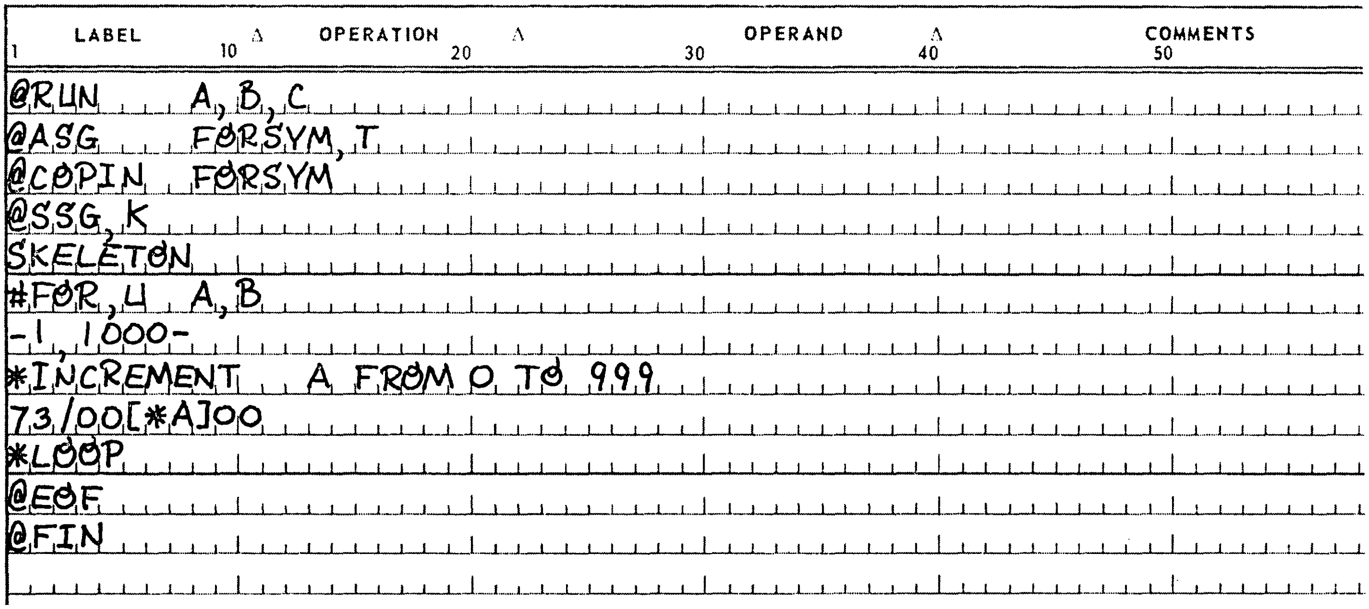
```
LABEL       OPERATION          OPERAND           COMMENTS
1        10        20        30        40        50
@RUN      A,B,C
@ASG      FORSYM,T      . TAPE CONTAING SYMBOLIC
@COPIN    FORSYM
@SSG,K                  . CALL SSG
SGS                     . SGS'S FOLLOW
N  0 1  2  3  4  5  6  7  8  9  0  .
@EOF                    . END OF SGS'S
SKELETON                . SKELETON FOLLOWS ON CARDS
*DEFINE LINE
73/00[#1][#2][#3][#4]00
*END                    . CORRECT FORTRAN SYMBOLICS WITH
                        . THE FOLLOWING IMAGES - RHW '-'
#FOR,U  A,B             . SIGN HAS SPECIAL SIGNIFICANCE
-1,1000-                . TO SIR.
*INCREMENT A TO 1
*INCREMENT B TO 10
*INCREMENT C TO 10
*INCREMENT D TO 10
*PROCESS LINE [N,1,A,1] [N,1,B,1] [N,1,C,1] [N,1,D,1]
*LOOP
*LOOP
*LOOP
*LOOP
@EOF
@FIN
```

Another way of accomplishing the same result would be:



```
LABEL      OPERATION                 OPERAND                  COMMENTS
1        10        20             30         40               50
@RUN      A,B,C
@ASG      FØRSYM,T                . SYMBØLIC TAPE
@CØPIN    FØRSYM
@SSG,K
SKELETØN
#FØR,U    A,B
-1,1000-
*INCREMENT  A FRØM 0 TØ 1
*INCREMENT  A FRØM 0 TØ 9
*INCREMENT  C FRØM 0 TØ 9
*INCREMENT  D FRØM 0 TØ 9
73/00[*A][*B][*C][*D]00
*LØØP
*LØØP
*LØØP
*LØØP
@EØF
@FIN
```

If leading zeros are not required, the following could be written:



```
LABEL      OPERATION                 OPERAND                  COMMENTS
1        10        20             30         40               50
@RUN   A,B,C
@ASG   FØRSYM,T
@CØPIN FØRSYM
@SSG,K
SKELETØN
#FØR,U A,B
-1,1000-
*INCREMENT  A FRØM 0 TØ 999
73/00[*A]00
*LØØP
@EØF
@FIN
```

(4) The following example manipulates the permanent (PCF) and temporary (TCF) correction files. The PCF is contained on mass storage; the TCF is from card images. In the example, the basic problem is to correct existing images and insert new ones.

The three files are:

PCF	TCF	SGS
*A@ . Lowest possible name		INSERT COMPD
*COMPA	*COMPA	
———	–0,0/2,2 . replace 2nd image	
———*	——— . with the following	
———		
*COMPB	*COMPD	
———	———	
———	———	
———	———	
.		
.		
.		
*Z . Highest possible name		

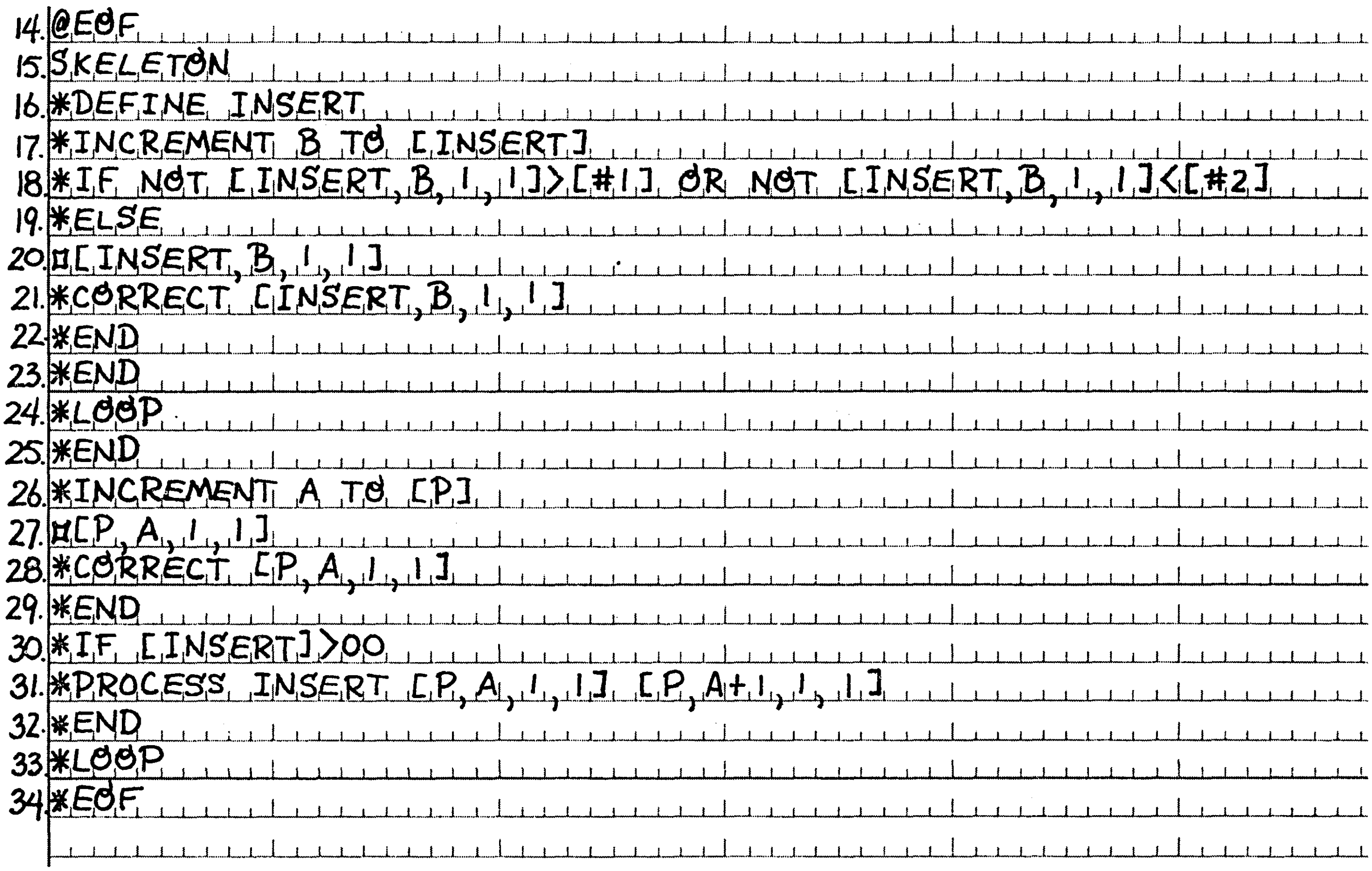
The PCF contains data, identically formatted, on each company in a city.

Assume that company A (COMPA) has an address change and a new company (COMPD) is to be inserted in the PCF from the TCF. An SGS is required for the insert to take place. Each new company has its name on a separate INSERT SGS card (INSERT COMPD).

An insert process, to insert new companies into the run stream (the new permanent correction data file), is defined. If any INSERT SGS's are present, a loop determines which of the new companies is to be inserted at this point (the data file is in alphabetical order). If a company is to be inserted, it is inserted at this point. The skeleton images which determine the find and insertion are lines 16 through 25 in the following run stream.

To generate the new data file, a loop is made through the present companies, and each company is corrected. A determination of new companies to be inserted is performed after each present company is updated. The skeleton images are lines 26 through 34 in the following run stream.

```
     LABEL         OPERATION                    OPERAND              COMMENTS
    1         10               20           30            40          50
 1. @RUN      A,B,B
 2. @ASG      PCF,F2
 3. @ASG      NEWPCF,T
 4. @SSG,KB
 5. PERMANENT CORRECTIONS PCF
 6. TEMPORARY CORRECTIONS
 7. *COMPA
 8. 0,0/2,2
 9. ~~~~~~~~~~
10. *COMPD
11. ~~~~~~~~~~
12. ~~~~~~~~~~
13. ~~~~~~~~~~
```



```
14.@EOF
15.SKELETON
16.*DEFINE INSERT
17.*INCREMENT B TO [INSERT]
18.*IF NOT [INSERT,B,1,1]>[#1] OR NOT [INSERT,B,1,1]<[#2]
19.*ELSE
20.¤[INSERT,B,1,1]
21.*CORRECT [INSERT,B,1,1]
22.*END
23.*END
24.*LOOP
25.*END
26.*INCREMENT A TO [P]
27.¤[P,A,1,1]
28.*CORRECT [P,A,1,1]
29.*END
30.*IF [INSERT]>00
31.*PROCESS INSERT [P,A,1,1] [P,A+1,1,1]
32.*END
33.*LOOP
34.*EOF
```

20.9. MERGE OF INPUT STREAMS

The *CORRECT directive merges input streams (permanent, temporary, skeleton) under a given element/version name. All line correction numbers for a given element/version are relative to a base level symbolic stream. The input streams contain images that are merged directly with other input streams. The temporary stream also contains images that modify the permanent stream. Both the direct and modifying images can be intermixed for a given entry in the temporary stream.

If an entry set consists of the line correction numbers with their respective corrections, then a subset is a line correction number optionally followed by corrections. To modify or delete a subset in the permanent stream, the modifying temporary image has the following format:

–N,M/a,b

To create a subset within an entry set in the permanent stream, the modifying temporary image has the following format:

–N,M/a,b/N′,M′

The image containing the line correction number is referenced as line number 0 within the subset. Any number of references can be made to a subset.

Any conflicts in the merging of the input streams are handled by the following priority: temporary stream overrides permanent stream; permanent stream overrides skeleton stream.

Example:

An element is to be corrected with the following subset of permanent corrections:

```
–10,12
   P COR LINE 1
   P COR LINE 2
   P COR LINE 3
   P COR LINE 4
   P COR LINE 5
```

(1) The following is applied to this element within the temporary corrections:

```
–10,12/1,2
   T COR LINE 1
```

The following is sent to the output stream:

```
–10,12
   T COR LINE 1
   P COR LINE 3
   P COR LINE 4
   P COR LINE 5
```

(2) The following is applied to the element within the temporary corrections:

```
–10,12/0,2
   T COR LINE 1
–10,12/4
   T COR LINE 2
   T COR LINE 3
```

The output stream then contains:

```
   T COR LINE 1
   P COR LINE 3
   P COR LINE 4
   T COR LINE 2
   T COR LINE 3
   P COR LINE 5
```

(3) The temporary correction applied to the element contains:

```
–10,12/0,0/10,15
   T COR LINE 1
   T COR LINE 2
–10,12/2,4
–10,12/4/17,17
```

The following is sent to the output stream:

```
–10,15
   T COR LINE 1
   T COR LINE 2
   P COR LINE 1
–17,17
   P COR LINE 5
```

(4) The temporary correction applied to the element contains:

```
–10,12//14,14
–10,12/2,4
–10,12/4/16
```

The output stream then is:

```
–10,12
–14,14
   P COR LINE 1
–16
   P COR LINE 5
```

20.10. BACKUS NORMAL FORM OF SYMSTREAM ELEMENTS

The following is the structure in Backus normal form of the basic elements in SYMSTREAM:

1. <number> :=±99999|±99998|...|±1|0|1|...|999999
2. <alphabetic> :=A|B|C|...|Y|Z
3. <special-character> :=#|Δ|)|&|$|*|(|%|:|?|!|\|'|¤|/|@|[|]|
   –|+|<|=|>|,|.| |≠|;
4. <character> :=<alphabetic>|<special-character>|<number>
5. <name> := <alphabetic>|<name><character>
6. <string> :=<character>|<string><character>
7. <SGS-field> := <string (see NOTE A)>|<SGS-field>,
   <string (see NOTE A)>
8. <SGS> :=<name (see NOTE B)>|<SGS> <SGS-field>
9. <global-variable> := [*<name (see NOTE C)>] |<name (see NOTE C)>
10. <increment-index> := [*<name (see NOTE C)>] |<name (see NOTE C)>
11. <variable> :=<increment-index>|<global-variable>
12. <process-parameter> := [#<number>,<name (see NOTE B)>] | [#<number>]
13. <simple-expression> := <number>|<variable>|<process-parameter>
14. <integer-expression> :=<simple expression>|<integer-expression>
    +<simple-expression>|<integer-expression>
    <simple-expression>
15. <label> := <process-parameter>|<name (see NOTE B)>
16. <SGS-integer-reference> := [<label>] |[<label>,<integer-expression>] |
    [<label>,<integer-expression>,<integer-expression>]
17. <SGS-reference> :=<SGS-integer-reference>|[<label>,<integer-expression>
    <integer-expression>,<integer-expression>]
18. <numeric-expression> :=<integer-expression>|<numeric-expression>
    ±<integer-expression>|<numeric-expression>
    ±<SGS-reference>
19. <relation> :=>|=|<
20. <Boolean-operator> := AND|OR|XOR
21. <Boolean-operand> := Any of a number of a series of operations that gives a
    TRUE or FALSE result |<Boolean-operator>ƀ<Boolean-operand>
22. <Boolean-expression> := <Boolean-operand>|<Boolean-expression>
    <Boolean-operator> <Boolean-operand>

NOTE A: String of 18 or fewer characters not containing a space, period, semicolon, comma, left bracket, or slash (unless enclosed in quotes).

NOTE B: Name of 18 or fewer characters not containing a space, period, semicolon, comma, left bracket, slash, plus, minus, or right bracket.

NOTE C: Name of six or fewer characters not containing a space, period, semicolon, comma, left-bracket, slash, plus, minus, or right bracket.

20.11. DIRECTIVES STRUCTURE

The following is the structure of the directives in terms of satisfied field definitions. The conventions used for the syntax description of the directives will be used.

IF - ELSE - END Directives:

*IF ƀ <Boolean expression>

$$\left[\begin{matrix}\text{———}\\\text{———}\\\text{———}\\\text{———}\end{matrix}\right]$$

$$\left[\text{*ELSE}\right]$$

$$\left[\begin{matrix}\text{———}\\\text{———}\\\text{———}\\\text{———}\end{matrix}\right]$$

$$\left[\text{*END}\right]$$

where the Boolean expression consists of Boolean operands which may be:

$$\text{<variable-name (6-char)>ƀISƀ}\left\{\begin{matrix}\text{SET}\\\text{CLEAR}\end{matrix}\right\}\text{ƀ}$$

VALUEƀOFƀ<number>

$$\left\{\pm\right\}\text{<number>}$$

$$\left\{\begin{matrix}\text{<number (digital)>}\\\text{<string (18-char)>}\end{matrix}\right\}\left[\text{<relation>}\left\{\begin{matrix}\text{<number (digital)>}\\\text{<string (18-char)>}\end{matrix}\right\}\right]\text{ƀ}$$

$$\left\{\begin{matrix}\text{ROW}\\\text{COLUMN}\end{matrix}\right\}\text{ƀSEARCHƀFROMƀ<name (label of 18 char)>}\left[\text{,<number (giving-start-image)>}\right.$$

$$\left[\text{<number (giving start field)>[,<number (giving-start-subfield)>]}\right]\left.\right]\text{ƀFORƀ}\left\{\begin{matrix}\text{<number (digital)>}\\\text{<string (18-char)>}\end{matrix}\right\}\text{ƀ}$$

$$\text{<name (12-char-giving-element)>[/<name (12-char-giving-version>]ƀHASƀ}\left[\left\{\begin{matrix}\text{PERM}\\\text{TEMP}\end{matrix}\right\}\right]\text{ƀCORRECTIONSƀ}$$

INCREMENT — LOOP Directives:

```
*INCREMENTƀ<increment-index-name (6-char)>ƀ[FROMƀ<number>] ƀ[TOƀ<number>] ƀ[BYƀ<number>] ƀ;

 WHILEƀ<variable-name (6-char)>ƀISƀ {SET  } ] ƀ
                                   {CLEAR}

[ ____________ ]
[ ____________ ]
[ ____________ ]
[ ____________ ]

*LOOP
```

PROCESS Directive:

```
*PROCESSƀ<name (18-char)>ƀ[<definition-of-parameters>]
```

SET Directive:

```
*SETƀ<variable-name (6-char)>ƀ[TOƀ<number>]
```

CLEAR Directive:

```
*CLEARƀ<variable-name (6-char)>
```

EDIT Directive:

```
*EDITƀ {ON } ƀ[<char (1-char-for-edit-symbol)>]
       {OFF}
```

CREATE Directive:

```
*CREATEƀSGS:ƀ<any-number of strings and bracketed references to create the secondary image which is the created
      SGS>

*CREATEƀ {PERM} :ƀ<name (12-char-giving-element)>[/<name (12-char-giving-version)>]
         {TEMP}
```

REMOVE Directive:

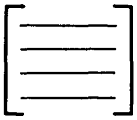
```
*REMOVEƀSGSƀ<name (label-of-18-char)> [,<number (giving-start-image)>[,<number (giving-number-to-be-removed)>]]

*REMOVEƀ {PERM} ƀ<name (12-char-giving-element)>[/<name (12-char-giving-version)>]
         {TEMP}
```

CORRECT — END Directives:

```
*CORRECTƀ<name (12-char-giving-element)>[/<name (12-char-giving-version)>] ƀ {PERM}
                                                                             {TEMP}

[ ____________ ]
[ ____________ ]
[ ____________ ]
[ ____________ ]

*END
```

DEFINE — END Directives:

```
*DEFINEb̸<name (18-char-name-of-sequence)>
```



```
*END
```

BRKPT Directive:

```
*BRKPT[,K]b̸[<name (12-char-name-of-external-file)>]
```

MULTIPLY Directive:

```
*MULTIPLYb̸<number>b̸BYb̸<number>b̸GIVINGb̸<variable-name (6-char)>
```

DIVIDE Directive:

```
*DIVIDEb̸<number>b̸BYb̸<number>b̸GIVINGb̸<variable-name (6-char-for-quotient)>
     [,<variable-name (6-char-for-remainder)>]
```

21. DOCUMENTATION PROCESSORS

21.1. INTRODUCTION

This section describes two documentation processors: FLUSH (flowcharting language for user's simplified handling) and DOC.

The FLUSH processor (see 21.2) can be used to generate flowcharts from assembler-format input. It can:

(a) be directed through the use of option parameters coded in the comments field of the instruction; or

(b) perform an analysis of the assembler instruction statements.

The DOC processor (see 21.3) is used to produce properly formatted printed listings of a document and to update a document. The document is simply a symbolic element in standard format and it may be manipulated by the FURPUR processor (see Section 8) just like any other symbolic element.

Control of the document is provided in the following ways:

- @DOC control statement options
- Control commands within the text that provide:
  - listing control
  - text control
- Editing commands that permit:
  - input line image editing
  - character string editing

21.2. FLOWCHART GENERATOR (FLUSH)

The FLUSH processor accepts assembler-formatted input and produces a flowchart. The FLUSH processor is called by the @FLUSH control statement.

All parameters in the @FLUSH control statement are required except the label and options parameters which are optional.

The format of the @FLUSH control statement is:

```
@label:FLUSH,options   eltname
```

where:

options — The options are:

- F – Specifies that the symbolic element is a FORTRAN element.
- I – Specifies that the symbolic element immediately follows the @FLUSH control statement.
- U – Specifies that the symbolic element is to be updated by the images that immediately follow the @FLUSH control statement.

eltname — Names a symbolic element located in the temporary program file (TPF$).

The following are restrictions which apply only to FORTRAN elements. All other descriptions of FLUSH apply to assembler as well as FORTRAN elements.

(1) The B, C, D, L, R, and S directive options may not be used with FORTRAN elements.

(2) FLUSH code in FORTRAN elements can only be used on FORTRAN comment lines (indicated by a C in column 1).

(3) Labels, when used, must immediately follow the C in column 1 (Clabel:); C is not considered part of the label, but must appear. If there is no label, there must be a space after the C and before the period to mark the beginning of the FLUSH comment.

(4) Characters in columns 72-80 of a comment line containing FLUSH code may interfere with the box text editing unless the box text begins in or before column 11.

(5) The F option must always appear in the @FLUSH control statement to indicate FORTRAN symbolic elements.

21.2.1. GENERAL OUTPUT

FLUSH searches for labels and FLUSH directives, and then produces the following:

(1) a listing of the source input;

(2) a listing of all jumps sorted by source;

(3) a listing of all jumps sorted by destination;

(4) a listing of all labels encountered either in the label field or as destinations of jumps, along with their boxes of definitions; and

(5) box numbers included on the printed flowchart to be used as a reference in conjunction with the listings of labels and jumps.

21.2.2. OPERATION MODES

FLUSH operates in two modes (instruction or comment) which are determined by FLUSH directive options. FLUSH directives are found in the comment field. The format of FLUSH directives is:

. $<options><content>

where:

options A string of alphanumeric characters, terminated by one or more blanks, which are the primary means of directing the processor. Each available option is discussed in 21.2.3.

content A string of up to 60 characters to be inserted into the box generated in the flowchart.

The following restrictions apply to FLUSH directives:

- There must be exactly one blank between the period and dollar sign.
- A semicolon (;) specifies continuation and may be used where necessary within the content string (see 21.2.4).

The FLUSH processor operates in the comment mode when analyzing an element. Under comment mode (assumed initially or set by the D option), the FLUSH processor scans only for the label field, FLUSH directives, and location counter declarations. Comment mode continues until the FLUSH processor encounters a statement containing an R or C option which activates full or limited instruction analysis, respectively. A return to comment mode requires a directive containing the D option.

In either mode, location counter declarations are interpreted so that the coding is flowcharted starting with location counter 1, followed by all the odd location counters, location counter 0, and all the even location counters. If no location counter is specified at the beginning of an element, location counter 0 is assumed when beginning analysis.

The instruction mode begins with a directive containing the R option and continues until the FLUSH processor encounters the D option (resets comment mode). In the instruction analysis mode, the FLUSH processor produces the following:

(1) a processing box for each processing instruction including loads, stores, and arithmetic instructions;

(2) a decision box for each test instruction;

(3) a decision box followed by a change of sequence for each conditional jump;

(4) a change of sequence for each jump instruction; and

(5) a subroutine box for each Load Modifier And Jump (LMJ), Store Location And Jump (SLJ), and Execute Remote (EX) instruction.

The FLUSH processor ignores the DO statement, turning coding off and on with DO/PROC combinations flowcharted as two sets of coding. Upon encountering a PROC or FUNC, the FLUSH processor increments a counter and disregards all coding. When a corresponding END statement is found, the counter is decremented and, if it is zero, interpretation of coding resumes.

The FLUSH processor produces the following symbols and abbreviations when interpreting source language codes:

[]	contents of
<=	is replaced by
IND	indirect addressing
(+)	floating-point add

(/)	floating-point divide
(*)	floating-point multiplication
(–)	floating-point add negative
H+H	add halves
H–H	add negative halves
T+T	add thirds
T–T	add negative thirds
::	absolute value (a field between two colons)
EVEN	parity
ODD	parity

Partial words are indicated by standard mnemonics after the referenced field.

21.2.3. FLUSH DIRECTIVE OPTIONS

Table 21-1 lists the FLUSH directive options, types, and interpretations.

Option Type	Purpose of Option Type	Option Character	Option Description
I	Indicates the type of box to be generated	I J P T W	Produces an I/O box** Produces a connector (change of sequence)** Produces a processing box** Produces a decision box** Produces a subroutine box**
II	Supplies additional information to influence previous options	A	Specifies alternatives for a test
III	Generates certain boxes but does use the content parameter of the directive	E H	Produces an exit box** Produces a header box**
IV	Controls instruction analysis	B C D N R	Disregards location counter* Ignores operation and operand fields except for INSTAB flags* Sets comment mode* Applies options to the following statement Negates previous C and D options*
V	Analyzes specific assembler instructions	L S	Interprets the LMJ instruction as a change of sequence rather than as a subroutine call* Interprets the SLJ instruction as a change of sequence rather than as a subroutine call*

**Does not apply to FORTRAN elements.*

***See 21.2.5 for box formats.*

Table 21-1. Summary of FLUSH Directive Options

21.2.3.1. TYPE I OPTIONS (I, J, P, T, W)

I – Produces an I/O box

J – Produces a connector (change of sequence)

P – Produces a processing box

T – Produces a decision box

W – Produces a subroutine box

In the instruction mode, these options prohibit the interpretation of operation and operand fields for the directives in which they appear. Options I, P, T, and W always produce a box containing up to 60 characters scanned from the content parameter. If more than 30 characters are in the content parameter, the box contains up to three lines.

Only one type I option may be specified, but it may be specified in combination with option types III and IV.

Unless otherwise specified, there are two alternative test results: NO for test fails, and YES for test passes. The normal test fails destination is the next box in sequence. Example 1 shows the output of the FLUSH processor when no A option is applied (see type II option).

Example 1:

```
     LABEL       Δ    OPERATION      Δ               OPERAND        Δ              COMMENTS
1               10               20               30                40               50
                 • $T THIS IS A TEST
                 • $P NEXT BOX HERE
                 • $P FOLLOWING BOX
```

Produces:

```
                 :
    1            V
  ????????????????????????????
 ?                            ?
?        THIS IS A TEST        ?->:
 ?                         YES    :
  ????????????????????????????    :
                 : NO             :
                 :                :
    2            V                :
********************************  :
*                              *  :
*        NEXT BOX HERE         *  :
*                              *  :
********************************  :
                 :                V
                 : <———————————————
    3            V
********************************
*                              *
*        FOLLOWING BOX         *
*                              *
********************************
                 :
                 :
                 V
```

The numbers above the left corner of each box are the box numbers.

A decision box having more than two branches is generated by a one-digit numeric character, specifying the number of alternatives, directly after the T. All n alternatives and at least n–2 destinations must be specified by the A option.

21.2.3.2. TYPE II OPTION (A)

A – Specifies alternatives of a test

The A option must immediately follow the test to which it applies. Alternatives are placed in the comment field and are separated by blanks. The number of alternatives must match that number specified for the test. If a test has two alternatives, the A option is optional; if it has more than two, the A option must be used.

The A option may also be used to change the sequence as applied to a test. The format of the comment field is:

<alternative> [,<destination>] ...<alternative>[,<destination>]

The first alternative, without a specified destination, applies to the next box in sequence. The second alternative applies to the box following the next box in sequence, and so on.

Example 2:

```
      LABEL       Δ    OPERATION      Δ                 OPERAND        Δ               COMMENTS
1               10               20               30               40               50
               .  $T   TEST
               .  $A   BAD GOOD
               .  $P   BOX
```

Produces:

```
                 :
     4           V
  ??????????????????????????????
 ?                              ?
?             TEST               ?->:
 ?                           GOOD   :
  ??????????????????????????????    :
                :BAD                :
                 :                  :
  5              V                  :
*********************************   :
*                               *   :
*              BOX              *   :
*                               *   :
*********************************   :
                 :                  :
                 :                  V
                 V <-----------------
```

Example 3:

```
      LABEL       Δ    OPERATION      Δ                 OPERAND        Δ               COMMENTS
1               10               20               30               40               50
               .  $T   TEST
               .  $A   GOOD,L1   BAD,L2
```

Produces:

```
                         :
      6                  V
   ?????????????????????????????
  ?                             ?
 ?              TEST             ?->:
  ?                           BAD   :
   ?????????????????????????????    :
                         :  GOOD    :
                         :          :
                         V          :
      7                 ****        :
                      *  L1  *      :
                        ****        :
                                    V
                         : <---------
                         V
      8                 ****
                      *  L2  *
                        ****
```

Example 4:

LABEL	OPERATION	OPERAND	COMMENTS
•	$T TEST		
•	$A OK POOR,EXT		
•	$P BOX		

Produces:

```
                         :
      9                  V
   ?????????????????????????????
  ?                             ?
 ?              TEST             ?->:
  ?                            OK   :
   ?????????????????????????????    :
      10                 : POOR     :
                         :          :
                         V          :
                        ****        :
                      * EXT  *      :
                        ****        V
                         : <---------
     11                  V
  *******************************
  *                             *
  *             BOX             *
  *                             *
  *******************************
                         :
                         :
                         V
```

To generate multiple branch decisions, the decision box must be generated with a T option. The alternatives and destinations must then be specified on the following by using the A option.

Example 5:

```
LABEL      OPERATION             OPERAND            COMMENTS
1        10         20         30         40         50
         .$T5 BRANCH
         .$A   1,LAB1 2 3,LAB3 4,LAB4 5,LAB5
         .$P   NEXT BØX
```

Produces:

```
                :
   1            V
  ????????????????????????????
 ?                            ?
?            BRANCH            ?
 ?                            ?
  ????????????????????????????
                :
                : 1       ****
                : ----->* LAB1 *
   **** 3       :         ****
 * LAB3 *<----- :
   ****         : 4       ****
                : ----->* LAB4 *
   **** 5       :         ****
 * LAB5 *<----- :
   ****         :
                :
                : 2
   6            V
********************************
*                              *
*           NEXT BOX           *
*                              *
********************************
                :
                :
                V
```

Example 6:

```
LABEL      OPERATION             OPERAND            COMMENTS
1        10         20         30         40         50
         .$T5  BRANCH
         .$A   1,LAB1 2 3,LAB3 4 5,LAB5
         .$P   NEXT BØX
         .$P   FØLLØWING BØX
```

Produces:

```
                :
      1         V
  ?????????????????????????????
 ?                             ?
?            BRANCH             ?>:
 ?                             ?  :
  ?????????????????????????????   :
                :                 :
                :   1     ****    :
                : ----->* LAB1 *  :
    **** 3      :         ****    :
  * LAB3 *<---- :                 :
    ****        :   5     ****    :
                : ----->* LAB5 *  :
         ?      :         ****    :
                :                 :
       5        V                 :
********************************* :
*                               * :
*           NEXT BOX            * :
*                               * :
********************************* :
                :                 V
                :
       6        V
*********************************
*                               *
*        FOLLOWING BOX          *
*                               *
*********************************
                :
                :
                V
```

21.2.3.3. TYPE III OPTIONS (E, H)

E – Produces exit box

H – Produces header box

The E option indicates that this box (or the last box generated if the E option does not appear in the directive which generates a box) is the end of a chain of coding and should not be connected to the next box in sequence. A triangle containing the word EXIT is generated to terminate the chain. When applied to a decision box, an exit triangle occupies the position for the test fails condition, and is considered to be the next box in sequence.

The H option indicates that this box is the head of a chain (the box at the top of a new column). Preceding this box is a triangular box designating the entrance to the chain of coding. If a label is specified, it appears in the triangle. The word ENTRY always appears in the triangle.

Example 7:

LABEL (1)	Δ OPERATION (10)	Δ OPERAND (20)	Δ (30)	Δ (40)	COMMENTS (50)
	TE,U	AO,1	.	$E	
	A	AO,LØC			

Produces:

```
                 1         :
                           V
          ??????????????????????????????
         ?                              ?
        ?             1=A0               ?->:
         ?                           YES    :
          ??????????????????????????????    :
                           : NO             :
                           :                :
                           V                :
                       *********            :
                       * EXIT  *            :
                        *     *             :
                         *   *              :
                          * *               :
                           *                :
                                            V
                           : <--------------
                 2         V
         *******************************
         *                             *
         *        A0<=[A0]+[LOC]       *
         *                             *
         *******************************
                           :
                           :
                           V
```

Example 8:

LABEL (1)	OPERATION (10)	OPERAND (20)	COMMENTS (40)
	TZ	A0	
	A	A0,LOC	. $E

Produces:

```
                 3         :
                           V
            ??????????????????????????
           ?                          ?
          ?           [A0]=0           ?->:
           ?                       YES    :
            ??????????????????????????    :
                           : NO           :
                           :              :
                 4         V              :
        ********************************* :
        *                               * :
        *          A0<=[A0]+[LOC]       * :
        *                               * :
        ********************************* :
                           :              :
                           :              :
                           V              :
                       *********          :
                       * EXIT  *          :
                        *     *           :
                         *   *            :
                          * *             :
                           *              V
                           :<-------------
                           V
                        NEXT BOX
```

Example 9:

```
LABEL                    OPERATION                    OPERAND                    COMMENTS
1          10            20            30            40            50
LABEL      L,U           A0,1                        . $H
           EX            TAB,X1                      . $P GET CHARACTER
           $LJ           $SUBR
           J             RETN
```

Produces:

```
                    *********
                    * LABEL *
                     *ENTRY*
                      *   *
                       * *
                        *
                        :
                        :
           1            V
           ********************
           *      A0<=1       *
           *                  *
           ********************
                        :
                        :
           2            V
           ********************
           * GET  CHARACTER   *
           *                  *
           ********************
                        :
                        :
           3            V
           ********************
          *                    *
         *        SUBR          *
          *                    *
           ********************
                        :
                        :
           4            V
                       ****
                     * RETN *
                       ****
```

21.2.3.4. TYPE IV OPTIONS (B, C, D, N, R)

B — Disregards location counter

C — Ignores operation and operand fields

D — Sets comment field

N — Applies options to following card

R — Negates previous C and D options

The B option overrides location counter specifications in the directive in which it appears and all following directives and interprets coding in the order of input.

The C option discontinues instruction analysis except for certain instructions which include tests, jumps, SLJ, LMJ, and ER instructions. Full instruction analysis is continued when an R option is encountered.

The D option is similar to the C option except that it applies to all following directives regardless of the instruction. The comment mode continues until an R option (instruction mode) is encountered.

The N op.ion applies all following options to the next directive and not to the current directive.

The R option resumes interpretation of operation and operand fields of the current and subsequent directives and negates previous C and D options.

21.2.3.5. TYPE V OPTIONS (L, S)

L — Interprets the LMJ instruction as a change of sequence rather than as a subroutine call

S — Interprets the SLJ instruction as a change of sequence rather than as a subroutine call

These options apply only in the instruction mode and produce a connector rather than a subroutine box. The connector contains the six-character label found in the address field of the LMJ or SLJ instruction.

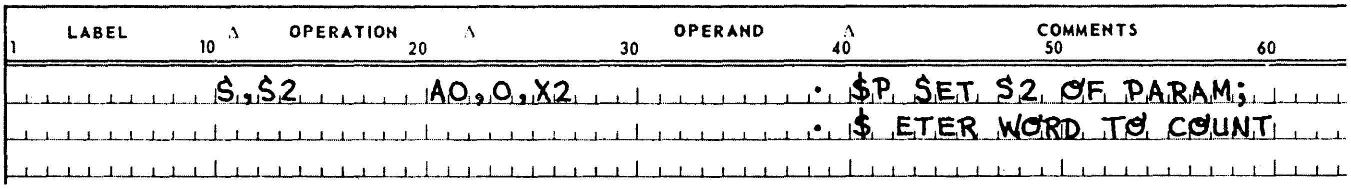
21.2.4. CONTINUATION REQUIREMENTS

When it is necessary to continue FLUSH documentation on a second card, the continued line is differentiated from program documentation by the characters, period-blank-dollar-blank (. $). This character string is followed by the rest of the comment from the previous statement. The required string (. $) may start in any column, but only one blank may appear between the dollar and the continued text.

The semicolon (;) identifies a comment to be continued and may appear in one of two positions.

(1) Immediately after the last character in a statement if that word is continued on the next statement.

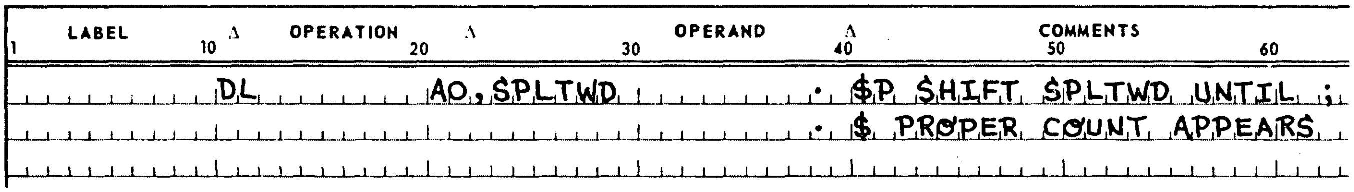
Example:


LABEL
OPERATION
OPERAND
COMMENTS
1
10
20
30
40
50
60

```
          $,$2      AO,0,X2               . $P SET $2 OF PARAM;
                                          . $ ETER WORD TO COUNT
```

(2) One blank after the last character in a statement if that word is complete, but more words follow in the next statement.

Example:


LABEL
OPERATION
OPERAND
COMMENTS
1
10
20
30
40
50
60

```
          DL        AO,$PLTWD             . $P SHIFT $PLTWD UNTIL ;
                                          . $ PROPER COUNT APPEARS
```

This format also applies to statement with the A option which may require continuation.

21.2.5. SUMMARY OF BOX TYPES

Connector:

```
  ****
*      *
  ****
```

Decision:

```
  ???????????????????
 ?                   ?
?                     ?
 ?                   ?
  ???????????????????
```

Entry and Exit:

```
*********
*       *
 *     *
  *   *
   * *
    *
```

I/O:

```
*******************
 *               *
  *             *
   *           *
    ************
```

Processing:

```
*******************
*                 *
*                 *
*******************
```

Subroutine:

```
  *******************
 *                   *
*                     *
 *                   *
  *******************
```

21.3. DOCUMENT PROCESSOR (DOC)

The DOC processor can be used to produce a properly formatted printed listing of a document or to update a document. The DOC processor is called by the @DOC control statement.

All parameters in the @DOC control statement are optional except @, DOC, eltname-1, and eltname-2.

The format of the @DOC control statement is:

```
@label:DOC,options    eltname-1,eltname-2,,FASTRAND-pos
```

where:

options — See Table 21-2

eltname-1 — Specifies the symbolic input element (see I option – Table 21–2)

eltname-2 — Specifies the symbolic output element

FASTRAND-pos — A one- to three-digit integer which specifies the number of FASTRAND positions to be used. If omitted, a maximum of 10 positions is assumed. Two consecutive commas must precede this parameter when it is coded.

Option	Description
A	Prevents ERR$ termination in case of an internal error.
D	Inserts the current date in columns 51 through 59 of the title on each page (but not in the output element) in format dd mmm yy.
F	Allows insertion of paragraph titles in the table of contents.
H	Turns off automatic hyphenator.
I	Accepts input element from run stream. When the I option is used (input from run stream), eltname-1 is not used and eltname-2 must be preceded by a comma.
L	Lists information from internal control as flags in the right-hand margin.
M	Moves line numbers within the margin guides and appends an asterisk (*) if the line was newly inserted or altered.
N	Produces no listing.†
S	Produces single-spaced listing (otherwise, assume double spacing).†
W	Lists correction directives.
X	Aborts run if table of contents overflows or input element is empty.

†May also be requested by internal control directives.

Table 21-2. @DOC Control Statement, Options

21.3.1. OUTPUT LISTINGS

The first image in a document is the title. It may begin in column 1 and extend to column 59 (column 49 if D option is used – see Table 21-2). The title of the document is printed at the top of each page, with the current date suffixed if the D option is used. The chapter and page number follow; a hyphen separates the two numbers. The maximum number of characters allowed in this field (numbers and hyphen) is six. Two identical lines, each containing two periods, are printed below the title. The periods indicate where to trim the standard printer form to 8 1/2-inch width. The M option controls the horizontal position of the periods on the page.

The text (up to 50 lines) appears below the title and page guides. The line numbers of the output element are printed in the left margin. To the right of the text, *NEW is printed if the line was inserted or altered. If the L option is specified, the first letter of the directive on each internal control directive is printed in the far right margin. The letter may be followed by a number if the control directive uses such a specification.

Hyphenation at the end of a line is automatic, except when directed by the H option or the HYPHEN control directive. When a document is modified, hyphenated words are joined together when necessary. However, when constructing tables, a SPACE 0 directive must be used where required to prevent text composition from occurring; text can not extend beyond column 66.

21.3.2. INTERNAL CONTROL DIRECTIVES

Internal control directives are distinguished from text by a nonblank character in column 1. The types of control directives available are described in the following paragraphs.

21.3.2.1. TITLE CONTROL

Title control directives have one of the digits, 1, 2, 3, or 4, in column 1, followed by text, which is the title for that portion of the document. This title has nothing to do with the title of the document (the first input image). Four counters are maintained by the DOC processor: chapter, section, subsection, and paragraph, in that order of precedence. Table 21-3 describes the functions of these counters and respective control directives.

Digit In Column 1	Counter	Description
4	Chapter	The chapter counter is increased by one; the other three counters contain zeros. The chapter number is edited, followed by a period, followed by the line of text. The line is printed and inserted in the table of contents, followed by the page number. A page eject always occurs before the line is printed.
3	Section	If the chapter counter is zero, it is set to one, and the other two counters contain zeros. The chapter and section numbers are edited; the line is printed and inserted in the table of contents.
2	Subsection	If the chapter or section counter is zero, it is set to one; the paragraph counter is set to zero. The three counters are edited; the line is printed and inserted in the table of contents.
1	Paragraph	Similar to subsection control except that the F option must be used to insert the paragraph title in the table of contents.

Table 21-3. DOC Processor, Title Control Directives

A page eject always occurs when the chapter number is changed. Appropriate spacing is performed before and after printing. The image inserted in the output element is the original image with a digit in column 1, not the edited image as it appears on the listing. In the table of contents (printed at the end of the listing), the section, subsection, and paragraph titles are indented to the right. Since the numbers are inserted at the left and the title is right-shifted, chapter, section, and other titles must not extend too far to the right. If a maximum of approximately 45 characters is exceeded, the image on the listing is truncated (the line image in the output element is unaffected).

21.3.2.2. LISTING CONTROL

Listing control directives specify the appearance of the listing. The format is a directive word, beginning in column 1, extending no further than column 6. Table 21-4 lists the directives and their functions.

Directive	Description
DOUBLE	Cancels the SINGLE directive or the S option on the @DOC control statement.
EJECT	Ejects paper to begin printing new page.*
INSERT	Inserts up to 66 characters, beginning in column 13, into the table of contents.
LIST	Turns on the listing. Cancels the UNLIST directive or the N option on the @DOC control statement.
REMAIN n	Ejects paper if fewer than n lines (an unsigned integer in columns 8-9) remain on the page. If n is omitted or is zero, the paper is not ejected.*
SINGLE	Same as the S option on the @DOC control statement.
SPACE n	Spaces paper down n lines, where n is an unsigned integer in columns 7-8. A page eject occurs if fewer than n lines remain on the page. If n is omitted, 1 is assumed. If a blank image is in the input, it is converted to a space directive with n omitted. If n is 0, no spacing is performed (SPACE 0 has a special function – see 21.3.2.3).*
UNLIST	Turns off the listing. (Same as the N option on the @DOC control statement.)

**When in the UNLIST mode, the EJECT, REMAIN, and SPACE control directives are ignored.*

Table 21-4. DOC Processor, Listing Control Directives

21.3.2.3. TEXT CONTROL

The DOC processor assigns proper margins to the text that appears on the listing (and in the output element). This is accomplished by moving words (strings of consecutive nonblank characters) and strings of blanks from the input line to an output area until it is filled. The output line is then printed and sent to the output element. Control over this process is available by using the COLUMN, HYPHEN, and SPACE 0 directives and an input line with a 0 in column 1.

The text movement and line composition are terminated under the following conditions:

- start of a new paragraph,
- recognition of a control directive

A new paragraph is recognized by a text line (not a control directive) on which the first nonblank character is not in the previously specified column n. This includes both indention to the right and extension to the left. For this reason, all lines to be moved must begin in column n. The COLUMN directive is used when text is indented.

The format of the COLUMN directive is:

```
COLUMN n
```

where n is an unsigned integer in columns 8-9 designating the column used as the left margin for the text. The integer must be greater than 1 and less than 61. If n is omitted, 2 is assumed. Column 2 is in effect until the first COLUMN directive appears. When the movement of text to form complete lines is halted, any partial line already created is printed and a new line begins.

The HYPHEN directive suppresses/activates the automatic hyphenator as needed. The format of the HYPHEN directive is:

```
HYPHEN mode
```

where mode begins in column 8 and is either ON or OFF; it designates that the automatic hyphenator is to be turned on or off, respectively.

Two special directives are available to halt text movement. A SPACE 0 directive stops text composition and begins a new line. The identical effect is obtained by placing a 0 in column 1 of a line, the text of which is to begin a new line. On the output, the 0 is replaced by a blank and the line is preceded by a SPACE 0 directive. A SPACE 0 directive or a 0 in column 1 of a text line is deleted if it appears where text movement is stopped because of another control directive or the start of a new paragraph.

21.3.2.4. EDITING CONTROL

Line image editing is provided by standard correction lines in the format common to all processors (see 9.5).

Character string editing is the manipulation of character strings on a particular line image of the input (text or control directive) in the run stream or input element. Character editing directives have an ampersand (&) in column 1, but are otherwise free form. They immediately follow the line to be altered.

Examples:

- To correct line 18 of the input element:

  ```
  –18
  &...
  ```

- To correct an image in the run stream:

  ```
  <BAD line>
  & ...
  ```

The character correction lines have four formats. In the following format descriptions, the slash (/) is used as a character string delimiter; however, any nonblank character which does not appear in the text on the line may be used. Only one character may be used as the delimiter on a card. The first nonblank character on the card (after the ampersand in column 1) is the delimiter character for that line and any number of blanks may precede it.

Format 1:

```
&/<oldtext>/<newtext>/
```

The line being corrected is searched for the first occurrence of old text which is replaced by new text.

Format 2:

```
&/<oldtext>/<newtext>/*
```

Similar to format 1, except that every occurrence of old text is replaced by new text. The asterisk must immediately follow the third delimiter. If a character other than asterisk is used, format 1 is assumed. However, this does not preclude the use of the asterisk as the delimiter.

Format 3:

```
&//<column-nbr>/<newtext>/
```

The newtext string is inserted in the line to be modified beginning at the column specified by column-nbr.

The insert overlays any previously existing characters. The first two characters after the ampersand are consecutive delimiters; no characters may separate them. The column-nbr may be stated or omitted. If it is omitted or contains a nonblank or nondigit character, the value assumed is the value on the most recent COLUMN directive.

Format 4:

& //<column-nbr>/

The first nonblank character, after column 1, on the line being altered is placed in the column specified by column-nbr. The entire character string is shifted to satisfy this specification. This format is similar to format 3, except that there must be exactly three delimiters on the card.

Character correction directives may alter the length of the line image. If it is shortened, the unused area is filled with blanks. If it is lengthened, the excess is checked to see if it is entirely blank. If so, it is ignored. Otherwise, the line is continued in such a way that a word is not broken between line images. This means that each line image ends with a string of blanks and the first word of each continuation line begins in column n, with columns 1 through n–1 containing blanks. The length of a word must not exceed 67–n characters, where n is the column number on the currently effective COLUMN n directive. A line may expand to a maximum length of 600 characters by using more than one character correction directive. The line to be changed is followed with as many character insertion directives as necessary with the changes made in the order of the directive encountered.

Examples:

■ The following is a portion of a document:

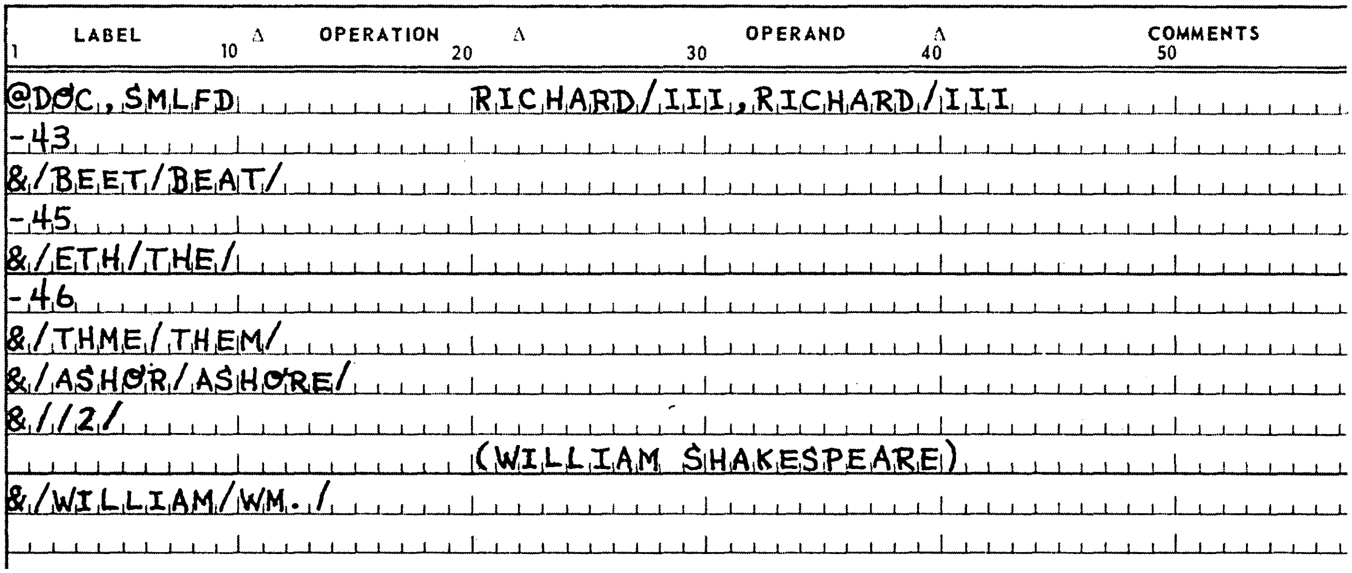
43 ...UNARMED AND UNPREPARED TO BEET THEM BACK. 'TIS

44 SAID THAT RICHMOND IS THEIR ADMIRAL. AND THERE THEY

45 HULL, AWAITING BUT EHT AID OF BUCKINGHAM TO

46 WELCOME THME ASHOR....

There are several typographical errors in the document which can be corrected as follows:



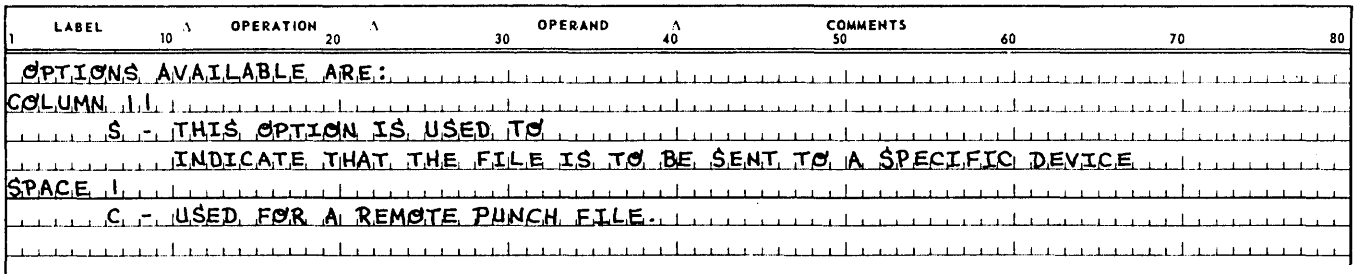
```
   LABEL      Δ   OPERATION    Δ          OPERAND      Δ          COMMENTS
1          10              20           30           40              50
@DØC,SMLFD        RICHARD/III,RICHARD/III
-43
&/BEET/BEAT/
-45
&/ETH/THE/
-46
&/THME/THEM/
&/ASHØR/ASHØRE/
&//2/
                  (WILLIAM SHAKESPEARE)
&/WILLIAM/WM./
```

Resulting in the following output, assuming no line number changes:

```
43*  ...UNARMED AND UNPREPARED TO BEAT THEM BACK. 'TIS

44   SAID THAT RICHMOND IS THEIR ADMIRAL. AND THERE THEY

45*  HULL, AWAITING BUT THE AID OF BUCKINGHAM TO WELCOME

46*  THEM ASHORE....

47             (WM. SHAKESPEARE)
```

- The following illustrates an appropriate use of the COLUMN directive to produce a blocked, left-justified description following each item in a list.

The input for a portion of a document is:



LABEL 10 OPERATION 20 30 OPERAND 40 COMMENTS 50 60 70 80

```
 OPTIONS AVAILABLE ARE:
COLUMN 11
      S - THIS OPTION IS USED TO
          INDICATE THAT THE FILE IS TO BE SENT TO A SPECIFIC DEVICE
SPACE 1
      C - USED FOR A REMOTE PUNCH FILE.
```

The output generated is:

```
         C1     C10       C20     C30     C40      C50     C60  C66
         |      |         |       |       |        |       |    |
    14   OPTIONS AVAILABLE ARE:
                                                                   C11
    16          S – THIS OPTION IS USED TO INDICATE THAT THE FILE IS TO BE
    17              SENT TO A SPECIFIC DEVICE.
                                                                   S1
    19          C – USED FOR A REMOTE PUNCH FILE.
```

The proper use of COLUMN control statements cannot be overemphasized.

22. MASTER FILE DIRECTORY

22.1. INTRODUCTION

This section describes the master file directory (MFD) structure. It also describes the functions used to retrieve and update entries in the MFD.

22.2. MASTER FILE DIRECTORY STRUCTURE

For catalogued files, entries are constructed containing the identification and characteristics of each file and these entires are maintained by the system in a MFD. The MFD consists of look-up table entries and directory items. A look-up table is used to link to the catalogued files. The filename and qualifier of a file are folded to provide an index to the look-up table. The length of the look-up table is a system generation parameter. Directory items are 28-word areas used to store information needed to maintain a file's identity and description. Directory items are defined as:

- Search Item — Used to locate pointers to lead items which have the same index into the look-up table.

- Lead Item — Used to provide the link between the qualifier/filename combination and the F-cycle of the file. The lead item contains pointers to the main items.

- Main Item — Used to store information pertaining to an F-cycle of a file. There is one main item for each F-cycle of the file. The main item contains pointers to the granule item.

- Granule Item — Used to define a file. A granule item is comprised of granule entries which define the absolute description of a file (for example, physical mass storage addresses or reel numbers). Since temporary files are not used by more than one run, only granule items are needed to describe them in the MFD.

A word in the look-up table contains one of the following types of entries:

- Zero — A look-up table word is zero if no file has an index equal to the word number.

- Pointer to Lead Item — If only one file has an index equal to the word number, the look-up table word points to the file's lead item.

- Pointer to Search Item — If more than one file has an index equal to the word number, the look-up table word points to the search item.

Figure 22–1 illustrates a MFD entry.

See 22.5 for detailed layout of the various directory items.

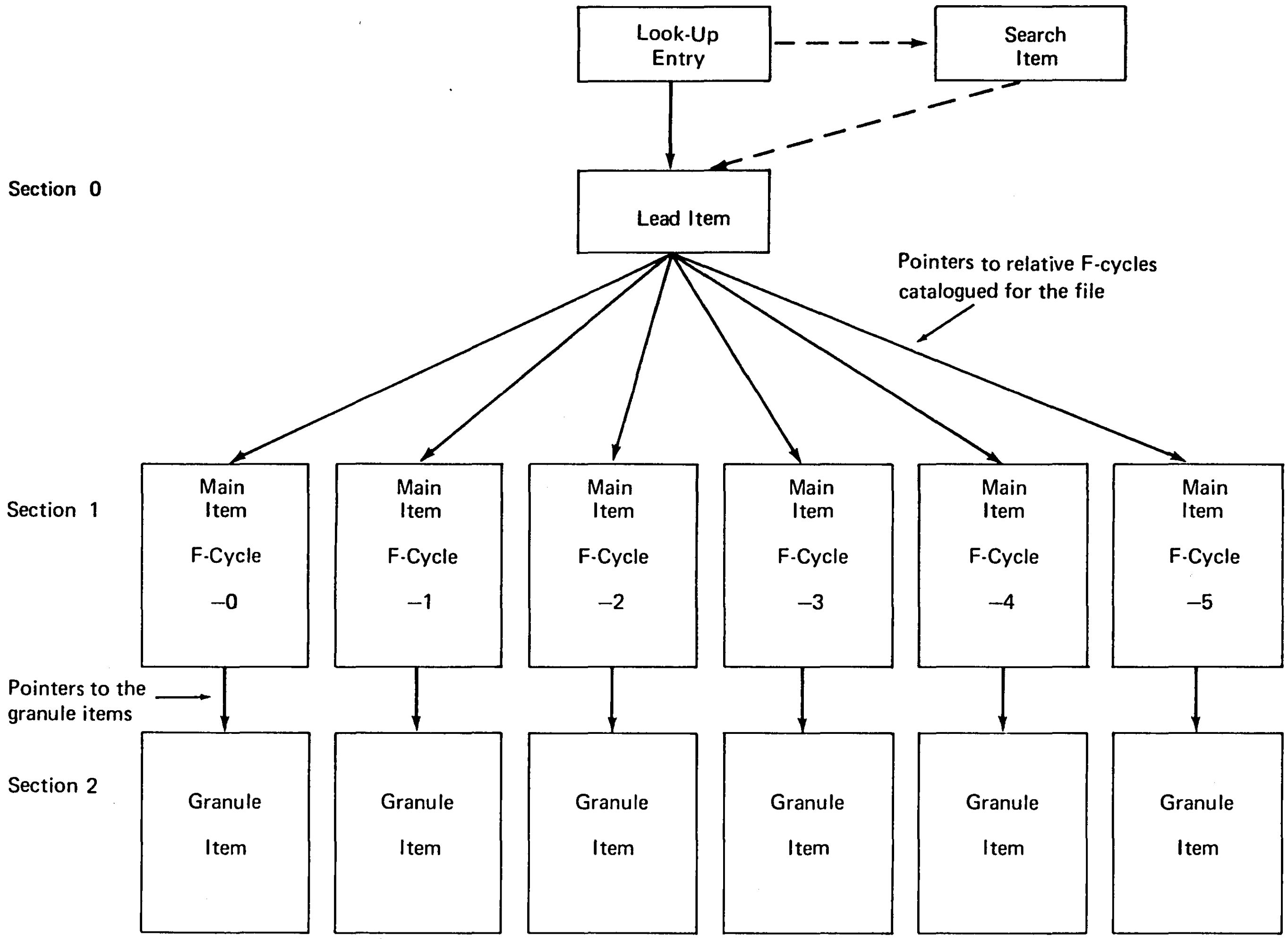


Figure 22–1. Example of a MFD Entry

22.3. MASTER FILE DIRECTORY MANIPULATION (MSCON$)

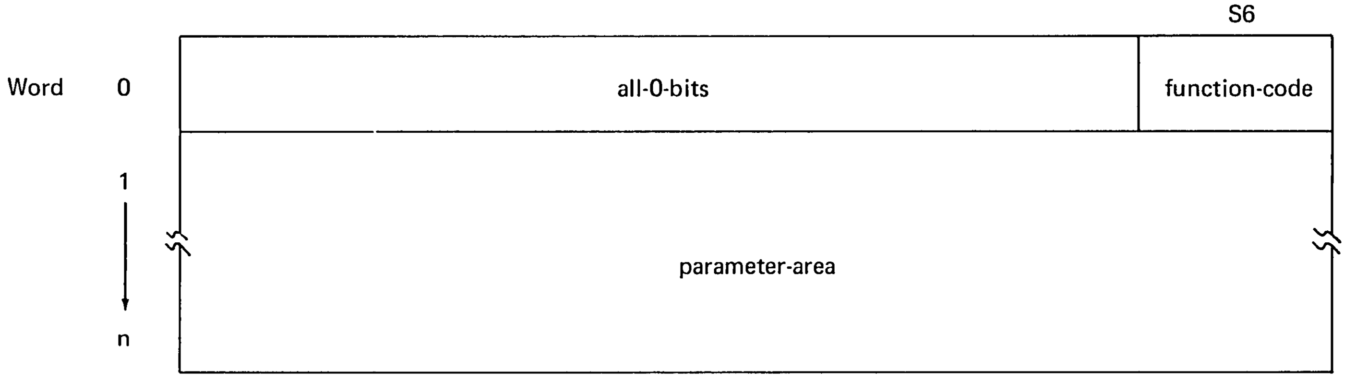
The MSCON$ executive request (ER) enables the user to obtain information from the MFD. It enables the user to obtain either the entire MFD or entries pertaining to a particular file. In addition, the MSCON$ request provides the means of altering indicators in the directory items.

Several of the MSCON$ functions can be performed only by privileged runs. Classified file information is revealed only to privileged runs. A privileged run is one that has the SYS$*DLOC$ file assigned to the run with the correct read/write keys specified at assign time. The SYS$*DLOC$ file is a system file which is created at system start-up time and its primary use is to identify privileged runs.

The general format of the MSCON$ request is:

```
L,U  A0,pktaddr
ER   MSCON$
```

The pktaddr loaded into register A0 references a packet having the following general format:



The function code in S6 of word 0 indicates the particular control routine desired. The parameter area is a communications area used to pass information from the user to the ER.

The function codes are as follows:

15_8	–	DGET$
16_8	–	DGETP$
20_8	–	DREAD$
30_8	–	DBITS$
31_8	–	DBACK$
32_8	–	DLAP$
33_8	–	DUNLD$
34_8	–	DCYC$
35_8	–	DKEY$
36_8	–	DBB$
37_8	–	DREG$
40_8	–	DLINK$
41_8	–	DADD$
60_8	–	MSALL$

22.3.1. ITEM RETRIEVAL FOR ALL FILES (DGET$)

This function creates a file containing the directory items. This file effectively acts as a checkpoint of the MFD. The format of the packet in the MSCON$ request (see 22.3) is:

			S6
Word 0			15_8
1	output-filename		
2			
3	initial-reserve-in-tracks		
4	addr-of-first-user-supplied-buffer	addr-of-second-user-supplied-buffer	

Words 1 and 2

output-filename Specifies the file to be created.

Word 3

initial-reserve-in-tracks Specifies the number of tracks initially reserved for the DGET$ output file.

Word 4

Addr-of-first-user-specified-buffer Specifies starting address of a track-size (1792 words), user-supplied buffer.

Addr-of-second-user-specified-buffer Specifies starting address of a track-size (1792 words), user-supplied buffer.

If the output file is sufficient in length to accomplish the DGET$ request, word 4 is used to set up double-buffered I/O for transferring MFD information to the output file. If, however, the output file is insufficient to contain the current total of directory items, a value is returned in word 3 that represents the number of tracks needed to perform the DGET$ request. As discussed in 22.4, this request is rejected and the appropriate error status is returned in register A0.

If the SYS$*DLOC$ file is not assigned to the referencing run, the read/write keys for files having keys and the project-id for files with a project-ids different from the project-id of the calling run are set to //////. Blank read/write keys are left as blanks and not set to //////.

If the SYS$*DLOC$ file is assigned to the referencing run, only files catalogued with a G option have the read/write keys and the project-id altered in this way.

The output from the DGET$ request consists of a table of contents (starting at sector 0), look-up table (starting at sector 1), and directory items (lead, main, granule, and search items starting at sector 0 of the track following the look-up table). The output is written into the user file named in the input packet. The output track formats are as follows:

- Track 0

Word		H1	H2	
0		*time-and-date-of-creation*		Table of Contents
1			*length-of-look-up-table*	
2		*track-nbr-of-first-MFD-track*	*nbr-of-tracks-written*	
3 ↓ 27				
28	*0*	*zero*		Look-Up Table
29	*0*	*lead-item-addr*		
30	*1*	*search-item-addr*		
31 ↓ n		*Up to 1761 additional look-up table entries*		

Word 0

The date and time when the DGET$ request created this file, where:

S1 —Month

S2 —Day

S3 —Year (modulo 1964)

H2— Time in seconds from midnight

Word 1

length-of-look-up-table This table starts in sector 1 of the first track. If the look-up table is greater than 63 sectors, the remainder is written in successive tracks.

Word 2

track-nbr-of-first-MFD-track — If the look-up table is located in the first two tracks, the number in this word is 3.

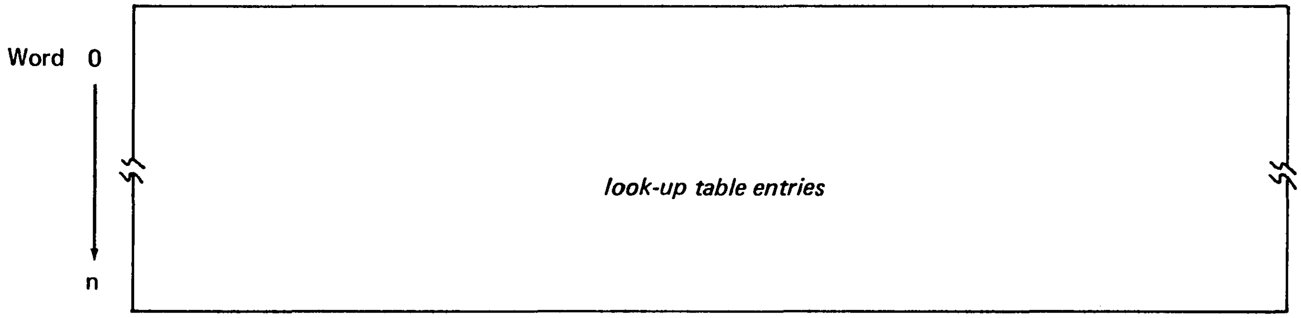
number-of-tracks-written — The number of tracks written by the DGET$ request in the user's file.

Word 28

This word is the start of the look-up table. (The length of the look-up table is equal to the DCLUTS tag in the executive, which has a value supplied by the DGET$ request in H2, word 1 of the output file.) The first 63 sectors (1764 decimal words) of the look-up table are written in track 0. If there are more than 1764 entries in the look-up table, the remaining entries are written in the following tracks as needed.

The formats of look-up table entries are illustrated in words 28 through 30; the entry is zero if no file has a look-up index equal to the word number. This type of format is shown in word 28. If only one filename has this index, the file's lead item address is stored in this word, as shown in word 29. If more than one file has this index, the entry contains the address of a search item, as shown in word 30.

- Track 1



- MFD Track

Start of the MFD tracks. The first MFD track written for each unit contains a directory allocation sector (DAS) as sector 0 of that track. Tracks for the unit are written to the user's file in the same order as listed in the allocation sector. The format of the DAS and MFD track is as follows:

	35	34 0
Word 0	*MFD-track-addr*	
1	*allocation-bits-for-first-32-sectors*	
2	*allocation-bits-for-last-32-sectors*	
3	*S*	*MFD-track-addr*
4	*allocation-bits-for-first-32-sectors*	
5	*allocation-bits-for-last-32-sectors*	
6 → 26	*Up to seven three-word entries as in words 0–2*	
27	*S*	*link-to-next-DAS*
28 → n	*MFD items (see Figures 22–2 through 22–10 for MFD item formats)*	

Word 0

Address of this MFD track.

Word 1

Allocation bits for the first 32 sectors of this track. Bit 35 corresponds to sector 0, bit 34 to sector 1, and so on. The bit is set if this sector is allocated.

Word 2

Allocation bits for the last 32 sectors of this track. Bit 35 corresponds to sector 32, bit 34 to sector 33, and so on. The bit is set if this sector is allocated.

Word 3

Address of the next MFD track. If this word is negative (S = 1), then a MFD track does not exist for this entry. This track does not contain a DAS in sector 0.

Word 4

Same as word 1.

Word 5

Same as word 2.

Word 27

Address of the next MFD track with a DAS in sector 0. If this word is negative, there are no more MFD tracks for this unit.

- Tracks t+2 through m–1:

  The value of m is found in the table of contents (track 0, sector 0) in H2 of word 2.

22.3.2. ITEM RETRIEVAL FOR DISC PACKS (DGETP$)

This function creates a file in the same manner as DGET$ (see 22.3.1). The only difference is that the directory items from a specified pack are inserted into the output file. The format of the packet used in the MSCON$ request (see 22.3) is:

Word			S6
0			16_8
1	output–filename		
2			
3			
4	addr-of-first-user-supplied-buffer	addr-of-second-user-supplied-buffer	
5	pack-id		

Words 1 through 4

Same as for DGET$ function (see 22.3.1).

Word 5

A one- to six-character alphanumeric that specifies the disc pack.

22.3.3. ITEM RETRIEVAL FOR AN INDIVIDUAL FILE (DREAD$)

This function provides a full complement of MFD information for a single file. The format of the packet used in the MSCON$ request (see 22.3) is:

<table>
<tr><th></th><th>35</th><th>23</th><th>17</th><th>11</th><th>5 0</th></tr>
<tr><td>Word 0</td><td colspan="4"></td><td>20_8</td></tr>
<tr><td>1</td><td colspan="5" rowspan="2">filename</td></tr>
<tr><td>2</td></tr>
<tr><td>3</td><td>buffer-length</td><td>start-item</td><td colspan="3">buffer-addr</td></tr>
<tr><td>4</td><td>starting-sector</td><td colspan="2"><em>sector-count</em></td><td colspan="2"></td></tr>
</table>

Words 1 and 2

filename — Specifies the file for which the MFD information is to be retained.

Word 3

buffer-length — Length of the buffer into which the directory items are to be written (in words).

start-item — Specifies the MFD items desired, where the value of starting item can be:

0_8 — Start with the lead item, and then continue with the main and granule items (in ascending order) until the end of the MFD for this file or the buffer limit is reached.

1_8 — Same as above, but start with main item.

2_8 — Same as above, but start with granule item.

buffer-addr — Address of buffer into which the directory items are to be written.

Word 4

starting-sector — If nonzero, start at indicated higher sector number within starting item (for example, start at sector 26 of granule item).

sector-count — Number of sectors written in user buffer (supplied by DREAD$ request).

The MFD information supplied by the DREAD$ request appears in the output buffer in the order specified in the user packet. If file SYS$*DLOC$ is assigned to the calling run, any read/write keys and project-id is passed as is. If SYS$*DLOC$ is not assigned to the calling run, then any project-id different from the user project-id is obscured by slashes. The read/write keys are also slashed except when the corresponding program control table (PCT) read/write inhibit bits are clear or the read/write keys are blank.

22.3.4. ALTERING MAIN ITEM (DBIT$)

This function allows the user to change certain bit settings in word 11, sector 0 of the main item (see Figures 22–5 and 22–8). The format of the packet used in the MSCON$ request (see 22.3) is:

<table>
<tr><th></th><th>35</th><th>17</th><th>5   0</th></tr>
<tr><td>Word 0</td><td colspan="2"></td><td>30_8</td></tr>
<tr><td>1</td><td colspan="3" rowspan="2">filename</td></tr>
<tr><td>2</td></tr>
<tr><td>3</td><td>setting-bits</td><td colspan="2">control-bits</td></tr>
</table>

Words 1 and 2

Specifies the file whose entry is being changed.

Word 3

If 2^{17} is set, use the contents of 2^{35} for the disabled status.

If 2^{16} is set, use the contents of 2^{34} for the facility reject status.

If 2^{15} is set, use the contents of 2^{33} for the facility warning status.

If 2^{14} is set, use the contents of 2^{32} for the SECURE reject status.

If 2^{13} is set, use the contents of 2^{31} for bad main item extension sector status, main item word 12, bit 2^{31}.

If 2^{7} is set, use the contents of 2^{25} for the write-only status.

If 2^{6} is set, use the contents of 2^{24} for the read-only status.

If any of the bits 2^{34}, 2^{33}, or 2^{32} in main item sector 0, word 11 is set, then bit 2^{35} is set. Conversely, if bit 2^{35} is cleared, then bits 2^{34}, 2^{33}, and 2^{32} are cleared.

22.3.5. ALTERING BACKUP FILE ENTRIES (DBACK$)

This function allows the user to change certain fields pertaining to backup files. In performing this function, the DBACK$ request clears word 10 (time stamp of first write after backup) of main item sector 0, makes certain that the BKUP bit is set in the main item descriptor, and clears the file-changed bit is PCFID1 of the PCT facilities item.

If the number of backup reels in word 3 of the packet is zero, the function will undo a previous DBACK$ by clearing the BKUP bit in the main item descriptor and setting the backup reel count to 0 in main item sector 1. For example, this allows a @SECURE,R control statement to reset a file's backup status when vital backup information has been lost on a recovery boot.

The format of the packet used in the MSCON$ request (see 22.3) is:

Word	35	29	23	17	11	5 0
0						31_8
1 2	filename					
3	nbr-of-backup-reels					
4	time-of-backup-creation					
5	media-codes	tape-mode-codes	reel-break	total-nbr-of-1800-word-text-blocks		
6	tape-noise-constant		starting-file-position-of-first-backup-reel		nbr-of-words-written-in-last-block	
7	first-backup-reel-nbr					
8 ↓ n	Any number of additional reel number words may follow, subject to length of packet					

Words 1 and 2

Specifies the file whose entry is being changed.

Words 3 – 7

These words replace equivalent words in section 1 of the main item (see Figure 22–6 for additional information on these words).

Word n

Packet length = 7 + nbr of backup reels

22.3.6. ALTERING LAPSE ENTRIES (DLAPS$)

This function allows the user to enter new lapse entries or to alter information pertaining to lapse entries (see Figures 22–6 and 22–7).

If words 3 and 4 of the input packet are both nonzero, the DLAPS$ request enters the new lapse entry in the next available location, adds 1 to the lapse entry count in sector 1 of the main item, and sets the has-lapse-entries bit in the main item descriptor.

If word 3 of the input packet is zero, the DLAPS$ request zeros out the lapse entry count and all existing lapse entries, and clears the has-lapse-entries bit in the main item descriptor.

If word 4 of the input packet is zero and word 3 is nonzero, the DLAPS$ request uses the value in word 3 to change the lapse entry count in sector 1 of the main entry. This, for example, allows a @SECURE,R control statement to update the lapse entry count after examining the main item extension sectors on a recovery boot.

The format of the packet used in the MSCON$ request (see 22.3) is:

		S6
Word 0		32_8
1	filename	
2		
3	new-lapse-entry-or-zero	
4		

Words 1 and 2

Specifies the file whose entry is being changed.

Words 3 and 4

The lapse entry.

22.3.7. CHANGING UNLOAD TIME (DUNLD$)

This function stores the unload time (nonzero for unload, zero for load) in sector 0 of the main item (word 10). When unload time is nonzero, the DUNLD$) request ensures that the unload bit is set. When unload time is zero, the DUNLD$ request ensures that the unload bit is cleared and that the this-file-has-been-changed bit in the PCT is cleared. This function is primarily for use by the SECURE processor (see Section 19).

Filename is used to specify the file whose entry is being changed.

The format of the packet used in the MSCON$ request (see 22.3) is:

		S6
Word 0		33_8
1	filename	
2		
3	unload-time	

22.3.8. CHANGING MAXIMUM CYCLE RANGE (DCYC$)

This function stores the new maximum range in sector 0 of the lead item (S3 of word 9 – see Figure 22–3), and deletes any F-cycles which no longer fall within the new maximum range. Filename is used to specify the file whose entry is being changed. The format of the packet used in the MSCON$ request (see 22.3) is:

		S6
Word 0		34_8
1	filename	
2		
3		max-range

22.3.9. CHANGING READ/WRITE KEYS (DKEY$)

This function allows the user to change file read and write keys. To do this, the old read and write keys must be furnished in the packet along with either or both of the new keys. If either key is not to be changed, a word of slashes (words 5 or 6 of the packet) is used to indicate this. To remove a key altogether, blanks are used. If either old key is incorrect, the DKEY$ request is rejected, a ABORT$ exit is taken, and control is not returned to the calling program. Filename is used to specify the file whose entry is being changed. The format of the packet used in the MSCON$ request (see 22.3) is:

Word		S6
0		35_8
1	filename	
2		
3	old-read-key (blanks if none)	
4	old-write-key (blanks if none)	
5	new-read-key-or-blanks (slashes if no change)	
6	new-write-key-or-blanks (slashes if no change)	

22.3.10. CHANGING BLOCK BUFFERING EOF SECTOR ADDRESS (DBB$)

This function allows the user to change the block buffering EOF sector address (word 12, Figure 22–5) and the block and item size information (word 9, Figure 22–5) entries in sector 0 of the main item. Filename is used to specify the file whose entry is being changed. The format of the packet used in the MSCON$ request (see 22.3) is:

Word	35	23	17	5 0
0				36_8
1	filename			
2				
3		block-buffering-EOF-sector-addr		
4	block-size		item-size	

22.3.11. MODIFYING FILE IDENTITY (DREG$)

This function was designed for use by the SECURE processor (see Section 19). It allows the user to change certain fields in the lead and main items of catalogued files. The DREG$ request sets descriptor bit 33 in word 13 of the main item sector 0 (see Figure 22–5) signifying that this item was created by a SECURE REGISTER function.

If the SYS$*DLOC$ file is assigned to the calling run, the DREG$ request uses the information found in the caller packet to alter certain other fields. The time of catalogueing is stored into main item word 19 (see Figure 22–5). If the time of last reference (word 9) equals a nonzero value, it is stored into word 18 of the main item. If the account-nbr field (words 5 and 6) contains anything other than binary zeros or Fieldata blanks (05_8), it is used to overlay the main item account number. If the project-id (words 3 and 4) contains anything other than binary zeros, it replaces the project-id in both the lead and main items of this field. If the nbr-of-times-assigned (word 7, H2) contains anything other than binary zero, it replaces the total-nbr-of-times-file-has-been-assigned field in the main item. The format of the packet used in MSCON$ request (see 22.3) is:

	35	17	5	0
Word 0			37_8	
1	filename			
2				
3	project-id			
4				
5	account-nbr			
6				
7	0	nbr-of-times-assigned		
8	time-of-cataloguing			
9	time-of-last-reference			

22.3.12. LINK INSERTION FOR REMOVABLE DISC PACKS (DLINK$)

This function allows a privileged requestor to insert the links from the fixed mass storage MFD items to the removable disc packs associated with the file. This function is utilized by the SECURE processor (see Section 19) during the restore process of removable disc packs.

Filename is used to specify the file whose entry is being changed.

The format of the packet used in the MSCON$ request (see 22.3) is:

	35	17	5	0
Word 0			40_8	
1	filename			
2				
3	initial-reserve	nbr-of-FASTRAND-granules		
4	nbr-of-pack-id-links-or-zero			
5	highest-granule-assigned	nbr-of-FH-432-granules		
6	link-to-MFD-for-first-pack			
7	link-to-MFD-for-second-pack			
8 ↓				
n	link-to-MFD-for-nth-pack			

22.3.13. ADDING GRANULE ITEMS (DADD$)

This function allows a privileged requestor to add sectors to a chain of granule items on fixed mass storage for a file.

Filename is used to specify the file whose entry is being changed.

The format of the packet used in the MSCON$ request (see 22.3) is:

Word	35	17	5 0
0			41_8
1	filename		
2			
3		nbr-of-granules-for-this-request	
4	S main-storage-addr-of-first-granule-table-for-this-request		
5 ↓			
n	main-storage-addr-of-last-granule-table-for-this-request		

Word 4-n

Address of a 28-word buffer within the calling program contains a granule table (see Figure 22–9) that is to be added to the granule table chain. The correct links (words 0 and 1 of the granule table) will be added by the MSCON function. If the sign bit is set (2^{35} set to 1) no new MFD sector will be allocated by MSCON but the last sector in the chain will be overwritten with the information supplied. This allows for the fact that mass storage files always have the first granule table allocated whether or not the file has been written into.

22.3.14. MONITORING MASS STORAGE AVAILABILITY (MSALL$)

This function enables a user program to monitor the availability of mass storage by transferring an image of the current FASTRAND-formatted availability table (FATBL) into a user-supplied buffer, followed by (for those sites with nonremovable disc storage) an image of the executive pack-id table (XPKID).

Upon successful completion of this function, packet word 2, H1 contains the length in words of FATBL, and packet word 2, H2 contains the total number of image words transferred; that is, the combined lengths of FATBL and XPKID.

The format of the packet used in the MSCON$ request (see 22.3) is:

Word	35	17	5 0
0			60_8
1	buffer-length-of-buffer-into-which-current-FASTRAND-formatted-availability-tables-are-written	addr-of-buffer-into-which-current-FASTRAND-formatted-availability-tables-are-written	
2	*length-of-FATBL*	*total-nbr-of-words-transferred*	

The format for FATBL is as follows:

		H1	H2
Word	0	*starting-addr-of-FASTRAND-II-or-III-mass-storage-unit-tables*	*starting-addr-of-FASTRAND-II-or-III-mass-storage-logical-subsystem-table*
	1		
	2	*starting-addr-of-FH-432-drum-storage-unit-tables*	*starting-addr-of-FH-432-drum-logical-subsystem-table*
	3	*starting-addr-of-FH-880-drum-storage-unit-tables*	*starting-addr-of-FH-880-drum-logical-subsystem-table*
	4	*starting-addr-of-FH-1782-drum-storage-unit-tables*	*starting-addr-of-FH-1782-drum-logical-subsystem-table*
	5	*starting-addr-of-8414-disc-storage-unit-tables*	*starting-addr-of-8414-disc-logical-subsystem-table*
	6	*starting-addr-of-8440-disc-storage-unit-tables*	*starting-addr-of-8440-disc-logical-subsystem-table*
	7	*starting-addr-of-unitized-channel-storage-unit-tables*	*starting-addr-of-unitized-channel-storage-logical-subsystem-table*

Start of logical subsystem table for FH-432 drum:

	H1	S4	T3
A		*nbr-of-logical-subsystems-containing-FH-432-drums*	
A+1	*tracks-available*	*nbr-of-units-in-first-subsystem*	*positions-available*
A+2	*tracks-available*	*nbr-of-units-in-second-subsystem*	*positions-available*
A+3			
A+n	*tracks-available*	*nbr-of-units-in-nth-subsystem*	*positions-available*

Start of logical subsystem table for FH-880, FH-1782, 8414, FASTRAND and U.C.S. have the same format as the logical subsystem table for FH-432 drum.

The FH-880 and FH-1782 drum, FASTRAND mass storage, 8414 disc, and unitized channel storage logical subsystem tables have the same format as the FH-432 drum logical subsystem table.

Start of unit tables for FH-432 drum:

Word	35	23	17	0
B	*total-nbr-FH-432-drums*	*maximum-tracks-obtainable*		
B+1	*remaining-nbr-of-tracks-available*			
B+2	*maximum-positions-obtainable*		*remaining-positions-available*	

Start of unit table 1:

	S1	S2	S3	S4	T3
B+3		*MFD-track-addr*			
B+4	*physical-subsystem-nbr-for-first-FH-432-drum*		*physical-unit-nbr*		
B+5	*remaining-tracks-available*			*logical-subsystem-nbr*	*remaining-positions-available*
B+6	*addr-of-master-bit-table*				

Start of unit table 2:

	S1	S2	S3	S4	T3
B+7		*MFD-track-addr*			
B+8	*physical-subsystem-nbr-for-second-FH-432-drum*		*physical-unit-nbr*		
B+9	*remaining-tracks-available*			*logical-subsystem-nbr*	*remaining-positions-available*
B+10	*addr-of-master-bit-table*				

Start of unit table n:

S1	S2	S3	S4	T3
	MFD-track-addr			
physical-subsystem-nbr-of-nth-FH-432-drum		*physical-unit-nbr*		
remaining-tracks-available			*logical-subsystem-nbr*	*remaining-positions-available*
addr-of-master-bit-table				

Start of unit tables for the 8414 disc:

	S1	S2	S3	S4	T3
Word 0		*MFD-track-addr*			
1	*physical-subsystem-addr*		*physical-unit-nbr*	*current-assign-count-for-this-unit* *	
2	*remaining-tracks-available*			*logical-subsystem-nbr*	*remaining-positions-available*
3	*complete-unit-assign-flag*	*declared-fixed-mass-storage*	*addr-of-master-bit-table*		
4	*pack-id*				

* Applicable only to removable discs.

The FH-880 and FH-1782 drum, FASTRAND mass storage, and unitized channel storage unit tables have the same format as the FH-432 drum unit table.

The format of XPKID is as follows:

Word	H1	H2
0		*maximum-nbr-of-fixed-disc-packs-on-the-system*
1	*subsystem/unit-of-first-fixed-disc-pack-on-system*	*reserved*
2	*subsystem/unit-of-second-fixed-disc-pack-on-system*	*reserved*
3		
↓		
n	*subsystem/unit-of-nth-fixed-disc-pack-on-system*	*reserved*

XPKID will contain the subsystem/unit numbers of disc packs which the executive will consider as permanent mass storage (nonremovable).

22.4 MSCON$ STATUS CONDITIONS

When MSCON$ returns control to the user, register A0 contains the original packet address in H2, possible error status codes in bits $2^{29} - 2^{18}$ and an exec-indicator in bits $2^{34} - 2^{30}$.

35	34 30	29 24	23 18	17 0
S	*exec-indicator*	*I/O-error-indicator*	*error-status-code*	*packet-addr*

The value of the exec-indicator is supplied by MSCON$ for use by the SECURE processor.

If bit 2^{35} = 0 and bits $2^{29} - 2^{18}$ = 0, this signifies normal completion of the requested function.

If bit 2^{35} = 0 and bits $2^{23} - 2^{18}$ contain 01, the 01 is a special status code returned by the DREAD$ function (see 22.3.3), signifying that the end of the user buffer has been encountered and there are more directory items to be returned.

If bit 2^{35} = 1 and the error status code is 24_8, the value in bits $2^{29} - 2^{24}$ is the I/O status code received by MSCON$.

If bit 2^{35} = 1 and the error status code is other than 24_8 the possible status codes are:

20_8 – Wrong MSCON$ function code is user packet.

21_8 – User packet not within program limits.

22_8 – Referenced file is not assigned to this user.

23_8 – User is referencing a temporary file.

25_8 – User buffer not within program limits.

26_8 – User is referencing a nonexistent start item (returned by the DREAD$ function, see 22.3.3).

27_8 – User buffer area not large enough (returned by functions DLINK$, see 22.3.12; DADD$, see 22.3.13; and MSALL$, see 22.3.14).

30_8 – The function requires a main item extension sector which is not in existence for this file (returned by functions DLAPS$, see 22.3.6, and DLINK$, see 22.3.12).

31_8 – The referenced disc unit has been marked down or reserved (returned by the DGETP$ function, see 22.3.2).

32_8 – The requested pack-id cannot be found in the FASTRAND-formatted availability table (returned by the DGETP$ function, see 22.3.2).

33_8 – The output file initial reserve is too small to contain the current total of system directory items (returned by the DGET$ function, see 22.3.1).

34_8 – The cumulative total of system directory items has dynamically expanded beyond the capacity of the output file (returned by the DGET$ function, see 22.3.1). This situation differs from that described for status code 33_8, in that, in this instance DGET$ has been in process and directory items have been output to the file.

22.5. DIRECTORY ITEM FORMATS

The MFD contains directory items for each catalogued file in the system. The content and format of the MFD are subject to change without prior notice. Directory item description is as follows:

Bits 2^{35} through 2^{32} in word zero of each directory item have the following significance:

Bit	Description
35	When set to 0, indicates that word 0 of this sector contains a link address to the next sector.
34–32	001_2 – Search item 010_2 – Lead item 100_2 – Main item 000_2 – Granule tables, or sectors 1-n of main item, or sector 1 of lead item.

Figures 22–2 through 22–10 illustrate the format of the various items and their sectors

Word	35	34	33	32	31	0
0	*U*	*0*	*0*	*1*	*link-to-next-sector-if-U = 0*	
1	*qualifier*					
2						
3	*filename*					
4						
5	*link-to-lead-item*					
6 ↓ 27	*Up to four more five-word entries (identical format to words 1 through 5)*					

Figure 22–2. Search Item

Word	35	34	33	32	31	S2	S3	S4	T3
0	*U*	*0*	*1*	*0*	*link-to-sector-1-of-lead-item-if-U = 0*				
1	*qualifier*								
2									
3	*filename*								
4									
5	*project-id*								
6									
7	*read-key*								
8	*write-key*								
9	*medium*					*count*	*maximum-range*	*current-range*	*current-absolute-F-cycle*
10	*link-to-main-item-for-next-F-cycle-to-be-catalogued*								
11	*link-to-main-item-of-catalogued-file-or-0*								
12	*link-to-main-item-of-catalogued-file-or-0*								
13 ↓ 26									
27	*link-to-main-item-of-catalogued-file-or-0*								

Figure 22–3. Lead Item – Sector 0

Word 9

medium — Type of device which F-cycles describe:

- $1_8 - 17_8$ – Magnetic tape
- $30_8 - 37_8$ – Mass storage
- 75_8 – 8414 removable disc

count — Current number of F-cycles in the lead item

maximum-range — Maximum number of F-cycles permitted for the file

current-range — Range of absolute F-cycles currently in this lead item; (current-absolute-F-cycle) (lowest-absolute-F-cycle) +1

current-absolute-F-cycle — The absolute F-cycle number whose link is in word 11 or would be in word 11 if it existed.

Word 11

Link to the main item which describes the current-absolute-F-cycle. If this absolute cycle does not exist, the entry is 0.

Word 12

Link to the main item which describes a file whose absolute F-cycle is one less than the current-absolute-F-cycle. If this absolute F-cycle does not exist, the entry is 0.

Word 27

Link to the main item which describes a file whose absolute F-cycle is 16 less than the current-absolute-F-cycle. If this absolute F-cycle does not exist, the entry is 0.

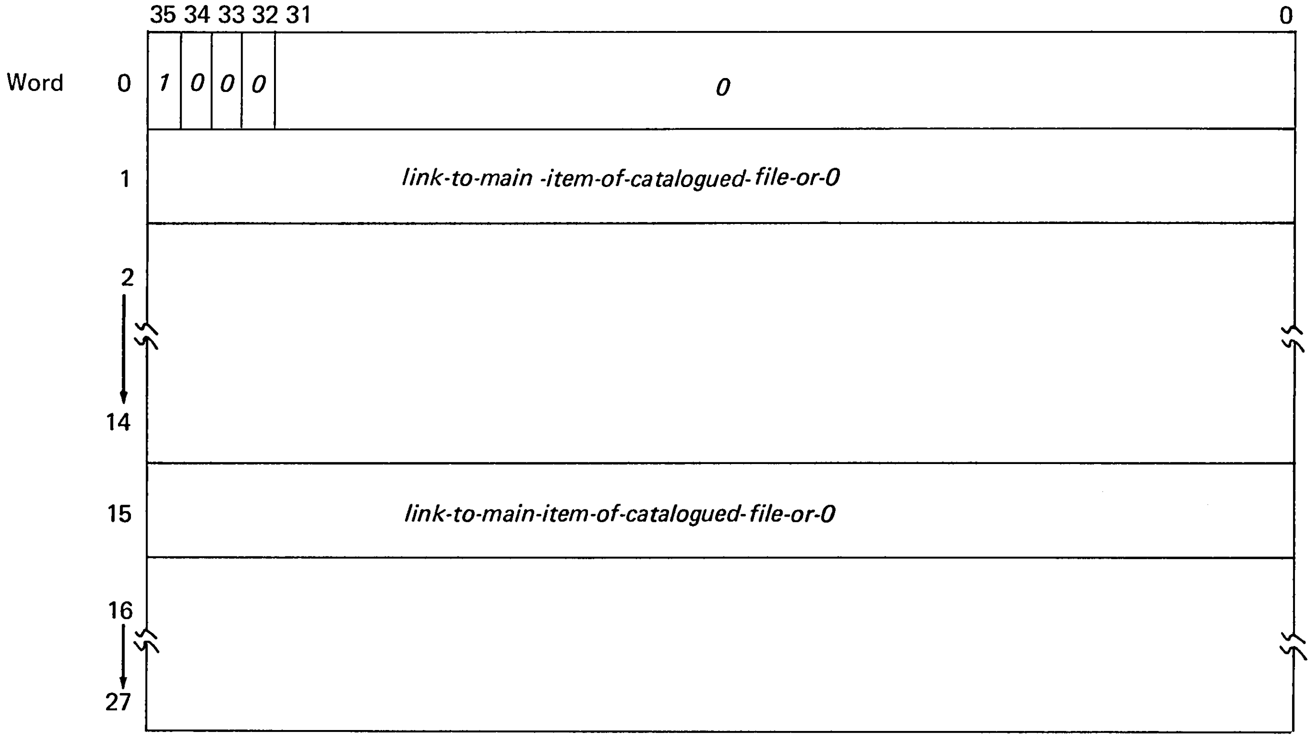

35 34 33 32 31
0
Word
0
1
0
0
0
0
1
link-to-main -item-of-ca talogued- file-or-0
2
14
15
link-to-main-item-of-catalogued- file-or-0
16
27

Figure 22–4. Lead Item – Sector 1

Word 1

Link to the main item which describes a file whose absolute F-cycle is 17 less than the current-absolute-F-cycle. If this absolute F-cycle does not exist, the entry is 0.

Word 2

Link to the main item which describes a file whose absolute F-cycle is 31 less than the current-absolute-F-cycle. If this absolute F-cycle does not exist, the entry is 0.

35 29 23 17 0

Word					
0	*U*	*1*	*0*	*0*	*link-to-granule-item-if-U = 0*
1 2	*qualifier*				
3 4	*filename*				
5 6	*project-id*				
7 8	*account-nbr*				
9	*block-size (negative if variable)*	*item-size (zero if item is variable in length)*			
10	*time-of-first-write-following-unload-or-backup-time (TDATE$ format - see 4.5.2)*				
11	*disable-flag*	*link-to-lead-item*			
12	*descriptor-flag*	*block-buffering-EOF-sector-addr*			
13	*PCHAR-flag*	*link-to-sector-1-of-main-item*			
14	*symbiont-link-to-initial-SMOQUE-entry*				
15	*symbiont-save-area-addr*	*total-nbr-of-times-this-file-has-been-assigned*			
16	*run-id-or-EXPOOL-addr*				

Figure 22–5. Mass Storage File Main Item — Sector 0 (Part 1 of 2)

	35	29	23	17	11	0
17	*media-codes*	*inhibit-flags*	*nbr-of-runs-currently-assigned-to-this-F-cycle*		*absolute-F-cycle-nbr*	
18	*date-and-time-current-assignment-started-or-last-assignment-ended (TDATE$ format - see 4.5.2)*					
19	*date-and-time-of-cataloguing (TDATE$ format - see 4.5.2)*					
20	*initial-nbr-of-granules-reserved-for-this-assignment*			*nbr-of-granules-of-FASTRAND-mass-storage-currently-containing-file*		
21	*maximum-nbr-of-granules*			*reserved-for-expansion*		
22	*highest-granule-nbr-assigned*			*nbr-of-granules-of-FH-432-storage-currently-containing-file*		
23	*highest-granule-nbr-written*			*nbr-of-granules-of-FH-880-storage-currently-containing-file*		
24				*nbr-of-granules-of-FH-1782-storage-currently-containing-file*		
25				*nbr-of-granules-of-8414-disc-currently-containing-file*		
26				*nbr-of-granules-of-8440-disc-storage-currently-containing-file*		
27				*nbr-of-granules-of-unitized-channel-storage-currently-containing-file*		

Figure 22–5. Mass Storage File Main Item – Sector 0 (Part 2 of 2)

Word 9

block-size — Used by block buffering package

item-size — Used by item handler

Word 10

time-of-first-write following-unload-or-backup-time — If word 12, bit 35 = 1, then word 10 contains unload time. If word 12, bit 35 = 0, then word 10 contains time of first write after backup file was created or all zeros.

Word 11

disable-flag — 1100_2 – Facilities reject (file disabled because it was destroyed)

1010_2 – Facilities warning (file disabled because it was an incomplete write)

1001_2 – SECURE processor reject (file disabled because it was rejected by the SECURE processor)

Word 12

descriptor-flag

These bits, when set, indicate:

Bit 35 – Unloaded

34 – Backed up

33 – File created by means of the SECURE processor REGISTER command

32 – Has lapse entries

31 – Bad main item sector 1

30 – Set if new item format (created on or after level 26 of the executive)

29 – Unused

28 – Communication between recovery and MFD maintenance routines for removable disc files. Extension sectors of the main item have been changed on permanent storage. This allows the update of extension sectors on the disc packs.

27 – Removable disc file

26 – Unused

25 – File created by @CAT control statement

24 – File is to be dropped

Word 13

PCHAR-flag

These bits, when set, indicate:

Bit 35 – Position granularity

34 – Disc pack granularity

33 – Word addressable

32 – Prefers high speed drum

Word 17

media-codes

Indicate type of peripheral containing file. Same as equipment codes, see Appendix E.

inhibit-flags

These bits, when set, indicate inhibit options on @ASG control statements as follows:

Bit 29 –	@ASG,G	Guarded file
28 –	@ASG,V	Unload inhibit
27 –	Not @ASG,P	Private file
26 –	@ASG,X	Exclusive use
25 –	@ASG,W	Write-only
24 –	@ASG,R	Read-only

Figure 22–5. Mass Storage File Main Item – Sector 0 (Part 3 of 3)

<table>
<tr><th>Word</th><th>35</th><th>34</th><th>33</th><th>32</th><th>31</th><th>29</th><th>23</th><th>17</th><th>11</th><th>0</th></tr>
<tr><td>0</td><td><i>U</i></td><td><i>0</i></td><td><i>0</i></td><td><i>0</i></td><td colspan="6"><i>link-to-sector-2-of-main-item-if-U = 0</i></td></tr>
<tr><td>1
2</td><td colspan="10"><i>qualifier</i></td></tr>
<tr><td>3
4</td><td colspan="10"><i>filename</i></td></tr>
<tr><td>5</td><td colspan="10"><i>'*NO.1*'</i></td></tr>
<tr><td>6</td><td colspan="10"><i>link-to-sector-0-of-main-item</i></td></tr>
<tr><td>7</td><td colspan="6"><i>nbr-of-backup-reels</i></td><td colspan="2"><i>nbr-of-two-word-lapse-entries</i></td><td colspan="2"><i>absolute-F-cycle-nbr-of-this-file</i></td></tr>
<tr><td>8</td><td colspan="10"><i>date-and-time-of-backup-file-creation (TDATE$ format - see 4.5.2)</i></td></tr>
<tr><td>9</td><td colspan="5"><i>media-codes</i></td><td><i>tape-mode-codes</i></td><td><i>reel-break</i></td><td colspan="3"><i>total-nbr-of-1800-word-text-blocks</i></td></tr>
<tr><td>10</td><td colspan="6"><i>tape-noise-constant</i></td><td colspan="2"><i>starting-file-position-of-first-backup-reel</i></td><td colspan="2"><i>nbr-of-words-written-in-last-block</i></td></tr>
<tr><td>11</td><td colspan="10"><i>reel-nbr-of-first-reel-of-backup-file</i></td></tr>
<tr><td>12</td><td colspan="10"><i>reel-nbr-of-second-reel-of-backup-file</i></td></tr>
<tr><td>13
14</td><td colspan="10"><i>first-lapse-entry-or-all-zeros</i></td></tr>
<tr><td>15
16</td><td colspan="10"><i>second-lapse-entry-or-all-zeros</i></td></tr>
</table>

Figure 22–6. Mass Storage File Main Item – Sector 1 (Part 1 of 2)

17	*disc-pack-entries-control-word*
18 19	*first-disc-pack-entry*
20 21	*second-disc-pack-entry*
22 23	*third-disc-pack-entry*
24 25	*fourth-disc-pack-entry*
26 27	*fifth-disc-pack-entry*

Figure 22–6. Mass Storage File Main Item – Sector 1 (Part 2 of 2)

Word 9

media-codes — Indicates type of peripheral containing file. Same as equipment codes, see Appendix E.

tape-mode-codes — These codes are:

1_8 – User to recover

2_8 – Software translate

4_8 – Hardware translate

10_8 – Low density

20_8 – Medium density

30_8 – High density

40_8 – Even parity

reel-break — A constant which indicates the number of reels per group if file was backed up onto multiple tape reel groupings because of the enormous size of the file.

Words 13 – 14

first-lapse-entry-or-all-zeros — Word 13: Time of first write following backup file creation
Word 14: Time of recovery when backup copy becomes primary copy

Date and time is given in TDATE$ format (see 4.5.2).

Word 17

disc-pack-entries-control-word — Bits 35 – 12: Reserved
11 – 0: Number of disc pack entries

Word 18 – 19

first-disc-pack-entry — Word 18: Pack-id of the disc pack (six characters in Fieldata)
Word 19: Link to main item on disc pack

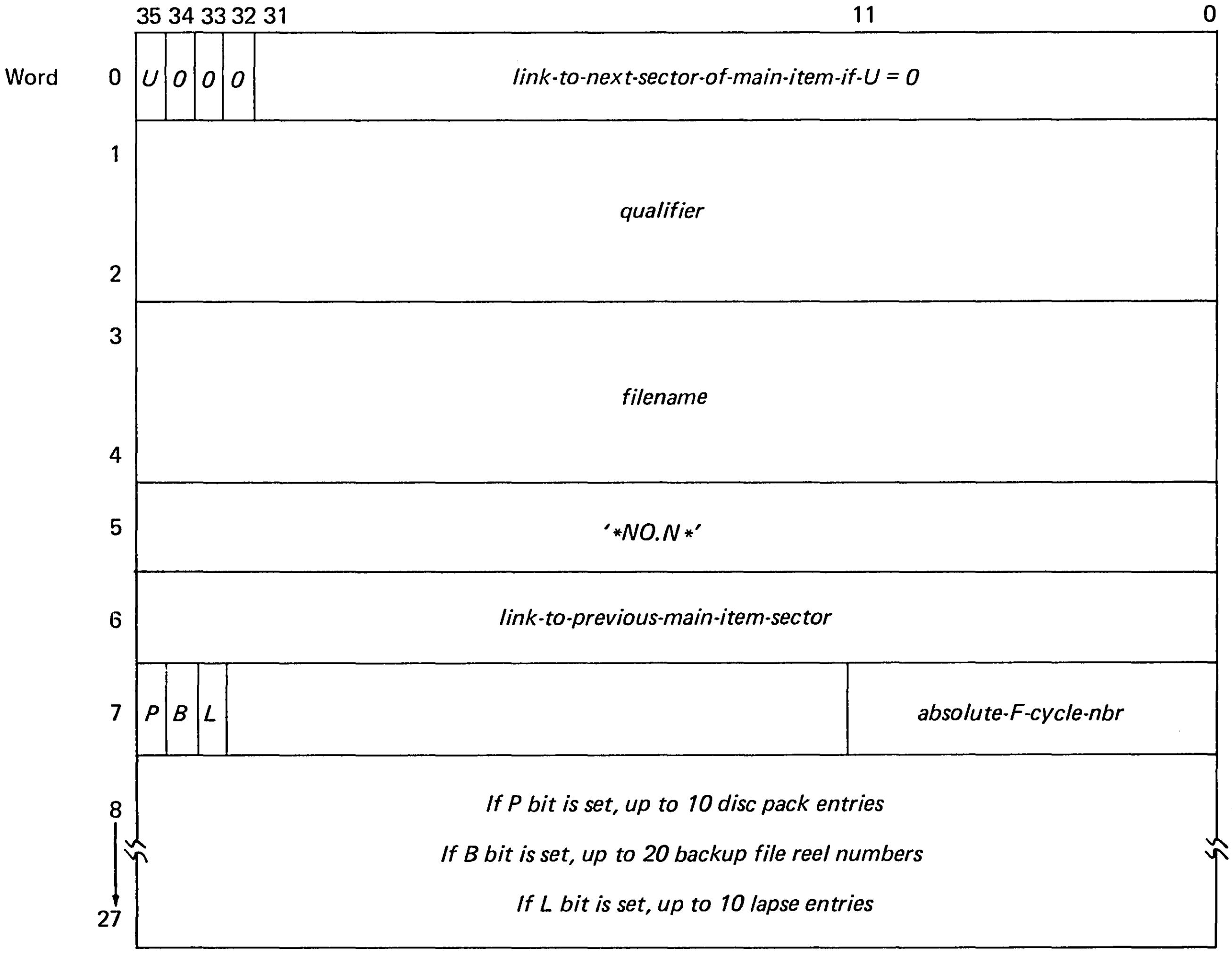

35 34 33 32 31
11
0
Word
0
U
0
0
0
link-to-next-sector-of-main-item-if-U = 0
1
2
qualifier
3
4
filename
5
'*NO.N*'
6
link-to-previous-main-item-sector
7
P
B
L
absolute-F-cycle-nbr
8
27
If P bit is set, up to 10 disc pack entries
If B bit is set, up to 20 backup file reel numbers
If L bit is set, up to 10 lapse entries

Figure 22–7. Main Item – Sectors 2–n

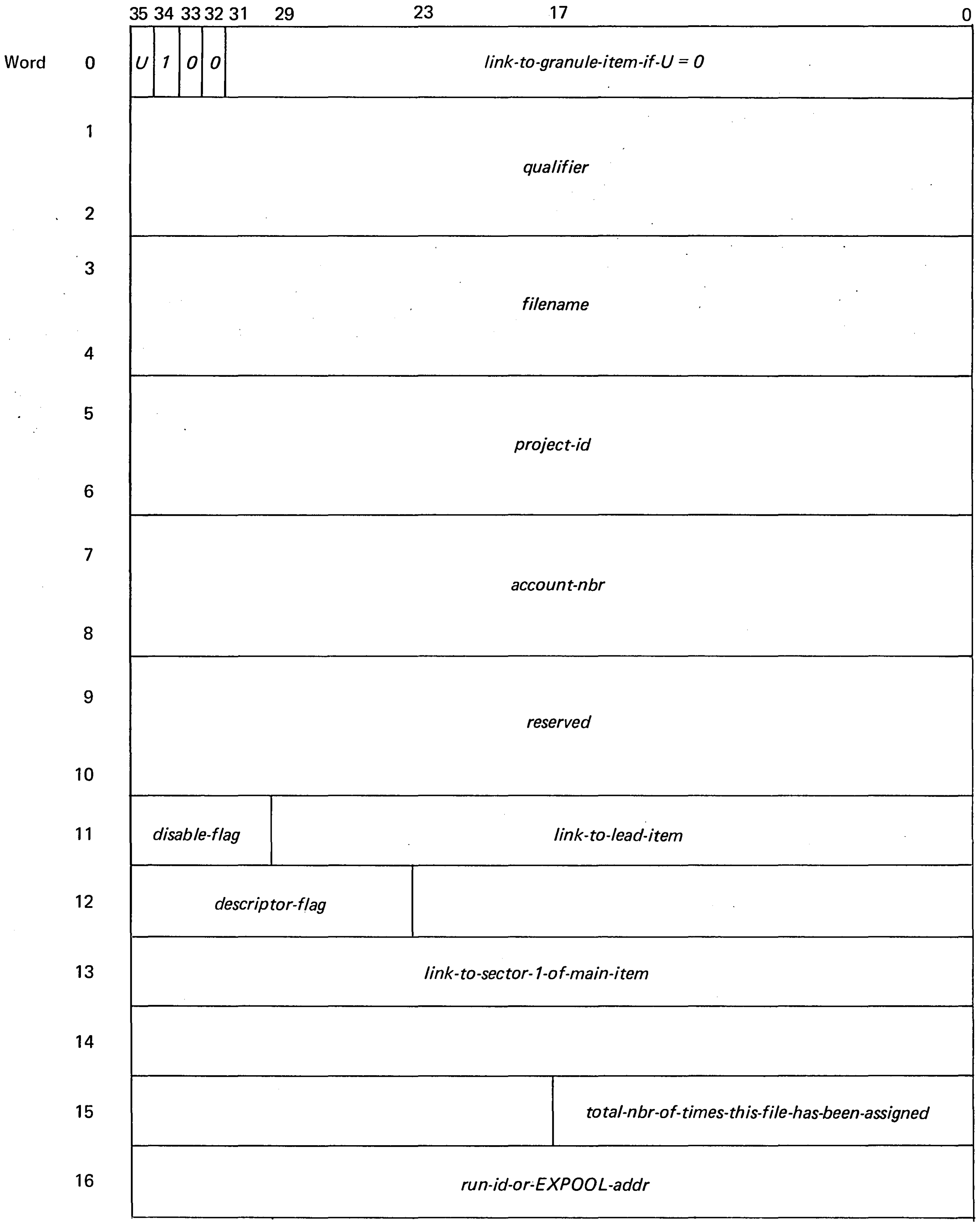


Figure 22–8. Tape File Main Item – Sector 0 (Part 1 of 2)

	S1	S2	S3	S4	T3
17	*media-codes*	*inhibit-flags*	*nbr-of-runs-currently-assigned-to-this-F-cycle*		*absolute-F-cycle*
18	*date-and-time-current-assignment-started-or-last-assignment-ended (TDATE$ format - see 4.5.2)*				
19	*date-and-time-of-cataloguing (TDATE$ format - see 4.5.2)*				
20	*current-reel-index-of-tapes-labeled-and-checked*			*nbr-of-reels-catalogued*	
21	*12/16-tape-modes*		*0*	*tape-mode-codes*	*noise-constant*
22	*reserved*				
23					
24					
25					
26	*reel-nbr-of-first-reel-catalogued*				
27	*reel-nbr-of-second-reel-catalogued*				

Figure 22–8. Tape File Main Item – Sector 0 (Part 2 of 2)

Word 11

disable-flag

1100_2 – Facilities reject (file disabled because it has been destroyed)

1010_2 – Facilities warning (file disabled because of an incomplete write)

1001_2 – SECURE processor reject (file disabled because it was rejected by the SECURE processor.

Word 12

descriptor-flag

These bits, when set, indicate:

Bit 35 – Unused

34 – Backed up

33 – Backup file cannot be read

32 – Unused

31 – Bad main item sector 1

30 – New item format (created on or after level 26 of executive)

29 – 26 – Unused

25 – File created by @CAT control statement

24 – File is to be dropped

Word 13

link-to-sector-1-of-main-item

Sector 1 of main item for tape files is identical in format to that for mass storage files (see Figure 22–6) except there are no disc pack entries.

Word 17

media-codes

Indicates type of peripheral containing file. Same as equipment codes, see Appendix E.

inhibit-flags

These bits, when set, indicate inhibit options on @ASG control statements as follows:

Bit 29	ASG,G	Guarded file
28	Unused	
27	Not @ASG,P	Private file
26	Unused	
25	@ASG,W	Write only
24	@ASG,R	Read only

Word 21

12/16-tape-modes

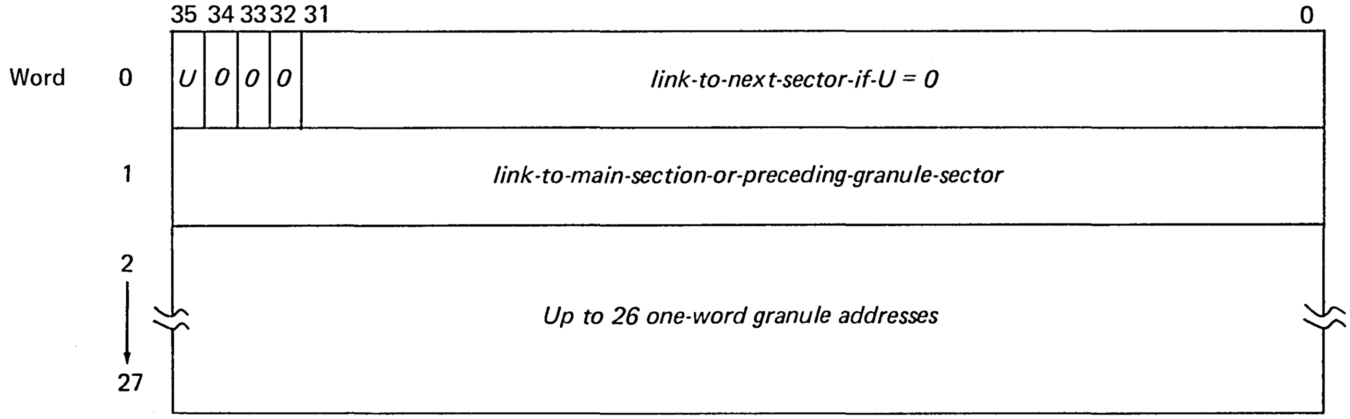
This field contains 0 if tape units are not UNISERVO 12 or 16 tape units. Mode is indicated by the following bit settings:

Bit 35 – High density
34 – Dual-density unit
33 – Eight-bit packed
32 – Six-bit packed
31 – Quarter-word
30 – Unit translate, BCD/EBCDIC
29 – XS3/ASCII in MSA
28 – XS3/EBCDIC in MSA
27 – Fieldata/ASCII in MSA
26 – Fieldata/EBCDIC in MSA
25 – Unused
24 – Unit translate data converter

tape-mode-codes

Applicable to tape units other than UNISERVO 12 or 16 tape units (field contains 0 if tape units are UNISERVO 12 or 16 tape units). The modes are:

40_8 – Even parity
30_8 – High density
20_8 – Medium density
10_8 – Low density
4_8 – Hardware translate
2_8 – Software translate


35 34 33 32 31
0
Word 0
U
0
0
0
link-to-next-sector-if-U = 0
1
link-to-main-section-or-preceding-granule-sector
2
27
Up to 26 one-word granule addresses

Word 1

There is the possibility that there are more than 26 granule addresses; therefore, there can be more than one granule item. Each granule item lists 26 granule addresses. When there is more than one granule item, the second, third, and so forth, granule items are linked back to the preceding granule, and the first granule item provides the link back to the main item.

Figure 22–9. Mass Storage File Granule Item

		35	34	33	32	31	0
Word	0	*U*	*0*	*0*	*0*	*link-to-next-sector-if-U = 0*	
	1	*link-to-main-section-or-preceding-granule-sector*					
	2 ↓ 27	*Up to 25 reel numbers*					

Word 1

There is the possibility that there are more than 25 reel numbers; therefore, there can be more than one granule item. Each granule item lists 25 reel numbers. When there is more than one granule item, the second, third, and so forth, granule items are linked back to the preceding granule item, and the first granule item provides the link back to the main item.

Figure 22—10. Tape File Granule Item

23. LOGGING AND ACCOUNTING

23.1. INTRODUCTION

Extensive logging and accounting capabilities are incorporated into the executive system to collect information pertaining to each run and to certain general executive actions, such as I/O error logging. The information can be used for accounting and general postprocessing purposes. A summary accounting file containing information about each account is maintained and updated automatically by the system at the termination of each run. In addition, a master log of all logging and accounting information is produced.

Two utility routines are provided: BILLER (see 23.7) is used to process the summary account file; LOGFED (see 23.8) is used to process the master log file. Both routines are provided only as examples to the installation manager to aid him in writing routines unique to his installation. The BILLER routine serves as a minimum provision to inserting account numbers and their associated site-manager specified parameters. The LOGFED routine serves only to list the contents of one or more master log files.

Figure 23-1 is a block diagram of the logging and accounting process.

23.2. LOG ENTRY INITIATION AND CONTROL

Log entries are initiated by the executive at various significant points throughout a run in order to obtain and preserve in the master log a chronological record of the activities of the system. In addition, the user run can enter messages into the master log. Console messages are also entered. The total information gathered is of value not only for accounting (billing) purposes, but also to capture the unique actions of a particular run and to evaluate system performance.

The information generated and placed in the temporary and later the master log fall into the following categories:

- Run initiation
- User-specified log control statement (@LOG) from either a run stream or submitted via a CSF$ request.
- Program termination
- I/O errors
- Console activity
- Tape labeling information
- Checkpoint/restart information
- Facility usage
- Run termination
- Catalogued mass storage file usage

The log control routine controls the flow of all logging information that is generated to the temporary log file. As each logging request is made, the log control routine chains the new log entry information by run in the temporary log file.

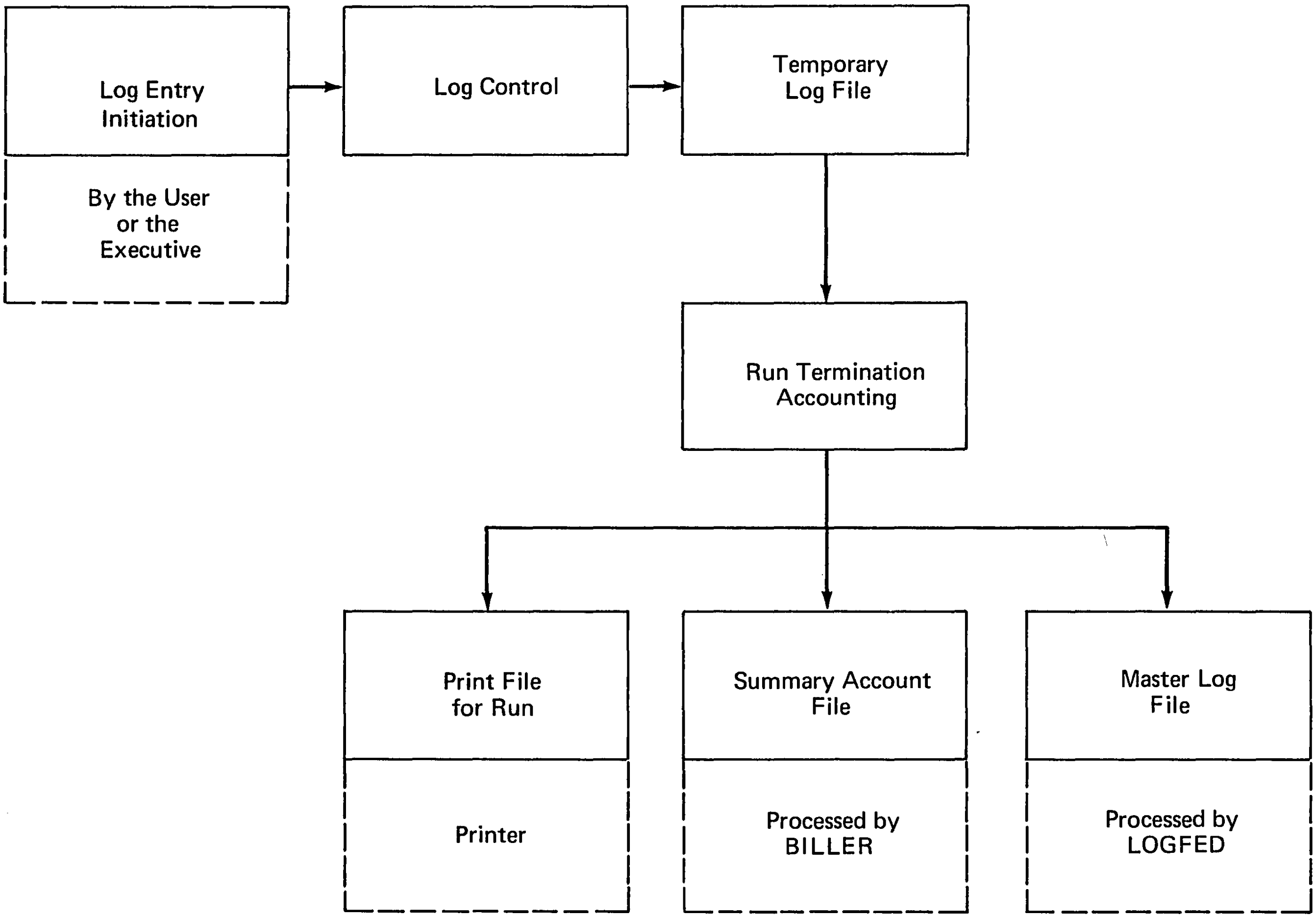


Figure 23–1. Logging and Accounting Process, Block Diagram

23.3. PRINT FILE OUTPUT

At each run termination, information which pertains to the run is placed at the end of the PRINT$ file by the run termination accounting routine. The information presented is as follows:

- Run identity
- Control language log statements
- Console messages pertaining to the run
- Executive request log statements
- Project identity
- Account number
- Total run time
- Pages of printing applicable to run
- Number of cards read in and punched out
- Time and date of run initiation
- Time and date of run termination

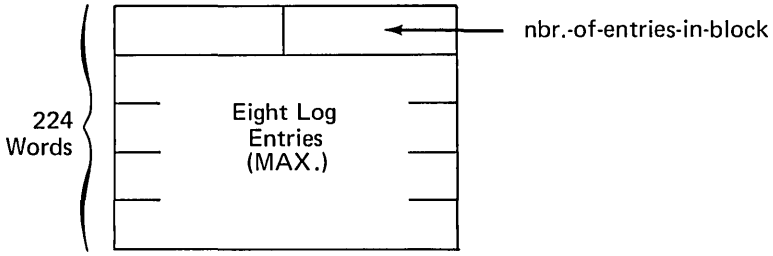
23.4. SUMMARY ACCOUNT FILE CREATION AND UPDATING

A second function of the run termination accounting routine is to update the totals in the summary account file for the given account. This is done in the course of creating the master log file, since each log entry is moved from the temporary log file to the master log file.

The summary account file, called ACCOUNT$, is a FASTRAND-formatted mass storage file which is set up at system initialization and permanently assigned to the executive system. Totals are kept until cleared by a unique BILLER or a user-supplied billing routine, or until the file is replaced during initial system loading. Procedures for executing a billing routine are usually established by the installation manager. See 23.7 for a discussion of BILLER routine provided. For each allowable account number, a 56-word block exists in the summary account file, which is described in detail in 23.6.3.

23.5. MASTER LOG FILE CREATION AND CONTROL

The run termination accounting routine inserts entries into the master log file (SYSTEM$LOG$). This file may be catalogued either on tape or on FASTRAND-formatted mass storage (specified at system generation time). The file is made up of a number of 224-word blocks each containing up to a maximum of eight log entries. The number of these blocks depends upon the length of the file set at system generation time. The blocks have the following format:



This file is initially catalogued by the executive during the initial boot from tape. Log entries are written on the file until the preset maximum length of the file is encountered. Then the current F-cycle of the file is released and a new one is started. This switch enables the user to assign and read the file just released. An operator keyin also causes the executive to switch files. In case of I/O errors on either tape or mass storage, the executive performs an automatic switch to obtain a new file. When the F-cycle limit is reached, the system drops the oldest cycle to make room for the newest F-cycle. If the file is on tape and an end-of-tape is encountered, TSWAP$ (see 7.2.7) is called for a new reel of tape and an F-cycle change does not occur.

At system initialization time, the file is catalogued with a (–0) relative F-cycle. If the file (–0) is on Fastrand-formatted mass storage, the user should use caution in assigning and reading this file since the executive will be writing on it at the same time. However, if the file is on tape, the user is not able to assign and read this F-cycle until the executive releases it. When a switch occurs, the executive frees the current (–0) F-cycle and catalogues a new (–0) F-cycle. The former (–0) F-cycle becomes relative (–1) and is available to the user. The end-of-file (EOF) sentinel is one word of all 7's (octal) at the beginning of the block. If the file is on tape, an end-of-tape sentinel is one word of all 6's (octal) at the beginning of the block.

The particular log entries found in the master log are described in detail in 23.6. The LOGFED routine is described in 23.8.

23.6. FILE FORMATS

23.6.1. BASIC NOTATION

All values in the summary account and master log file entries are binary unless it is obviously a symbolic name such as program name, version name, run-id, qualifier, filename, project-id, or account number, in which case it is in Fieldata that is left-justified and space filled. Several items in all entries have a common format and are described here to facilitate easier understanding of the following entry formats. The common items are as follows:

- date and time of xxx — This is the format of time and date as received from a call to TDATE$ (see 4.5.2)
- message — Refers to a string of Fieldata characters, the maximum length of which is specified in the entry.

23.6.2. SUMMARY ACCOUNT FILE STRUCTURE

The structure of the summary account file is illustrated in Figure 23-2. The relationship between the 56-word table of contents and the corresponding 56-word account number entries is shown. For more detailed knowledge of the manipulation of this file, study must be made of portions of the executive accounting routine (see 23.2 through 23.6) and the routine BILLER (see 23.7).

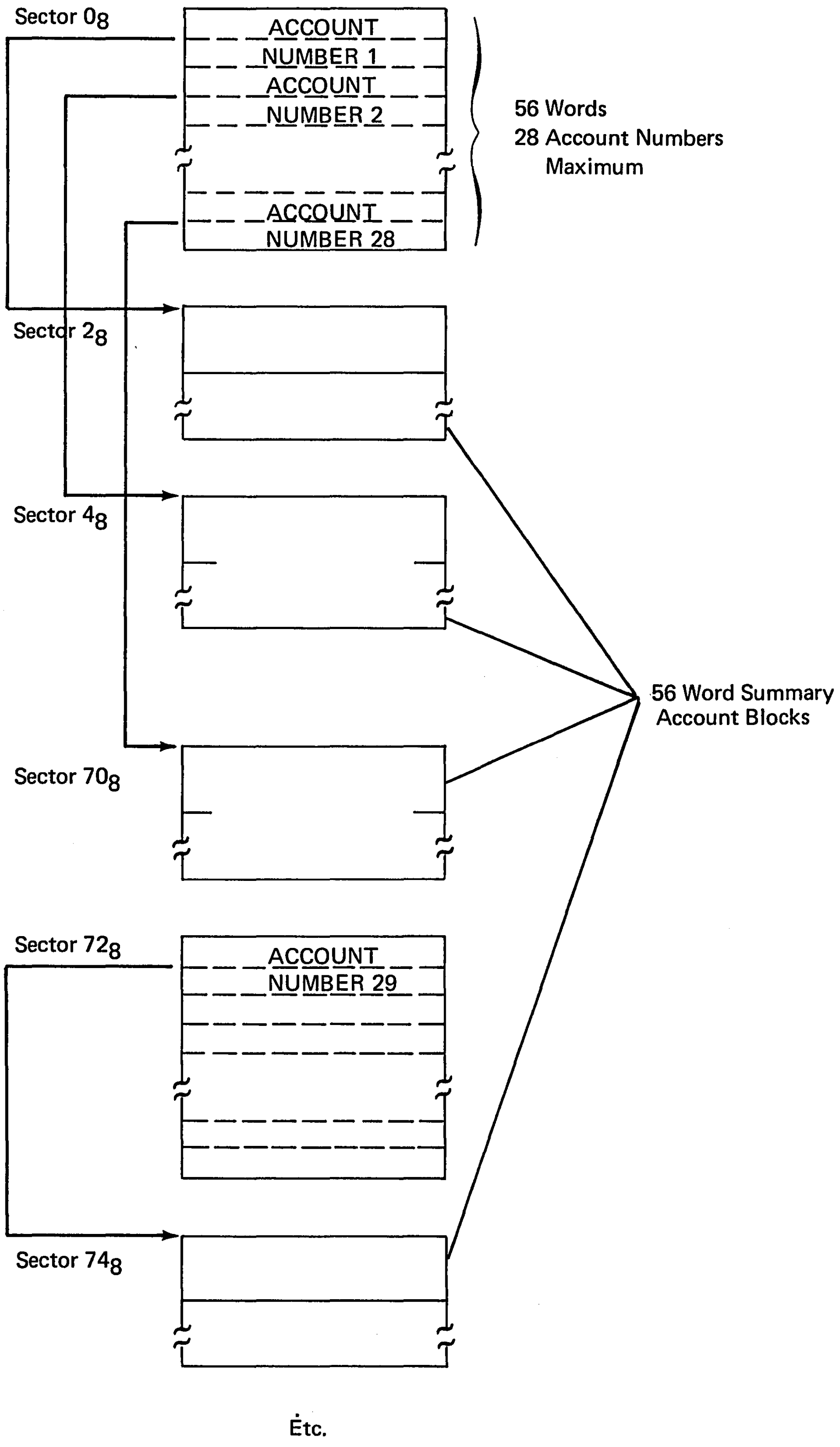


Figure 23–2. Summary Account File Format

23.6.3. SUMMARY ACCOUNTING FILE ENTRY FORMAT

The format of the summary accounting file entry is:

Word	S1	S2	S3	S4	S5	S6
0	*account-number*					
1						
2	*highest-allowable-priority*	*priority to be-used-when-none-is specified*	*nonzero-if-deadline-allowed*	*nonzero-if-real-time-allowed*	*nonzero-if-entry-added-by-operator*	
↓						
10	*first-entry-time (DATE$ format)* *					
11						
12	*last-entry-time (DATE$ format)* *					
13						
14	*entry-last-cleared (DATE$ format)* *					
15						
16	*total-number-of-runs*					
17	*total-compute-time-used*					
18	*total-elapsed-time-of-runs*					
19	*total-cards/images-in*					
20	*total-cards-out*					
21	*total-lines-out*					

** See 4.5.1 for DATE$ format*

22	*not used*
23	
24	
25	*track-seconds-of-FH-432-usage*
26	*track-seconds-of-FH-880-usage*
27	*track-minutes-of-FASTRAND-usage*
28	
29	total-seconds-UNISERVO-16-tape-assigned
30	total-seconds-UNISERVO-12-tape-assigned
31	total-seconds-UNISERVO-VIII-C-tape-assigned
32	total-seconds-UNISERVO-VI-C-tape-assigned
33	total-seconds-UNISERVO-IV-C-tape-assigned
34	total-seconds-UNISERVO-III-A-tape-assigned
35	total-seconds-UNISERVO-II-A-tape-assigned
36	total-seconds-FH-432-assigned
37	total-seconds-FH-880-assigned
38	total-seconds-FH-1782-assigned

39	
40	total-seconds-card-subsystem-assigned
41	total-seconds-paper-tape-subsystem-assigned
42	total-seconds-printer-assigned
43	total-seconds-UNIVAC-1004-card-processor-assigned
44	total-seconds-CTS-assigned
45	total-seconds-WTS-assigned
46	total-seconds-CTMC-assigned
47	
55	

23.6.4. MASTER LOG ENTRY FORMATS

The different types of log entries which may appear in the master log are described in paragraphs 23.6.4.1 through 23.6.4.13.

Each of the entry formats contains a word that provides either the date and time of the log entry or the date and time of some program action. The format of this word is:

mm/dd/yy/seconds-after-midnight

where the year (yy) is module 64.

23.6.4.1. CONTROL STATEMENT LOG ENTRIES

Log entries specified by the @LOG control statement are placed in the master log file in the order in which they occur. The entry format is:

	S1	S2		S6
Word 0	*entry-type (1)*	*message-length*		*nbr-of-log-entries-in-224-word-block*
	message *(22-word maximum)*			
25	*date-and-time-of-log-entry*			
26				
27	*run-id**			

*Run-id word is zero if the entry (and the block) pertains to the executive rather than a specific run.

23.6.4.2. FACILITY USAGE LOG ENTRIES

Whenever the configuration of a run is changed by assigning or freeing a tape or arbitrary device file, an entry is made in the master log file. The entry format is:

	S1	S2	S3	S4	S5	S6	
Word 0	*entry-type (2)*	*nbr-of-wds-that-all-entries-occupy*				*nbr-of-entries-in-a-224-word-block*	
1	*subsystem-nbr*		*unit-nbr*		*equipment-code* *		Entry 1
2	*date-and-time-of-@ASG-or-@FREE*						
3							Entry 2
4							
↓							↓
25	*date-and-time-of-log-entry*						
26							
27	*run-id*						

* See Appendix E for equipment codes.

23.6.4.3. CATALOGUED MASS STORAGE FILE USAGE ENTRY

Whenever a catalogued mass storage file is freed (using a @FREE control statement or at run termination), an entry is made in the master log. The entry format is:

	S1	S2	S3	S4	S5	S6
Word 0	*entry-type (3)*	*nbr-of-wds-needed-by-entry*				*nbr-of-entries-in-224-word-block*
1	*qualifier*					
2						
3	*filename*					
4						
5	*project-id*					
6						
7	*account-nbr*					
8						
9	*equipment-code*	*flag-2*	*current-assigns*		*absolute-F-cycle*	
10	*date-and-time-of-@FREE*					
11	*date-and-time-of-cataloguing*					
12	*date-and-time-assigned*					
13	*Final count of file granules on mass storage devices having equipment codes 30_8 through 37_8. Count taken at @FREE. See Appendix E for equipment codes.*			*Original count of file granules on mass storage devices having equipment codes 30_8 through 37_8. Count taken at @ASG. See Appendix E for equipment codes.*		
14						
15						

16		
17		
18		
19		
20		
21		
22		
23		
24		
25	*date-and-time-of-log-entry*	
26		
27	*run-id*	

where:

current assigns — The number of file assignments during the run (other users excluded).

flag-2

40_8 – Position granularity

20_8 – Private file

10_8 – File is being dropped

4_8 – File was assigned with exclusive use

2_8 – Write-only file

1_8 – Read-only file

23.6.4.4. PROGRAM TERMINATION LOG ENTRY

For each program in the run, termination information is entered in the master log. The entry format is:

		S1	S2	S3	S4	S5	S6
Word	0	entry-type (4)	nbr-of-wds-in-entry				nbr-of-entries-in-224-word-block
	1	program-name					
	2						
	3	version-name					
	4						
	5	date-and-time-of-program-initiation					
	6	date-and-time-of-program-termination					
	7	run-time (200-μsec-increments)					
	8	I-bank-length			D-bank-length		
	9	program-type			last-reentry-addr		
	10	condition-word					
	11	nbr-of-I/O-references					
	12	data-words-transferred					
	13						

25	date-and-time-of-log-entry
26	
27	run-id

23.6.4.5. RUN TERMINATION LOG ENTRY

At the completion of each run, termination information is entered in the master log. The entry format is:

Word	S1	S2			S6
0	entry-type (5)	nbr-of-wds-in-entry			nbr-of-entries-in-224-word blocks
1 2	account-nbr				
3 4	project-id				
5	date-and-time-of-program-initiation				
6	date-and-time-of-program-termination				
7	cards-in			cards-out	
8	priority	print-line-count			
9	estimated-run-time (millisecs)				
10	actual-run-time (millisecs)				

11	Track seconds count for mass storage devices having equipment codes 30_8 through 37_8 (see Appendix E for equipment codes). Track-seconds = (nbr-of-tracks) x (nbr-of-secs-used) Applicable to temporary files and expansion of catalogued files assigned to this run.	
12		
13		
14		
15		
16		
17		
18		
19	total-nbr-sectors-allocated/released	total-nbr-allocation/release-calls
20	granule-table-r/w	total-master-file-directory-control-calls
25	date-and-time-of-log-entry	
26		
27	run-id	

where:

granule-table-r/w — A count of the read/write operations used to maintain the granule table.

total-directory-control-calls — The total count of references to the executive cataloguing routines to support all catalogued files in the run.

23.6.4.6. I/O ERROR LOG ENTRY

A record of all I/O errors is kept in the master log. A count of valid references to a unit is maintained in main storage and when an error occurs, this information along with the error information is placed in the master log. The reference count is cleared to zero at that time. Note that after a predetermined number of retries (number of retries is device dependent), all of which fail, the operator is notified and operator intervention is required. The entry format is:

	S1	S2	S3	S4	S5	S6
Word 0	entry-type (6)	nbr-of-words-in-entry				
1	MSA-device-flag		equipment-code	unit-nbr	subsystem-nbr	
2	EI-status-word (1)					
3	EI-status-word (2)-or-zero					
4	EI-status-word (3)-or-zero					
5			nbr-of-refs-since-last-error			
6				IOC-or-MSA-nbr	IOC-or-MSA-channel-nbr	CPU-channel-nbr
7	EF-word (1)					
8	EF-word (2)					
9 ↓ 20	Up to 12 additional EF words					
20 ↓						
24	reel-nbr (tape-only)					
25	date-and-time-of-log-entry					
26						
27	run-id					

where:

MSA-device-flag	A zero indicates a nonMSA device. A nonzero indicates a MSA device.
equipment-code	See Appendix E

23.6.4.7. CONSOLE LOG ENTRIES

Each console message is placed in the master log. At run termination, every message pertaining to the run is printed at the end of the program listing. The entry format is:

	S1	S2			S5	S6
Word 0	entry-type (7)	nbr-of-wds-in-msg (+1)				nbr-of-entries-in-224-word-block
1						
2					msg-nbr	
	message (22-word maximum)					
25	date-and-time-of-log-entry					
26						
27	run-id					

23.6.4.8. CHECKPOINT LOG ENTRY

When a checkpoint is used in a run, an entry is made in the master log with pertinent information concerning the checkpointed run. The entry format is:

Word	S1	S2		S6
0	entry-type (8)	nbr-of-wds-in-msg		nbr-of-entries-in-224-word-block
	message (24-word maximum)			
25	date-and-time-of-log-entry			
26				
27	run-id			

23.6.4.9. RUN INITIATION LOG ENTRY

When a run is opened, an entry is made in the master log with the pertinent information concerning the run. The entry format is:

Word	S1	S2	S3	S4	S5	S6
0	entry-type (9)	nbr-of-wds-in-entry				nbr-of-entries-in-224-word-block
1	A	priority	start-time (in minutes)		deadline-time (in minutes)	
2	estimated-pages-out			estimated-cards-out		
3	run-id (new)					
4	run-id (old)					
5	project-id					
6						
7	account-nbr					
8						
9						
10	device-association			estimated-run-time-(secs)		
↓						
25	date-and-time-of-log-entry					
26						
27	run-id					

Word 1

A The possible values are:

10_8 – T option is specified on @RUN control statement

4_8 – P option is specified on @RUN control statement

2_8 – C option is specified on @RUN control statement

1_8 – S option is specified on @RUN control statement

This entry is always the first one in the first block of a series of contiguous blocks in the master log file for a given run. The run-id field (new) contains the same identity as word 27 of each entry for the subject run.

23.6.4.10. CONSOLE REPLIES LOG ENTRY

Replies to console type and read messages are placed in the master log. The replies as well as the type and read messages are printed at the end of the program listing. The entry format is:

		S1	S2		S5	S6
Word	0	entry-type (10)	nbr-of-words-in-msg (+1)			nbr-of-entries-in-224-word block
	1				msg-nbr	
	↓	message (11-word maximum)				
	25	date-and-time-of-log-entry				
	26					
	27	run-id				

23.6.4.11. LOG KEYIN ENTRY

The entry format is:

		S1	S2		S6
Word	0	entry-type (11)	nbr-of-wds-in-msg		nbr-of-entries-in-224-word block
	↓	message (9-word maximum)			
	25	date-and-time-of-log-entry			
	26				
	27	run-id			

23.6.4.12. UNSOLICITED KEYIN LOG ENTRY

The entry format is:

Word	S1	S2	S3	S4	S5	S6
0	entry-type-(12)	nbr-of-wds-in-msg (+1)				nbr-of-entries-in-224-word-block
1				actual-keyin-in-Fieldata-format (left-justified)		
↓	message (23-word maximum)					
25	date-and-time-of-log-entry					
26						
27	run-id					

23.6.4.13. TAPE LABELING LOG ENTRY

When the tape labeling feature of the executive is used, log entries are made in the master log. These entries contain pertinent information concerning allocation and release of tape reels, and errors encountered during tape labeling. The entry format is:

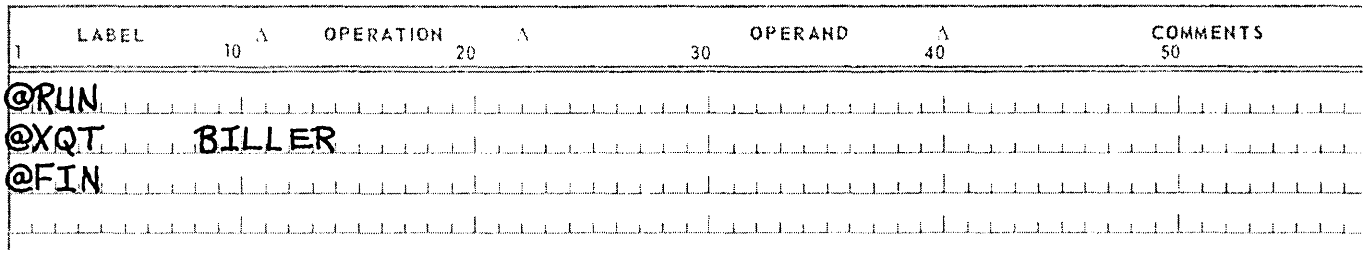
Word	S1	S2	
0	entry-type (13)	nbr-of-wds-in-msg	
↓	message (24-word maximum)		
25	date-and-time-of-log-entry		
26			
27	run-id		

23.7. BILLING ROUTINE (BILLER)

23.7.1. GENERAL DESCRIPTION

The billing routine accesses the summary account file (SYS$*ACCOUNT$) which is generated by the executive accounting routine. For each account number, it performs various updating functions as well as generates a listing of some of the information found in the file.

The billing routine (BILLER) is executed in the following manner:



LABEL
OPERATION
OPERAND
COMMENTS
1
10
20
30
40
50

```
@RUN
@XQT     BILLER
@FIN
```

The routine is self-contained in that it assigns the ACCOUNT$ file internally, and it requires no library routines for allocation.

The billing routine performs the following basic functions:

- Reads each entry in the account file
- Resets each item to the cleared state (while the file is locked out from summary accounting)
- Prints each total for the various facilities used by the account
- Flags entries added by the operator
- Totals each entry type for all accounts
- Prints a summary for the entire system since the previous billing
- Inserts new entries into the file
- Deletes entries from the file

23.7.2. CONSTRAINTS FOR USER–IMPLEMENTED BILLING ROUTINES

It is most important that the implementor of a user billing routine understand the certain portions of the sample billing routine. This is necessary to ensure that the user routine does not cause destructive action.

The code for BILLER has three parts to its organization:

(1) Card Read and Card Editor Element

(2) ACCOUNT$ Interface Element

(3) Output Editor Routine

The user is fairly free concerning modifications to the output editor. Modifications can be made to the card editor, but the user should be aware of what is acceptable input to the interface element. Improper modification of either the card editor or the interface routine will destroy ACCOUNT$ or lock the file against executive access.

23.7.3. INITIALIZING AND CHAINING OF ACCOUNT ENTRIES

BILLER accomplishes many different tasks such as inserting account numbers and parameters into ACCOUNT$, purging information about all or any account numbers, reading information about account numbers between purges, and removing account numbers from the file.

The four basic commands (card inputs) available with BILLER are INSERT, REMOVE, PURGE, and READ,

23.7.3.1. INSERT COMMAND

The INSERT command is used to enter new information. The format of the INSERT command is:

```
    INSERT    acct-id,param-1    param-2    param-3    param-4
```

Where acct-id is the account number and param indicates the location of parameters.

The word INSERT may begin in any column, but at least one blank must follow it. If the account number is more than 12 characters long or is omitted, the command is rejected. A comma must separate the account number from all following fields. The number of blanks before or after the commas is completely arbitrary.

Four parameters are allowed. These parameters can be listed in any order, but for the purpose of description, are presented in the following order:

param-1 If deadline times are permitted for the specified account number, code the following.

DL

If deadline times are not permitted, omit this parameter and the routine assumes a 'no'. The message DEADLINE ALLOWED or DEADLINE NOT ALLOWED is printed after the INSERT card is interpreted.

param-2 This parameter specifies the highest allowable priority (A-Z) for a run using this account number. If omitted, M is assumed. The priority is specified as follows

MP = z

where z is the priority letter. A message is printed noting which priority level.

param-3 This parameter specifies the priority to be used if none is specified on the @RUN control statement. If omitted, M is assumed. The priority is specified as follows

BP = z

param-4 where z is a priority letter.

This parameter specifies what level of real time, if any, is allowed for programs in a run using this account number. The allowable levels are 2 through 35 and the means of specifying a real time level is as follows

RTL = y

where y is a value from 2 to 35. If omitted, programs running with this account number are not allowed to go real time.

Example:

```
    LABEL          Δ     OPERATION      Δ                OPERAND        Δ                 COMMENTS
1              10                   20                30                 40                  50
INSERT 399103,MP=C    ,    DL    ,    RTL  =  25
```

Insert account number 399103, with highest priority C, run – blank, priority M, deadline permitted, and real time-highest level 25.

BILLER prints this command and the following messages:

```
MAXIMUM PRIORITY = C

BLANK PRIORITY = M

REAL–TIME LEVEL = 25

DEADLINE ALLOWED
```

If the account number already exists in the file, a message is printed noting this and an automatic PURGE (see 23.7.3.2) is performed. The new parameters are inserted into the file.

23.7.3.2. REMOVE, PURGE, and READ COMMANDS

The format for each command is identical:

```
READ   acct-id-1,...,acct-id-n
```

The command identifying word READ, PURGE, or REMOVE may start in any column. It must be followed by at least one blank before the first (acct-id). Separate all following account numbers with commas. Again any number of blanks can precede or follow the commas.

The function of each command is as follows:

Command	Description
REMOVE	This command instructs BILLER to delete the indicated account numbers and perform an automatic purge of information associated with those accounts.
PURGE	This command instructs the routine to purge ACCOUNT$ of information concerning the specified accounts. After the purge, the information is set to zero and accumulation is started over. The absence of account numbers purges the entire file.
READ	This command is identical to the PURGE command except that the information is not cleared to zero. This is a read-out of the file for given account numbers or of the entire file if no account numbers are specified.

Examples:

```
   LABEL        OPERATION           OPERAND          COMMENTS
 1         10           20         30          40         50
1. REMOVE   399104    ,      399105

2.  READ    399106

3. PURGE  399107,399108

4.  PURGE

```

1. All information (already accumulated and being accumulated) for accounts 399104 and 399105 is deleted.

2. All information being accumulated for account 399106 is listed.

3. All information being accumulated for accounts 399107 and 399108 is deleted and the accumulation of these accounts is restarted.

4. All information in the ACCOUNT$ file is deleted.

23.7.3.3. NO INPUT SPECIFIED TO BILLER

If no card input is provided for BILLER, it purges all account numbers in the same manner as if a PURGE command with no account numbers specified was executed.

23.7.4. PRINTER OUTPUT

In addition to the printer output already described, the following information is output for each account number:

(1) TIME/DATE OF FIRST ENTRY

The time and date when ACCNTG encountered the first run initiation type of log entry for this account number; in other words, the time the first run started.

(2) TIME/DATE OF LAST ENTRY

The time and date when ACCNTG last encountered a run termination type of log entry for this account number.

(3) TIME/DATE ENTRY LAST CLEARED

The time and date when a billing routine last cleared the information in the summary account block for this account number.

(4) TOTAL RUNS PROCESSED

(5) TOTAL COMPUTE TIME OF RUNS

(6) TOTAL CARDS READ

(7) TOTAL CARDS PUNCHED

(8) TOTAL PAGES GENERATED

(9) SUBSYSTEM USAGE

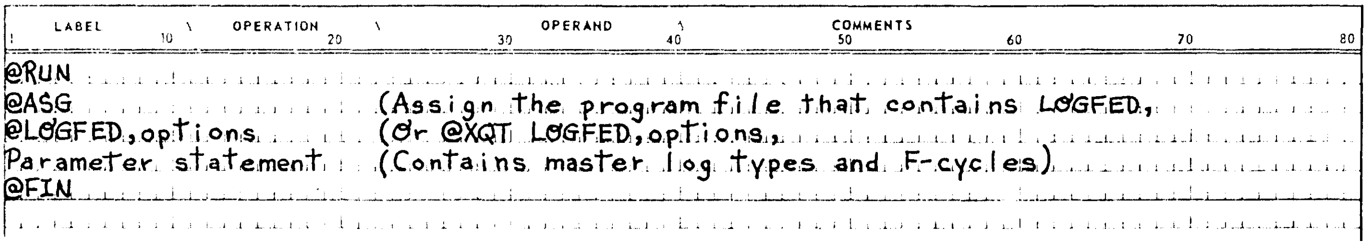
The elapsed time that an I/O facility was assigned to this account.

An example of the output of this routine is:

```
                                   ***  399101  ****

TIME/DATE OF FIRST ENTRY           11:58:84 – OCT 17, 1968
TIME/DATE OF LAST ENTRY            14:54:45 – OCT 17, 1968
TIME/DATE ENTRY LAST CLEARED       14:57:72 – OCT 17, 1968
TOTAL RUNS PROCESSED               8
TOTAL COMPUTE TIME OF RUNS         00:03:16.617
TOTAL CARDS READ                   3323
TOTAL CARDS PUNCHED                0
TOTAL PAGES GENERATED              252
SUBSYSTEM USAGE:  8C   02:29:59
```

23.8. LOG FILE EDITOR (LOGFED)

LOGFED is a routine designed to extract and edit master log entries created by the executive accounting routine. Any number of entries may be extracted, depending on the parameter statement used. LOGFED may be called as either a processor or a user program. The general calling sequence is given below:


LABEL
OPERATION
OPERAND
COMMENTS
1
10
20
30
40
50
60
70
80

```
@RUN
@ASG                          (Assign the program file that contains LOGFED,
@LOGFED,options               (or @XQT LOGFED,options,
Parameter statement           (Contains master log types and F-cycles)
@FIN
```

Available options are:

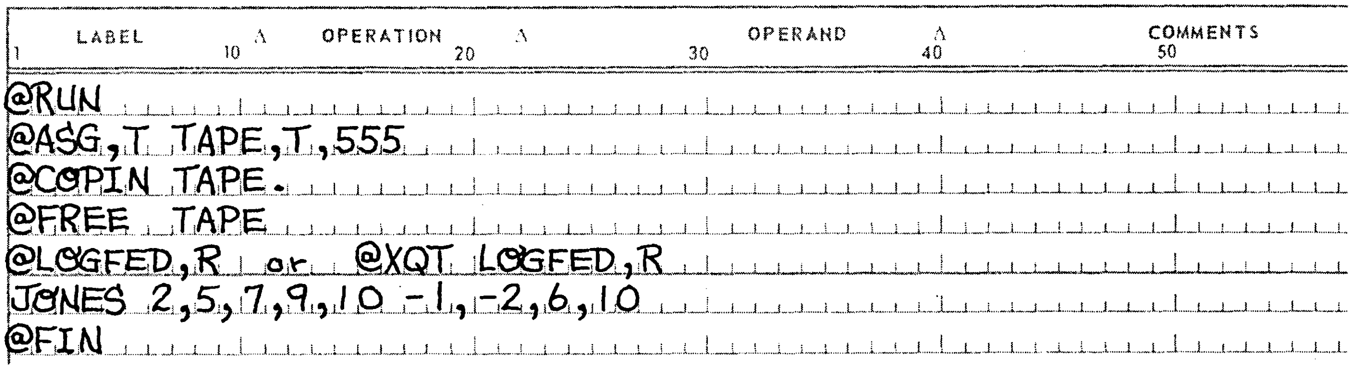
A – Extract log entries for a specific account number.

R – Extract log entries for a specific run-id.

If no options are present, entries are extracted with no regard to account number or run-id.

The parameter statement takes on three general forms depending on the presence or absence of options. The following examples best explain the parameter statement. Assume a run-id of JONES and an account number of 399126. Log entry types 2, 5, 7, 9, and 10 are to be extracted from relative F-cycles –1, –2, and absolute F-cycles 6 and 10. Assume that the absolute element LOGFED is part of a program file on tape numbered 555.

Example 1: Extract log entries for RUNID only (R option). The run deck would appear as follows:

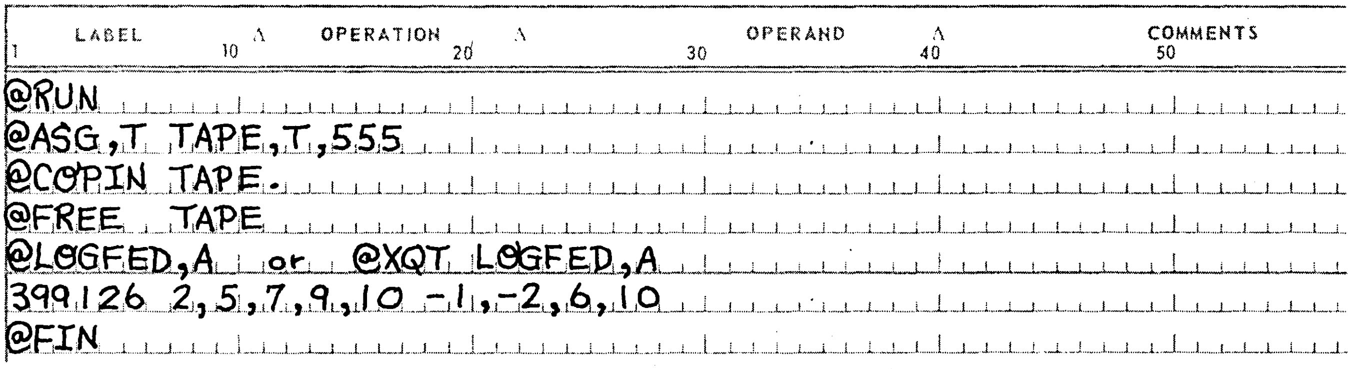


LABEL OPERATION OPERAND COMMENTS
1 10 20 30 40 50

```
@RUN
@ASG,T TAPE,T,555
@COPIN TAPE.
@FREE  TAPE
@LOGFED,R  or  @XQT LOGFED,R
JONES 2,5,7,9,10 -1,-2,6,10
@FIN
```

The parameter statement must be coded as shown; that is, the run-id must begin in column 1, be followed by a blank followed immediately by the log entry types separated by commas with no intervening blanks, followed by one blank and the F-cycles.

Example 2: Extract log entries created under the account number (A option). The run deck appears as follows:

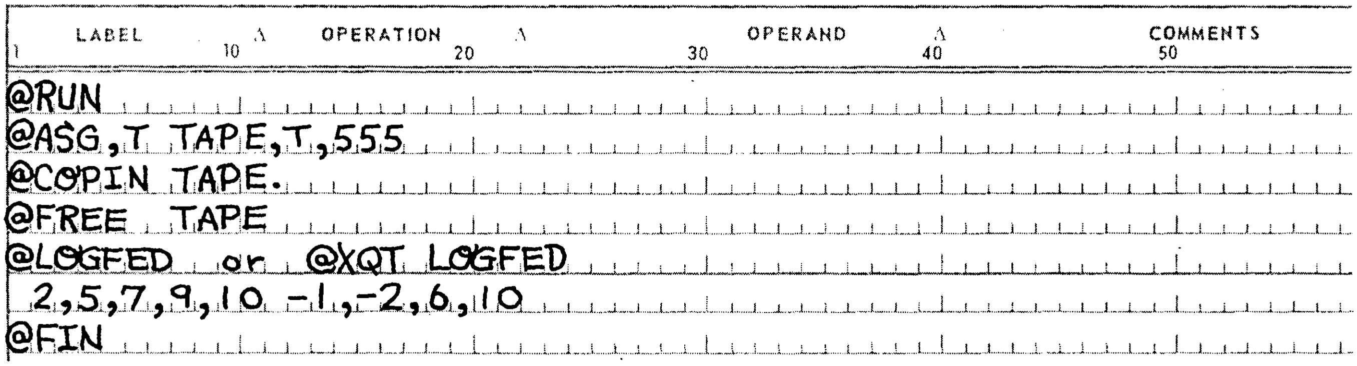


LABEL OPERATION OPERAND COMMENTS
1 10 20 30 40 50

```
@RUN
@ASG,T TAPE,T,555
@COPIN TAPE.
@FREE  TAPE
@LOGFED,A  or  @XQT LOGFED,A
399126 2,5,7,9,10 -1,-2,6,10
@FIN
```

Note the deck remains the same except the R option was replaced with an A option, and the run-id was replaced with the account number.

Example 3: Extract log entries regardless of runid or account number (no option). The run deck appears as follows:



LABEL OPERATION OPERAND COMMENTS
1 10 20 30 40 50

```
@RUN
@ASG,T TAPE,T,555
@COPIN TAPE.
@FREE  TAPE
@LOGFED  or  @XQT LOGFED
 2,5,7,9,10 -1,-2,6,10
@FIN
```

The parameter statement has changed in that as no run-id or account number is given, it begins with a blank in column 1 followed immediately by the log entry types to be extracted, followed by a blank and the desired F-cycles.

LOGFED is capable of processing F-cycles in any order they may be given on the parameter statement. However, it would be logical to arrange the F-cycles in chronological order; that is, the oldest F-cycle appearing first in the parameter statement. The examples given for the parameter statements show F- cycles given in a more or less random order.

The error messages produced by LOGFED are as follows:

- FILE CANNOT BE ASSIGNED   A0 = (facility-status-bits)

  The specified F-cycle could not be assigned.

- INCORRECT PARAMETER CARD

  The user's parameter card is mispunched. LOGFED exits through ERR$.

- END OF CYCLE

  The end of an F-cycle has been reached.

- I/O STATUS CODE = xxx – SWITCHING TO NEXT F–CYCLE

where xxx is:

1_8 – An ER to IOW$ has returned a status code of 1_8.

4_8 – An ER to IOW$ has returned a status code of 4_8.

11_8 – An ER to IOW$ has returned a status code of 11_8.

12_8 – An ER to IOW$ has returned a status code of 12_8.

13_8 – An ER to IOW$ has returned a status code of 13_8.

See Appendix C for I/O status codes.

Sample of Output from LOGFED:

An example of LOGFED output for one master log entry of type 9 is as follows:

```
Run Card (Type 9)

OPTIONS             –

PRIORITY            –   M

START TIME          –   0000

DEADLINE            –   0000

EST RUN TIME        –   600

EST PRINT OUT       –   100

EST CARDS OUT       –   100

NEW RUN-ID          –   SYS

OLD RUN-ID          –   SYS

PROJECT             –   EXEC8$

ACCOUNT NO          –   INSTALLATION

DEVICE ASSOC.       –   75 octal

TIME OF LOG ENTRY   –   07/09/71 – 12:24:06
```

24. FILE STRUCTURE AND MAINTENANCE

24.1. INTRODUCTION

This section describes the file formats and file maintenance software, both of which are normally transparent to the user. The information is provided:

- To give insight into the file structure used by the FURPUR processor, the language and system processors, and the symbiont complex;
- To enable the user to write additional software to build, insert, and retrieve from the files.

The operating system generates three major types of files:

- Program File
- Element File
- System Data File (SDF)

The format of each of these files and the manner in which they are manipulated are described in the following paragraphs.

24.2. FILE FORMATS

24.2.1. PROGRAM FILE FORMAT

A program file can be defined as a random access file consisting of a group of elements residing on FASTRAND-formatted mass storage. A program file may contain symbolic, relocatable, or absolute elements or a combination of elements. It may be either a temporary or a catalogued file. Since the elements are named, they may be manipulated on an individual basis. Thus, the elements needed to produce an executable program may be collected from one program file or from several program files.

It must be emphasized that while the program file is logically structured as shown in Figure 24-1, physically the elements that make up the file are not necessarily contiguous. Linkages are automatically generated by the executive to logically structure the file as a separate continuous entity.

The program file (see Figure 24-1) has three major sections:

- File Table Index – Provides the key to where the tables (in the file) appear relative to the beginning of the file.
- Tables of Contents – Provides the pointers to the elements and procedures in the text section, and entry points for relocatable binary elements.
- Text – The elements.

The file table index contains pointers and links to the tables which comprise the file's table of contents. The table of contents consist of an element table, procedure tables, and an entry point table. These tables are described in the following paragraphs.

The element table contains the Fieldata element and version name of each element, its type (symbolic, relocatable, or absolute), and pointers to its text within the program file. It also provides information concerning:

- the size of the element,
- the time and date the element was created,
- the sequence number of the element in the file which is used for linking entries within the element table (the sequence number specifies the order in which the element texts are entered in the file), and
- the address of the text.

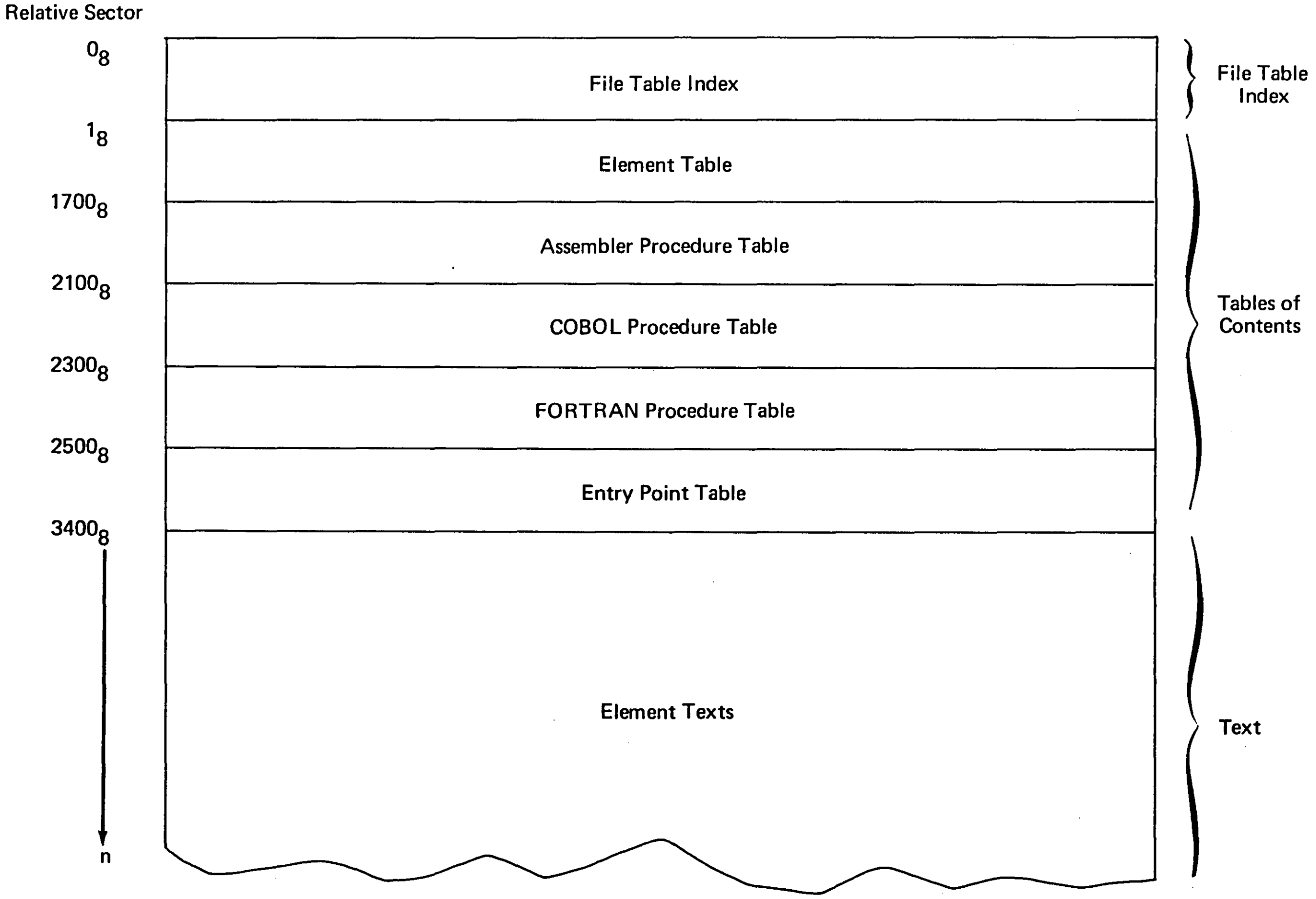


Figure 24–1. Program File Format

The assembler procedure table has an entry for each assembler procedure (entered in the file by the @PDP control statement – see 9.7). Each entry consists of:

- the procedure name,
- a link to the element in which it appears, and
- the procedure's relative location within the element.

The COBOL and FORTRAN procedure tables contain essentially the same type of entries as the assembler procedure table except that they pertain to COBOL and FORTRAN procedures.

The entry point table is the set of all entry point names and the link from each name to the relocatable element in which it occurs. The user must request the generation of this information using the @PREP control statement (see 8.2.11); it is not done automatically by the executive.

Images in symbolic elements within a program file are in SDF format (see 24.2.3). Relocatable and absolute elements have different formats.

For more detailed information concerning the table of contents and element formats, see the *UNIVAC 1108 Operating System Technical Documentation* (current version).

24.2.2. ELEMENT FILE FORMAT

An element file is produced from a program file by using a @COPOUT control statement (see 8.2.3). It is a sequential file, found only on magnetic tape, and it may consist of a series of symbolic, relocatable, and absolute elements. It may be a temporary or a catalogued file.

The elements (see Figure 24-2) are written in sequential order on the tape. Each element (see Figure 24-3) contains a 28-word element label block and the element text. The element label block is created from information contained in the program file element table item. The element label block contains the following information:

- element file identifier,
- element name, version, type, and size, and
- the time and date the element was added to the system.

The remainder of the element consists of 224-word blocks of the text of the element. This information is identical to the element text in the program file from which the element file was created. The only difference is that the element text is blocked into 224-word blocks; the last block is padded to force a 224-word block if the text does not occupy an exact multiple of 224 words.

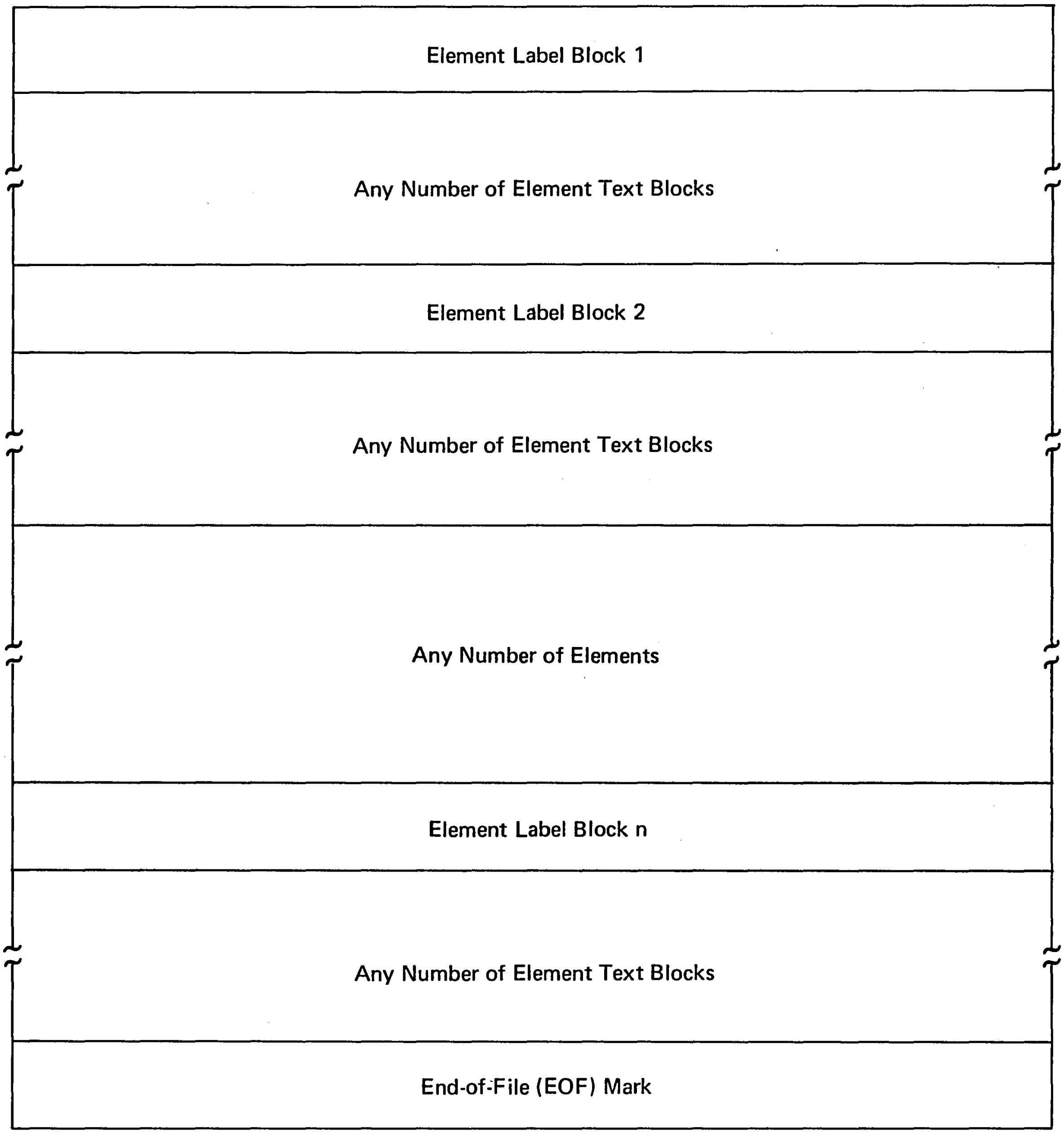


Figure 24–2. Element File Format

24.2.3. SYSTEM DATA FILE (SDF) FORMAT

SDF provides the system with a basic format for data handling between the various system components, and between the system and the user. Data in an SDF-formatted file is recorded in either Fieldata or ASCII. SDF format is also used for symbolic elements within a program file.

Data in an SDF-formatted file is recorded in variable-length images, with each image being preceded by a control word which specifies image length and type. Images are of two general types:

(1) Control Images

(2) Data Images

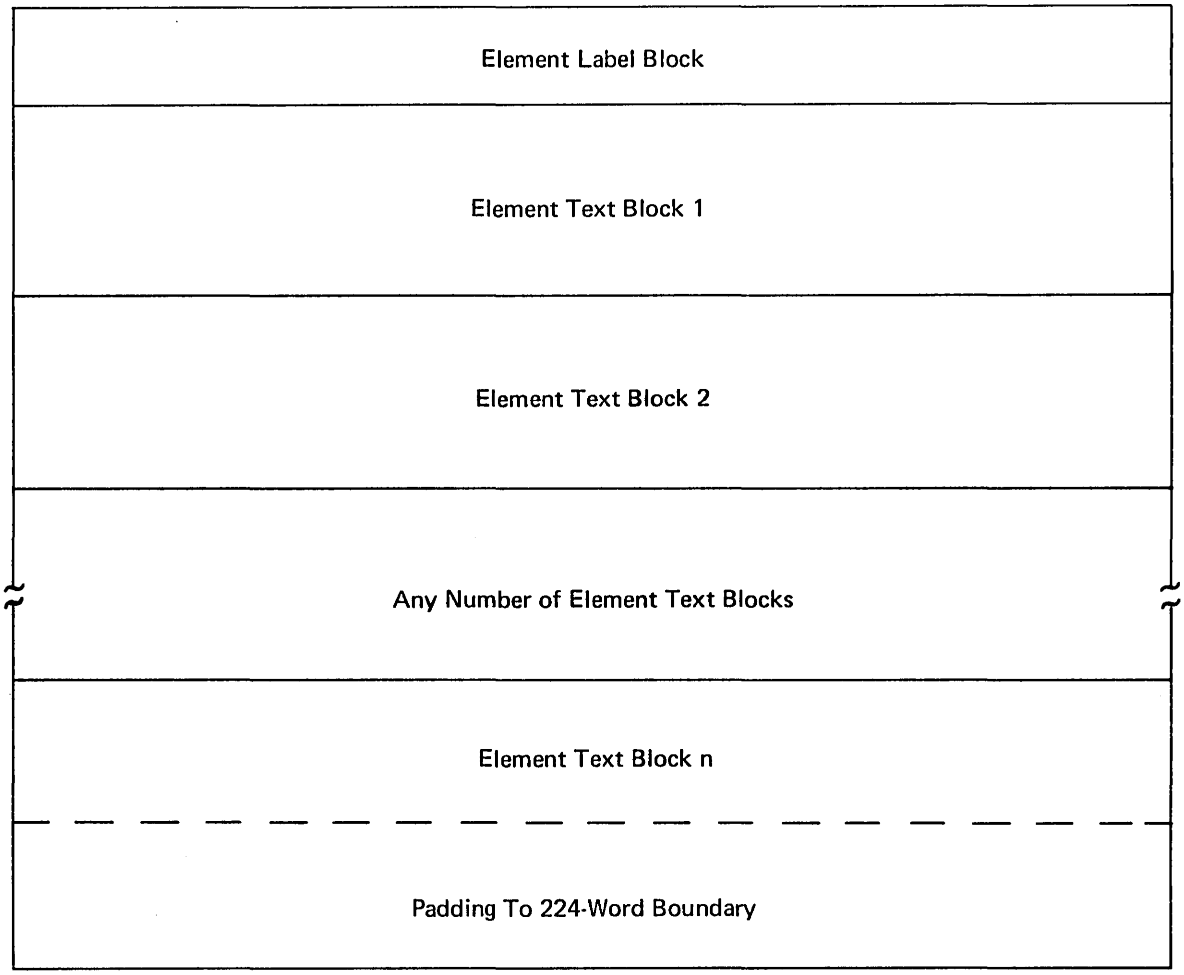


Figure 24–3. Element in Element File Format

Control images provide various file control information as is needed by the individual component processing the file. A control image is indicated by bit 2_{35} being set in the control word preceding it. The initial image of every SDF-formatted file must be a label image which is defined by a 50_8 in bits 35-30 of the control word. The control word for SDF control images has the format:

S1	S2	S3	S4	S5	S6
c	l	ft	p		ct

Where:

c A unique code indicating what information is contained in the control image. The possible values of c are:

40_8	–	Bypass this image
41_8	–	Unique READ$ file label image
42_8	–	ASCII/Fieldata switch
43_8	–	Special FORTRAN backup
50_8	–	Label image
51_8	–	Continuation of previous image
54_8	–	End-of-reel
56_8	–	Start of accounting information in print file
57_8	–	Output symbiont position marker
60_8	–	Print control image
70_8	–	Punch control image
76_8	–	Demand breakpoint EOF
77_8	–	End-of-file

l The length in words of the following image.

ft Used only in label blocks and it is the file type as follows:

C	–	Symbiont card file
F	–	FORTRAN library data file
I	–	Symbiont input file (created by @FILE control statement)
P	–	Symbiont print file

p Used only in the label block of symbiont files and is the part number of the file (that is, a count of the breakpoints performed on the file).

ct The code type of the following images where:

0	–	Fieldata
1	–	ASCII

Any image is an SDF-formatted file whose control word does not have bit 2_{35} set is a data image. The control word for an SDF data image has the format:

0	l	n	ct
35	34 24	23 6	5 0

where:

l The length in words of the following image.

n Used by each component to contain special information (that is, symbolic elements use this field for cycle numbers and the symbionts use bits 23-12 for spacing increments).

ct The code type of the following image where:

0	–	Fieldata
1	–	ASCII

SDF-formatted files residing on magnetic tape are recorded in 224-word blocks. Images are allowed to span two consecutive blocks, however, a 51_8 control word is inserted preceding the second portion of the image to indicate that the image is continued. The end-of-file (EOF) sentinel control image terminates file processing. A single tape mark is used to separate files residing on magnetic tape, and two consecutive tape marks specify end-of-recording. Mass storage files use a software-defined EOF mark for file operations at the block level. This is defined as a block which has as its first word the Fieldata sentinel '$@EOF'. The block buffering package interprets this sentinel as it would a tape mark.

SDF maintains integrated FASTRAND-formatted mass storage and magnetic tape compatibility by:

(1) Adhering to block lengths of FASTRAND sector multiples.

(2) Imbedding file labels in the data as the initial image of the file.

(3) Writing the file as a continuous set of data without block control words.

(4) Incorporating the block buffering package's facility of sequential file processing.

24.3. FILE MAINTENANCE

Within the operating system are contained various library routines, executive service functions, and processors that may be used to create and manipulate files. The file utility processor, FURPUR, (see Section 8) may be used to process, in various ways, files of the previously discussed formats. In fact, element files are created and processed only by the FURPUR processor.

SDF-formatted files and elements are produced and processed by the FURPUR, DATA, ED, ELT, and other processors. In addition, the symbiont and @ADD control statement (see 3.9.1) as well as the various symbiont interface executive functions (see Section 5) are used to create and process SDF-formatted files.

Program files are created and processed by the language processors, FURPUR, ELT, LIST, CULL and other processors. In addition, there are several executive service functions (see 24.3.1) and relocatable library routines (see 24.3.2) available for processing program files.

Paragraphs 24.3.1 and 24.3.2 describe the mechanism for updating a program file by a user program. Both features were designed primarily for use by the Univac language and system processors. The program file maintenance executive requests (see 24.3.1) provide a limited capability in that only selected functions are available. The program file basic service package (see 24.3.2) is a library routine that can be included with a user program to provide more capability with less overhead if many operations are to be done.

The executive requests are also used by the executive in its normal operations, such as finding an absolute program to execute.

24.3.1. PROGRAM FILE MAINTENANCE EXECUTIVE REQUESTS

The executive requests described in the following paragraphs are used to maintain the table of contents entries for a program file. As a group, the requests are called the program file package (PFP). The formats of the program file table of contents entries can be found in the *UNIVAC 1108 Operating System Technical Documentation* (current version).

For each of the requests described, upon return from the requests, register A2 contains the status of the operation performed (see 24.3.1.6).

24.3.1.1. UPDATING THE ELEMENT TABLE (PFI$)

Purpose:

Inserts an entry in the program file's element table.

Format:

```
L    A0,(pktaddr)
ER   PFI$
```

Pktaddr is the address of a packet whose format is:

	T1	S3	H1
Word 0	filename		
1			
2	element-name		
3			
4	*version-link-sequence-nbr*		*pointer-link-sequence-nbr*
5	flag-bits	element-type	*type-link-sequence-nbr*
6	element-version-name		
7			
8			
9			
10	text-address		
11	date-and-time-of-creation		

Word 5

flag-bits:

2^{35} — Marked for deletion

2^{28} — ASCII symbolic

2^{26} — Third-word sensitive

2^{25} — Quarter-word sensitive

2^{24} — Marked in error

Element-types:

1_8	—	Symbolic
2_8	—	Assembler procedure
3_8	—	COBOL procedure
4_8	—	FORTRAN procedure
5_8	—	Relocatable
6_8	—	Absolute

Words 8–9

The contents of words 8 and 9 are dependent upon the type of element as follows.

Symbolic and procedure elements:

		35 31	30 24	23 18	17 12	11 0
Word	8	cycle-limit		latest-cycle		cycle-count
	9	processor-code	0		sector-length-of-text	

processor-code:

0_8	—	Unmarked
1_8	—	@ELT
2_8	—	@ASM
3_8	—	@COB
4_8	—	@FOR
5_8	—	@ALG
6_8	—	@MAP
7_8	—	@DOC
10_8	—	@SECURE

Relocatable elements:

		H1	H2
Word	8	sector-location-of-preamble-within-file	
	9	sector-length-of-preamble	sector-length-of-text

Absolute elements:

	H1	H2
Word 8	program-I-bank-word-length	program-D-bank-word-length
9	first-D-bank-addr-in-program	sector-length-of-text

Description:

The element version can be present, zero, or blank. When the version is zero, blanks are substituted.

When an absolute element is being inserted, its sequence number is recorded in the file table index.

For the relocatable elements, the file table index pointers to the entry point table are cleared and a new entry point table may have to be created.

If the date-and-time-of-creation field is zero, the PFI$ request inserts the current date and time. When this field is nonzero, the contents are used for the date and time.

The link sequence numbers are supplied by the PFI$ request.

24.3.1.2. TABLE OF CONTENTS SEARCH (PFS$)

Purpose:

Searches program file's table of contents for a given item.

Format:

```
    L,U  A0,pktaddr
    ER   PFS$
```

Words 8 and 9 of the packet are supplied by the executive.

Pktaddr is the address of a packet whose format is identical to that of the PFI$ request (see 24.3.1.1).

Description:

When the delete flag (bit 2^{35} of word 5) is set, a find can be made on an element marked for deletion.

When the element name is left blank and the element desired is an absolute element, the PFS$ request supplies the last absolute element added to the file.

If the version name is zero, a find is made on element name only. When a version name other than zero is specified, it is used along with the element name in the search. When a version name is blank, a find is justified on the element name and blanks, for the version.

When the element is found, the packet is filled with information from the element table entry. This information is used to access the element text.

24.3.1.3. MARK ELEMENT FOR DELETION (PFD$)

Purpose:

Sets delete flag in element table item for requested element.

Format:

```
L,U   A0,pktaddr
ER    PFD$
```

Pktaddr is the address of a packet whose format is:

Word		S3	
0	filename		
1			
2	element-name		
3			
4			
5		element-type	
6	element-version		
7			

Word 5

element-type	Same as PFI$ (see 24.3.1.1)

Description:

When the element being deleted is the most recently added absolute element in this file, the file table index entry containing the element sequence number of the most recently added absolute element is cleared to zero.

24.3.1.4. UPDATING NEXT WRITE LOCATION (PFUWL$)

Purpose:

Updates the next write location in a program file. The task of the PFUWLS request may also be performed by using PFI$ (see 24.3.1.1). This is accomplished by complementing register A0 on the call to PFI$.

Format:

```
L,U   A0,pktaddr
L     A1,(new-address-in-program-file)
ER    PFUWL$
```

Pktaddr is the address of a packet whose format is:

Word 0	filename
1	

The next write location is the next available sector at which the text portion of the element can be written without destroying other text words.

24.3.1.5. RETRIEVING NEXT WRITE LOCATION ADDRESS (PFWL$)

Purpose:

Obtains the next write location in the program file.

Format:

```
L,U   A0,pktaddr
ER    PFWL$
```

Pktaddr is the address of a packet whose format is:

Word 0	filename
1	

The next write location is stored in register A1 upon normal return. The next write location is as defined for PFUWL$ (see 24.3.1.4).

24.3.1.6. PROGRAM FILE PACKAGE STATUS CONDITIONS

The status conditions are stored in register A2 on the return from the program file package ER's. The possible values are:

0_8	–	Normal status (operation completed)
1_8	–	No find
2_8	–	I/O error
3_8	–	Program file not defined
5_8	–	Program file overflow

24.3.2. PROGRAM FILE BASIC SERVICE PACKAGE

The basic service package (BSP) is an interface routine between the user program and a program file's table of contents. The user may, through selective calls to the BSP, perform the following functions:

(1) Read the file table index into main storage buffer.

(2) Read a selected program file table into a main storage buffer.

(3) Search a selected program file table for a specific entry.

(4) Delete a specific entry in a program file table.

(5) Locate a program file table entry from its sequence number in the table.

(6) Add an entry to a program file table.

(7) Write the last entry referenced.

(8) Write a program file table.

(9) Write the file table index.

(10) Unlock the file table index.

Any file assigned and not previously written in may be prepared as a program file by requesting (1) above. A table is created upon the first call to read it if not previously created.

Several features of this package and rules for use are listed:

(1) The user attempting to create a program file in a file area previously written in receives an error return.

(2) Each table (except for the element table) on drum begins at the system defined sector or at a greater sector if a previous table now occupies the defined sector.

(3) The table of contents is a fixed length – 1792 sectors.

(4) Tables must be written back from the main storage area assigned, if the area is to be used for others. A diagnostic is given if a table cannot accept another item.

(5) The user must provide a 34-word area (file control table) for the file table index (FTI) and main storage buffers for each table read.

(6) No main storage buffer may be less than 196 words in size, or an error diagnostic results.

(7) The FTI in main storage is used by BSP to contain necessary information about the file.

(8) Each table that has been altered in main storage requires a final call on the write program file subroutine in order to insure all information changed reflected on mass storage.

See *UNIVAC 1108 Operating System Technical Documentation* (current version) for further information.

25. INTERNAL EXECUTIVE DESIGN

25.1. INTRODUCTION

This section is not required reading prior to intelligent use of the system. The intention is to provide the interested reader with additional insight into the key concepts and algorithms incorporated in the internal design of the executive. The information presented is in no way intended as a complete description of internal logic in that many important aspects of the design are not mentioned. It is thought, however, that this section along with the internal design specifications either stated or implied in other sections (especially Sections 1 and 2, which discuss general capabilities and concepts) will provide the user with an overview of the total internal system design from the user programmer point of view. Design and capabilities relative to system generation and operator interface are not discussed, but they can be found in other Univac publications.

Univac reserves the right to make changes in the internal design without notice.

25.2. BASIC DESIGN PHILOSOPHY

To achieve the broad spectrum of functional and operational capabilities defined for the executive in a reasonable amount of space and time, certain fundamental design criteria have been adopted:

- Wherever feasible and appropriate, individual components operate reentrantly. Where true reentrancy is not feasible, the individual component rather than its requestor, is in general responsible for serializing its operation.

- Components and data types which are permanently resident in main storage are limited to ones for which nonresidency is either impossible or would impose unacceptable overhead.

- Central processor units (CPU) are in general treated equally. Components must be prepared to execute on any CPU; exceptions are limited to certain maintenance functions, and to input/output (I/O) situations in which a particular CPU does not have a direct electrical path to a particular peripheral device. This approach allows maximum CPU utilization in multiprocessor systems.

- To assure timely response to real time related events, interrupt lockouts, and software interlocks on interrupt-related data are kept to a minimum. Wherever feasible and appropriate, real time requests are given top service priority.

- To preserve generality, most executive components operate as ordinary activities, called exec workers, and are managed in a fashion very similar to the management of user task activities. Exceptions are limited to functions for which switching is either impossible (for example, the dispatcher and interrupt queueing) or requires unacceptable overhead.

- To meet system integrity and program protection requirements, individual user service requests are validated to whatever extent is necessary to eliminate undesired interaction with the system or other users.

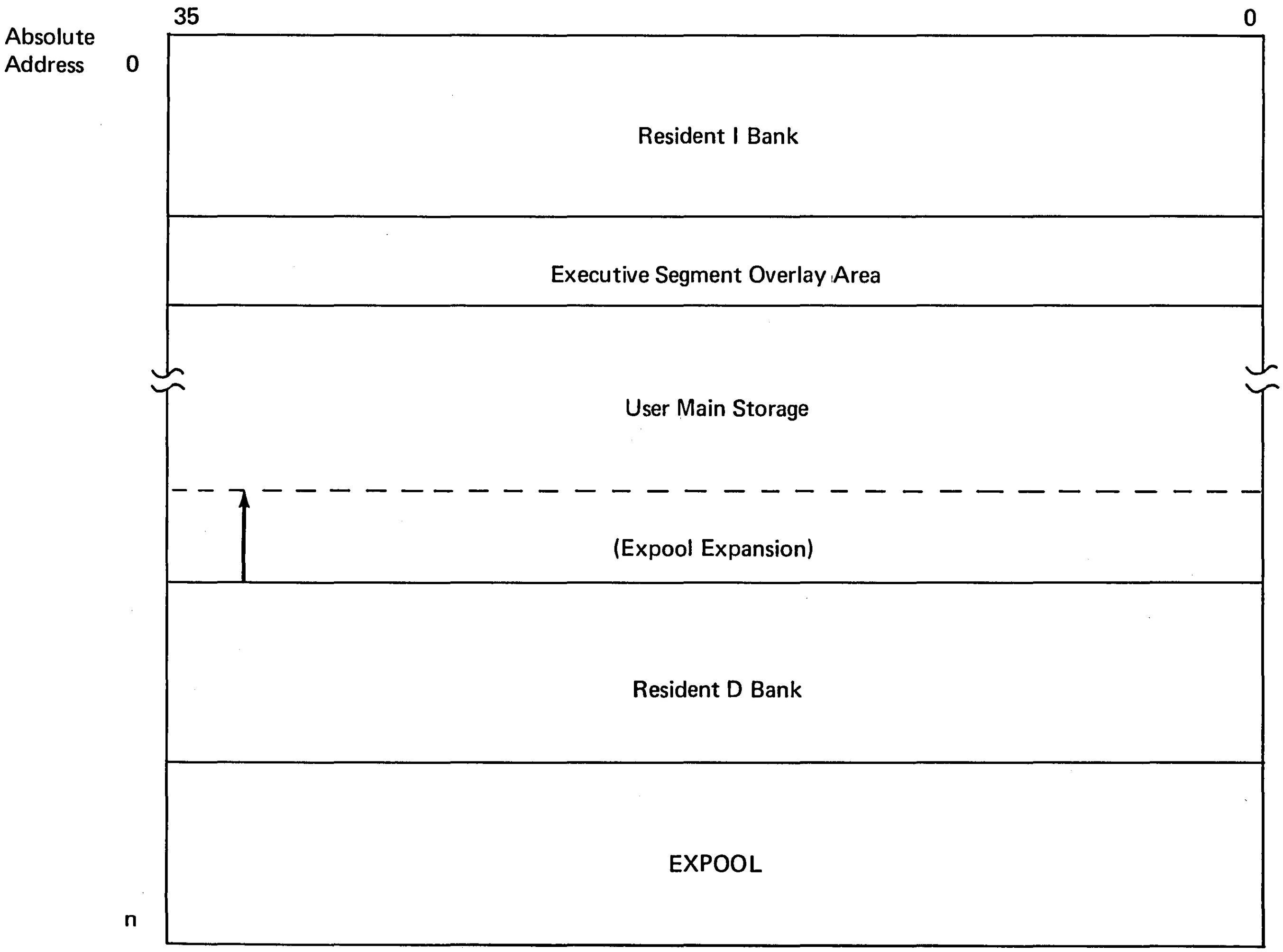
- Code required to support optional hardware components and software features is written such that it is not generated if the associated component or feature is not configured.

- To avoid wasting high cost resources like CPUs and main storage, data transfers involving low speed peripherals (for example, card readers, teletypes, and so forth) are normally buffered. Generally, large-volume data is buffered to mass storage and low-volume data is buffered in main storage until sufficient data has accumulated to warrant loading a task to process it. This approach means that user tasks can be executed at rates commensurate with the overall power of the system, and need not spend long periods essentially idle in main storage waiting (usually many milliseconds) for short data transfers to complete. The symbiont complex constitutes the principal application of this philosophy.

25.3. EXECUTIVE MAIN STORAGE USAGE

25.3.1. GENERAL LAYOUT AND DISCUSSION

Main storage has the following general physical layout:



The individual areas are as follows:

(1) The resident I bank area includes the interrupt locations and permanently resident executive instructions. In general, no data is stored in this area. Also, it does not include any routines which are referenced directly by executive segments (nonresident routines) since the latter are also handled as I banks.

(2) The executive segment overlay area is devoted exclusively to holding executive segments (transient or nonresident routines). This area is at least as large as the largest segment. Design rules for executive segments stipulate that a segment must not require any other segment to be loaded at the same time. Thus, at minimum, a serial operation is guaranteed.

(3) The user main storage area is devoted primarily to user tasks and their associated program control tables (PCTs). Executive segments are also executed in this area when space permits; however, user tasks are always given priority over executive segments. In a few cases, the executive may acquire independent data buffers from this area, but these are normally acquired from EXPOOL (see 5).

When the system is heavily loaded, the high address end of the user main storage area may be used to temporarily expand EXPOOL space.

(4) The resident D bank area includes fixed (as opposed to pooled) data storage and those permanently resident routines which are referenced directly by executive segments.

(5) EXPOOL is the executive data space pool. The EXPOOL mechanism is fundamental to executive design, since the ability to rapidly and conveniently acquire data space is essential for efficient reentrant design. EXPOOL allocation is based on a powers-of-2 structure wherein buffer sizes are 3, 7, 15, 31, 63, 127, 255, or 511 words (one word is always used for buffer control). While this sizing may cause some waste in in-use buffers, the structure allows extremely rapid buffer collection, which is very difficult in pooling structures which provide a continuum of sizes. Briefly, whenever a buffer of a particular requested size is not available, a check is made for an available buffer of the next larger size; if there is one, it is split into two equal halves; if not, the process is recursively repeated for still larger buffer sizes until a buffer can be obtained. When a buffer is released back to EXPOOL, it is recombined with adjacent available buffers to make the largest possible buffer.

25.3.2. PCT USAGE

Most data controlling the execution of a particular user run or task is maintained in the PCT. Such data is maintained in EXPOOL only if the data may be required while the task (including its PCT) is swapped out (for example, swapping control information) or if it may apply to another run (for example, information pertaining to catalogued files). This allows many runs (especially demand runs) to be controlled concurrently without an exorbitant load on EXPOOL. A prime example of this approach is activity control; the basic information concerning the status of an activity is kept in EXPOOL, but the activity environment, including control registers, is saved in the PCT.

PCTs have two logical parts: a fixed area containing data always needed to control a run or its current task, and a variable area. The space in the variable portion is pooled by a mechanism analogous to EXPOOL. Also, the PCT may dynamically expand from a normal minimum of two main storage blocks (1024 words) for batch runs and one main storage block for demand runs up to a normal maximum of nine main storage blocks.

25.3.3. DEFINITION AND RESIDENCY OF COMPONENTS

Table 25-1 lists the executive components that permanently reside in main storage, and 25-2 lists the executive components that are normally transient in main storage.

Executive Component
Dispatcher and Activity Control
ER Interrupt Handler
Fault Interrupt Handlers
Test and Set Interrupt Handler
Executive Segment Controller
Clock Interrupt Handlers and Clock Management
Basic Internal Service Routines (EXPOOL, Access Word Check, and so forth)
Basic Dynamic Allocator
Basic I/O Control and Interrupt Queueing
Basic Communications Control
ISI Device Handlers
Console Handler
Symbiont/User Interface Routines (READ$, PRINT$, and so forth)
Basic Symbiont Interface and Control
Basic Symbiont Device Control
Logging Control

Table 25-1. Executive Components that Reside Permanently in Main Storage

Executive Component
Coarse Scheduler (Including Run Initiation)
Control Statement Interpreter
Run Termination
Control Statement Function (CSF$)
@START Control Statement Handler
Logging & Accounting
Checkpoint/Restart
User Task and Segment Loader
Dynamic Allocator Periodic Algorithm Adjustment
Program Expansion/Contraction (MCORE$/LCORE$)
Reentrant Processor Registration (RLIST$)
Activity Error Termination
Task Termination
PCT Expansion
Activity Naming (NAME$)
Unsolicited Keyin Distributor and Handlers
Date Control and Keyin Handler
Console Patch Routine
Symbiont Probe
Symbiont Device Initialization/Termination
@ADD Statement Control
Symbiont Forms Control
Symbiont File Manipulation (Alternates, Breakpoint, SYM, and so forth)
Symbiont Output File Queueing and Termination
Symbiont Operations Control (Repositioning, Error Handling, and so forth)
Facilities Inventory & Assignment
Facilities — I/O Interface
Mass Storage Space Allocation
Master Directory Control
Catalogued File Rollout/Rollback
Catalogued File Recovery
Tape Reel Control (TSWAP$ and TINTL$)
Tape/Disc Labeling
Disc Prep
Communications Initialization and Termination
System Initialization
Program File Package
Panic Dump Editor

Table 25-2. Nonresident (Transient) Components of the Executive System

25.4. MULTIPROCESSING

The scheduling and CPU switching techniques used in the executive system are designed to provide multiprocessing capabilities. The extension to multiprocessing leads naturally from the multiprogramming aspects of the system where many independent activities in user programs are available for execution at any instant. Within the executive itself a similar situation exists where at any moment more than one independent exec worker requires CPU time.

Consider an executive activity for a program that has an outstanding request for an I/O operation. When the I/O completion interrupt occurs, the executive interrupts the activity at a point unknown to it. The activity environment is saved, the interrupt processed, and control returned to the activity. The activity is never aware of the event, and would not be the wiser had the interrupt taken the executive to another CPU where the task was performed to complete the I/O operation.

In a multiprocessor configuration, one executive in a shared main storage controls all processing. The multiprocessing extensions provide the ability to isolate each of the available CPUs. Each, in turn, acting as executive CPU inspects the list of current activities and selects one to be executed. One CPU may interlock the others while referencing critical areas of common data.

The many runs being input to the system provide a number of tasks to be multiprocessed. Within any run, the individual tasks are executed in a serial manner as directed by the user. Among the many runs, the executive typically uses the processors of the system to work on tasks of more than one run.

The executive provides the ability, via forking to generate multiple activities, for a user to split a program into an arbitrary number of independent execution paths. Each activity available for processing on any of the CPUs of the system. System design provides the ability to use all available CPUs on the execution of a particular program. The executive is of course free to use the available CPUs for program-related but independent operations necessary within the executive.

The areas of executive coding which reference common data and others with specialized coding methods must be protected from simultaneous execution, but many areas will be open and multiprocessed as necessary.

25.5. SCHEDULING

25.5.1. GENERAL

The general scheduling technique used by the executive removes any difficulty in the advancement of an installation into the use of multiprogramming. The technique is easily understood, and any installation can readily modify or extend its capabilities if necessary.

The scheduling components of the supervisor are responsible for the control of facilities as well as the actual scheduling of runs and tasks. This includes both the assignment and release of facilities. There are five components within the system for handling the scheduling of runs and the tasks within runs; these are:

- Facilities Inventory
- Control Statement Interpreter (CSI)
- Coarse Scheduler
- Dynamic Allocator
- Dispatcher

Each routine is discussed individually in the following paragraphs.

25.5.2. FACILITIES INVENTORY AND SELECTION

The facilities at the disposal of the executive system include the I/O channels and all peripheral equipment attached to these channels, including available communications line terminals. Available facilities and their disposition are indicated to the system at system generation time; thereafter, the executive assigns these facilities as needed and as available, to fulfill the facility requirements of all runs entering the system. The executive maintains and continually updates inventory tables that reflect what facilities are available for assignment, and which runs are using the currently 'in use' facilities.

As demonstrated in the following discussion, facilities inventory is a basic constituent of the scheduling section of the supervisor. Only by establishing an efficient and convenient means of facilities control is the multiprogramming environment practical. The routines controlling the facilities available to the system are designed to optimize utilization of those facilities while requiring a minimum of user-generated statements concerning operational requirements.

Devices such as magnetic tapes are normally assigned before run execution time, because they cannot be shared by two or more runs; and they normally require operator set ups. Such devices are always released automatically by the termination of the run; however, they may also be released during the course of the run by the user.

Mass storage space is dynamically assignable and releasable by the user. The user is encouraged to do so whenever possible, since no relocation of information is necessary in allocating mass storage facilities (as may be the case for main storage).

Word-addressable devices (FH-432, FH-1782, and so forth) are allocated internally in FASTRAND granules (track, position). This allows the same space to be assigned as either word-addressable drum format or as FASTRAND-formatted mass storage, depending on user requirements at the particular time. When word-addressable format is requested, an attempt is made to allocate the largest possible continuous space in order to give users maximum benefit of the word-addressable characteristics of the device. A portion of drum (or disc) is set aside at system generation time for the residence of the system and the processors. Normally, all user files which utilize drum during the course of a run are purged at the completion of the run, and the space used for such files is returned to the pool of available facilities. If so specified, however, a file may be retained for future reference (catalogued).

Mass storage space is allocated in granules, with one granule equal to a track or a position (64 tracks). Given the equipment type from the @ASG control statement, the unit and area for the allocation of a granule of a particular file are selected according to the following hierarchy (only a simplified view of the algorithm is presented):

(1) Allocate in a particular place on the unit which will permit a contiguous allocation to a previous allocation for this file.

(2) Allocate anywhere on the same unit as per the previous allocation.

(3) If the file is catalogued, allocate on the unit containing the master file directory information for this file.

(4) Allocate on the unit (of the desired equipment type) with the most availability.

(5) Change the equipment type to next most desirable type and repeat step (4).

By use of the @ASG control statement, a user specifies the number of granules to be initially reserved for the file.

Each dynamic request for additional space then results in the assignment of an additional granule. When using the system file control routines, the user will not have to request additional mass storage space because this procedure is taken care of automatically by the system.

In addition to maintaining cognizance of system information concerning device errors, and so forth, the facilities inventory routine is able to accept direction from an operator concerning device reliability, and so forth. The operator may request that devices or channels be removed from the pool of available facilities.

When the coarse scheduler selects a run to be opened, an initial facility synopsis is performed to determine if the run can be opened. If it is found that a requested facility is not available, information pertinent to the run and the reason for the hold are gathered in a queue. The queue has three sections based on whether the hold is for initial facilities, facilities between executes or facility requests from a demand terminal.

The hierarchy of the sections within the queue are as follows:

(1) Facility requests from a demand terminal.

(2) Facility requests between executes.

(3) Initial facility requests.

A held run remains in the queue until the hold mechanism is stimulated by the release of a facility. At this point, a search of the queue is performed based on the aforementioned hierarchy to determine if there is a held run which is waiting for the facility just released. If such a run is found, it is removed from the queue and a facility synopsis is again performed.

The operator is informed via a console display when the executive has tried to open a held run n times; where n is specified at systems generation time. The operator may then remove the held run via keyin. From a demand terminal, the break key followed by an X keyin removes the demand run from the held status.

25.5.3. CONTROL STATEMENT INTERPRETER (CSI)

Control statement interpretation is accomplished by CSI. It is used by the executive for the purpose of interpreting the input run stream, and is exercised by both the input symbionts and the coarse scheduler. It is also used by the CSF function to interpret control statement images passed to that function internally via CSF$ requests. For each control statement image passed for interpretation, CSI performs the following:

(1) Format checks the statement.

(2) Requests continuation statements (continuation character is treated as an illegal character if request was to CSF).

(3) Changes the external format of the control statement to an internally formatted table acceptable to the system.

The CSI is the single component of the supervisor that dictates the syntax of the executive control language. If it should become necessary to input a run stream different from that specified by the executive control language, the CSI could be modified to accept this input as long as the interpretation presented functions known to the coarse scheduler and in the proper order and grouping. The coarse scheduler is the level at which the capabilities and functions of the system are defined. The coarse scheduler scans the input stream via READ$ requests in search of the next logical task or parameter on which it must act. If the next statement is a control statement, READ$ activates CSI and the internal table it builds is passed back to the coarse scheduler. The interface between CSI and the system is fixed, but input to CSI is fixed only in the sense that the executive control language is defined to have a particular syntax.

25.5.4. COARSE SCHEDULER

The coarse scheduler has two basic functions. They are:

(1) The introduction of new runs into a backlog queue and the updating of runs already in the backlog queue.

(2) The processing of control statements in an open run and the subsequent initiation of an executive operation or user task which is specified by a control statement.

The first function is initiated when a new run is introduced to the system. The run may be introduced via a @START control statement, a @RSTRT control statement, or it may come through a symbiont input device. In all cases, the coarse scheduler performs legality checks on the @RUN control statement, and if the run is a batch run, an entry representing the run is built and inserted into the backlog queue which is more commonly known as schedule queue (SCHQ). Demand runs are not inserted into SCHQ after the legality check on the @RUN control statement. Instead, they are immediately opened.

The coarse scheduler also updates the scheduling parameters of runs in SCHQ. The update must be performed when one of the following conditions exists:

(1) An S option hold, a remote batch symbiont hold, or a console imposed hold is to be removed from a backlogged run.

(2) Scheduling parameters, for a particular run or global parameters, may be dynamically changed via a console keyin.

(3) A backlogged run's priority may be changed because a deadline time is imminent or the hold on a run which is imposed by a start-time must be removed because the start-time is imminent.

The first function also includes the selection of batch runs to be opened from SCHQ. The following rules and constraints govern this selection process.

(1) No runs except special system initiated runs are opened if:

(a) the maximum number of batch runs (defined at system generation) are already opened, or

(b) system overload is approaching (for example, EXPOOL or mass storage near saturation).

(2) If the above test is passed and a global scheduling hold is set, only 'removed' runs and 'allowed' runs are considered for opening. An allowed run has priority over a removed run, and the allowed run with the highest priority is chosen first.

(3) If a global hold is not set and test 1 is passed, a run is selected for opening on the basis of its current priority. Thus, a run is selected for opening only if its priority is the highest priority in SCHQ. If two or more runs contain the highest priority in SCHQ, the run which has been a candidate for opening for the longest period of time is selected. A run becomes a candidate for opening at the instant that it is free of all of the following hold conditions:

(a) a hold imposed by a start-time;

(b) an S option hold; and,

(c) a remote-batch symbiont hold.

If a run contains a deadline time as one of its scheduling parameters and the time when that run must be opened is approaching, that run's priority is adjusted so as to put it above the priorities of nondeadline runs.

After a run has been selected for opening, the second function, the processing of control statements, is started. The coarse scheduler reads the first set of facility control statements, links them together and sends them to facility inventory for processing. The reading and linking of facility type control statements by the coarse scheduler is called a facility synopsis. All control statements except the conditional control statements and the @MSG, @LOG, @HDG, and @EOF control statements terminate the facility synopsis mode. These control statements are considered transparent during the facility synopsis mode.

The processing of the initial facilities block by facilities inventory can have one of three results:

(1) Rejected due to insufficient facilities, in which case the run is not opened, but rather placed in a facilities hold (see 25.5.2).

(2) Rejected due to an error in the control statement, in which case the run is fully opened and all statements up to and including the statement causing rejection are reprocessed and printed along with any warning or error messages (includes transparent statements). The run is then terminated in error.

(3) Accepted.

If the facilities block is accepted, the run is fully opened and all statements in the block are processed as in (2), but run processing continues with the control statement that terminated facility synopsis mode. This statement is read and the operation it specifies is initiated by the coarse scheduler. This may be an executive operation in which an exec worker is initiated to process it or it may be a user task in which a main storage request packet is built and sent to the dynamic allocator thus directing it to load a program. When the exec worker has finished, or the user program has terminated, the coarse scheduler is again called to perform the following:

(1) If a user program has just terminated, the abort and error indicators for the run are tested to determine if:

(a) the run should be aborted,

(b) the run should be error terminated in which case only @PMD control statements are honored, or

(c) the processing of control statements can continue in normal fashion.

(2) The next control statement is read and the process is repeated until a @FIN control statement is read in which case facilities inventory is called to begin the run termination procedure.

Note that facilities control statements embedded in the run stream are blocked and handled in facilities synopsis mode in a fashion similar to that for the initial facilities block, the differences being that the run is already open and any hold is a between-tasks facility hold rather than an initial facilities hold.

25.5.5. DYNAMIC ALLOCATOR

25.5.5.1. GENERAL OVERVIEW

The function of the dynamic allocator is the dynamic allocation of main storage space to the current mix of tasks (programs), where 'current mix' is determined by the particular group of runs presently being processed. Assuming that more than enough requests exist for the resources (main storage space and CPU time) of the central computer, it is the job of the dynamic allocator to make an equitable allocation of these resources in order to best serve the varied interests of all users. The allocation is based on the type of tasks (real time, demand, and batch), as well as on the priorities and response times within a particular type. The basic principle under which the dynamic allocator operates is that the primary concern of any computing installation is the completion of batch runs at the required deadline (within the limitations of the operating environment) while at the same time attempting to maintain the required response times for demand users. Within this dynamic operating environment, the dividing line between demand and batch programs is subject to constant change as emphasis is placed upon allocating time to batch runs approaching the required completion time.

In order to achieve the desired goals in system throughput and provide complete time-sharing capabilities, each of the program types has characteristics which differentiate it from the others for main storage allocation purposes:

□ Batch Tasks – A batch task is the execution of a system processor or an object program in a nonconversational mode. Batch programs are ordinarily not removed from main storage to provide space for other batch programs while progressing toward completion. In some instances, a batch program is swapped because it has reached an impass such as a self-imposed wait condition or a system or operator imposed facility wait condition.

□ Deadline Tasks – A deadline task is a program which has a required completion time. Deadline tasks cause the swap of nondeadline tasks if necessary to achieve the run completion time requirement.

□ Demand Tasks – Demand tasks are contained within a run entered from a demand remote terminal and generally thought of as being conversational in nature. Demand programs preempt batch as well as other demand programs for main storage allocation (within user-defined constraints). Frequent but relatively short periods of main storage residency are generally required to provide the desired terminal response time.

□ Real Time Tasks – A real time program is a batch or demand program which is given preferential treatment because it is responding to real time hardware. These programs are never subject to swap or relocation by the dynamic allocator and as such are positioned in main storage so as to have minimum impact on system throughput. The presence of a real time program in main storage effectively reduces the amount of memory that can be allocated between the other program types.

25.5.5.2. DYNAMIC MAIN STORAGE ALLOCATION

The dynamic allocator (DA) is a service routine which reacts in response to requests from programs and other executive routines. The DA consists of three independent functions. These are:

(1) The DA proper, which selects the highest priority request from a core queue and determines the most economical method of satisfying the request.

(2) Core contents control (CCC), which carries out the requests in the manner decided upon by DA. This is the function which initiates the I/O requests for program swapping and loading.

(3) DA periodic adjustment (DAPA), which reviews program activity periodically and makes adjustments in the priorities of entries on the core queue to balance sharing between program types.

Each entry on the core queue is a service request for a program and has associated with it a request type and a priority. The priority is a function of program type, @RUN control statement priority, program characteristics, and request type. Entries are placed on the core queue in response to program requests and executive requests. The request types are:

(1) Swap Out – This is requested in order to position real time programs, to expand the program, or to service a higher priority program. A program is swapped for a higher priority program only if it is in a long wait state (for example, wait for tape mount, TWAIT$ of more than one-half second, or wait for terminal response) or if it has used up its first core quantum and the other program is not batch.

(2) Initial Load – This is the request to load a task (absolute element) and is the result of an @XQT or processor control statement.

(3) Reload – When an active program is swapped due to higher priority requirements, the request to reload that swapped out program is made at the time of the swap out. For a waiting program, the reload request is made when the wait condition is removed.

(4) PCT Expansion – The PCT must be expanded as the result of some control statement or program request.

(5) Program Expansion – The program has done an MCORE$ request.

(6) REP Load – The program is linking to a reentrant processor REP which is not already in main storage.

(7) REP Remove – A REP is removed to satisfy a higher priority request or to position it for real time execution.

Every program is assigned an area on mass storage for swapping. This area resides within a system file, SWAP$FILE, and is reallocated each time the program changes size. The entire program is moved to or from swap file with one I/O request. Reentrant processors are never swapped as it is assumed that they do not store into themselves. When a core area used by a REP is required, the space is used and the next time the REP is requested, it is loaded from the program file. When a program is reloaded after being swapped, the PSRs (processor state registers) and SLRs (storage limit registers) of all activities are

When main storage is allocated for a program, the DA attempts to position the I bank and D bank in different main storage modules in order to avoid main storage reference conflicts. Each program is also loaded as closely as possible to the extremes of available main storage in order to reduce fragmentation of the available space. Real time programs are always loaded at the absolute extremes of main storage, except for the executive and other real time programs, even if this involves swapping other programs.

Executive transient (nonresident) routines are also loaded into the same main storage area as programs when space is available. However, programs are never swapped to satisfy an executive routine main storage requirement. The DA also controls the loading of these transient routines.

Whenever a system transient routine completes its current operation, the main storage area it occupies is not actually released, but is placed in a release-if-necessary condition. Such a routine is therefore still available for use, if necessary, until the main storage space it occupies is required for some other operation. If the transient routine is requested again before such an event, its main storage is returned to the in-use condition. Each such transient routine has associated with it a 'sticking priority', so that the more frequently a transient is used, the longer it tends to retain its main storage space after each period of operation. This procedure prevents unnecessary loading of transient routines, since they remain in main storage as long as it is possible to do so without interfering with the over-all operation of the system.

Definitions:

Core Queue – This is the priority structured list or queue into which program requests for main storage allocation are placed. It is this list from which the DA selects the next task to receive the desired allocation.

Core Seconds – This is the amount of time spent using the CPU or an I/O channel. Additional time is accumulated for ERs which require the loading of nonresident executive segments.

Core Block Seconds – This is core seconds multiplied by program size in core blocks and is accumulated in the PCT for accounting purposes. It is also the basis for demand/batch sharing.

Core Second Accumulation – This is core seconds plus voluntary wait time. Voluntary waits include TWAIT$ and WAIT$ requests for user response from a demand terminal. This value is accumulated in the PCT for accounting purposes.

Core Quantum – The core quantum is the number of core seconds which a program is allowed to accumulate before it becomes available for swapping. The core quantum is computed each time the program is loaded. A program is not swappable until the first quantum is exceeded or it enters a voluntary wait state.

Core Priority – This value defines a program's position on the core queue and dictates how other programs may cause it to be swapped. For demand programs, the core priority is reduced one level and the quantum doubled each time a core quantum is exceeded. The priority is reset to the highest allowed for the program when the program responds to the terminal (via READ$ or TREAD$ requests).

25.5.5.3. DEMAND/BATCH SHARING

The following parameters provide basic input to the demand/batch sharing algorithm. They are alterable by the operator:

- DMIN – Minimum percent of computer usage guaranteed to demand, if requested.
- DMAX – Maximum percent of computer usage allowed to demand runs when batch runs are requesting service.
- DINC – Increment in percent to be added to DMIN for each active demand run. The percentage used is DMIN + (number of active terminals *DINC), except that if this value exceeds DMAX, DMAX is used.

Total core block seconds are accumulated by program type. The number of core blocks swapped are also accumulated. The blocks swapped are multiplied by average swapping time to convert to core block seconds and the result is added to the demand core block seconds. These values are used to determine the necessity of any sharing adjustments.

The goal of demand/batch sharing is to provide for maintaining some minimum batch thruput. The values used to control the sharing, represent that percentage of the normal batch thruput capability which may be used for demand processing. The exact correspondence between the percentage and thruput varies somewhat with program mix but should still maintain an approximately linear scale.

If demand programs are using in excess of the percentage allotted to them, and batch programs are waiting, the highest priority deadline or batch program is given a larger than normal quantum and is queued ahead of all demand programs. Demand programs do not cause the swapping of deadline programs under any circumstances unless the allotted demand minimum percentage is not being met.

25.5.5.4. TIMESHARING

Timesharing applies to demand tasks only and the control of response time. For a program whose size in core blocks is given by s, the cost of loading is proportional to s squared. The I/O transfer time is proportional to s and s core blocks are tied up for the duration of the transfer.

The queueing algorithms are designed such that a program at level n is loaded twice as often as a program at level n+1. The highest level allowed for a demand program is then determined as a linear function of s squared. This is the level at which the program initially receives control and the level to which it is raised when requesting input from the terminal.

The quantum allocated to the program is computed from the average time required to load the program. The quantum is constrained to be at least equal to this time, and is adjusted by @RUN priority and is doubled for each level below the highest level allowed for the program.

High priority programs must not be allowed to dominate the system to the exclusion of all others. In order to appropriately allocate system resources to all users, the following algorithm is applied to the core queue:

If the highest priority level is 2, the sequence of servicing is:

2,3,2,4,2,3,2,5,2,3,2,4,2,3,2,6,...

This results in each lower level being loaded half as often as the one above it with an even distribution of levels in the sequence.

25.5.6. DISPATCHER

The dispatcher has prime responsibility for controlling CPU usage. In a broad sense, dispatching includes all CPU control decisions; more commonly it applies just to switching among activities.

The CPU usage algorithm may be characterized as "pure preemptive". That is, if some process (typically an activity) having a certain priority is using a CPU and some event occurs requiring performance of a higher priority process, then the lower priority process is immediately preempted in favor of the higher. Conversely, if the new process is of lower or equal priority, it must wait until no higher priority demands for CPU service exist.

Two basic classes of CPU usage are defined: interlock and switchable.

25.5.6.1. INTERLOCK PROCESSING

Interlock processing is logically noninterruptable and nonswitchable. That is, once an interlock process begins, it is executed to completion and cannot, for example, be suspended and switched to another CPU or interleaved with other interlock processes of the same level. Interlock processes typically use the executive control registers (a privileged duplicate set of registers not usable under guard mode). They always have higher priority than any switchable process (activity). Three levels of interlock processing are defined:

(1) Interrupt Preprocessing — This is the highest level in the system, since the hardware locks out further interrupts on occurrence of an interrupt. Generally, interrupt preprocessing involves merely capturing essential information concerning the interrupt and recording it (typically in a queue) for subsequent processing. (See 25.7 for further discussion).

(2) ESI Processing — This is the next highest level. (See Sections 15 and 16 for details).

(3) Executive Interlock — This is the lowest level of interlock processing. Most interlock processing occurs at this level. It includes the main dispatcher, internally specified index (ISI) interrupt processing, clock control and nonESI (externally specified index) internal interrupt handling (ERs, Test And Set, and program faults). The switching routine of the main dispatcher is exercised only when no other interlock level processing remains.

The hierarchy among interlock levels is achieved by reserving some of the executive registers for each level (executive interlock is given all but a few).

25.5.6.2. SWITCHING

All processing below interlock levels is done by activities. The selection and control of multiple activities competing for CPU time is the principal concern of the dispatcher.

For switching purposes, activities are grouped into six types, and within each type there are up to 35 levels, numbered in order of priority from 1 to 35. Level 1 of each type is reserved for interrupt activities. The six activity types are:

(1) High Exec Workers – This type is relatively scarce and is generally limited to those exec workers which must run above real time; for example, the abort routine.

(2) Real Time User Activities – See Section 16.

(3) Low Exec Workers – Includes most executive workers and all transient routines. Important levels include:

2	– Symbionts
3	– Mass Storage Allocation
4	– DA Core Contents Control
5	– Main DA
6	–Logging and Accounting
10	– Coarse Scheduler, Facilities Inventory
11	– DA Periodic Adjustment
22	– Unsolicited Keyins, Checkpoint/Restart

(4) Demand Activities

(5) Deadline Batch Activities

(6) Normal Batch Activities

Real time and executive activities (types 1, 2, and 3) are switched with "infinite quantum". That is, an activity may execute indefinitely without "supervision" at the same level for as long as it wishes. Level control is the responsibility of the activity; no automatic level manipulation is performed by the executive (as is done for demand and batch, described next). Within types 1, 2, and 3, switching priority is simply by type (1 above 2 above 3) and by level within type.

Demand and batch activities (type 4, 5, and 6) are switched in a much more complex manner. The algorithm to be described has the fundamental objective of maximizing throughput by the strategy of:

(a) giving high priority for short periods to activities that request synchronous I/O service, and

(b) giving lower priority for longer periods in proportion to the amount of computation an activity does.

This strategy is intended to maximize utilization of peripheral equipment while reducing switching overhead for computational processes.

The demand/batch CPU switching algorithm operates as follows:

- Priority is by level (1 above 2 above 3...) and by type within level (contrast with real time and execitive).
- Associated with each level is a quantum of CPU time directly proportional to the power of two of that level (that is, a level n quantum is twice that of level n–1).
- Initial program activities start at level 2.

- As an activity executes, the amount of CPU time it consumes is accumulated in a quantum-used cell. When quantum-used exceeds the assigned quantum for the activity's level, the activity is dropped one level in priority, its quantum-used is cleared, and it is assigned a new quantum (twice as long); if the bottom level is reached, the same length quantum is assigned at the same level. In either case, the activity is placed at the end of the list of activities to be switched for the new level (and type).

- If an activity issues a synchronous request (ER) for service that involves I/O, the activity is raised to level 2 and assigned a full level 2 quantum.

- When an interrupt activity (level 1) is activated upon completion of its associated I/O request, it is issued a level 1 quantum. Expiration of that quantum, or any ER, causes the interrupt activity to be dropped to the level of the original activity.

- Forked activities receive the same quantum and level as the original activity.

- Quantums are timed with the real time clock. Whenever an activity is given control, the time remaining for its quantum is computed and the real time clock is set to interrupt no later than the computed time of quantum expiration.

- Standard quantum values are 16 milliseconds for level 2. Currently seven levels are provided for types 4, 5, and 6, thus making the longest (level 7) quantum 512 milliseconds. These times are doubled for the 1106 system.

Once any level control operations are performed, switching consists of simply taking the highest priority activity from a list, called the switch list, of all activities currently requiring CPU service. If the selected activity is of higher priority than the currently executing activity (if any), the latter is suspended and control switches to the selected activity. Otherwise, no switch occurs. If there are no activities requiring CPU time, the dispatcher goes into an idle loop until an interrupt occurs or, on multiprocessor system, an activity appears on the switch list. Note that the dispatcher is reentrant and can operate simultaneously on multiple CPUs by protecting the common switch list data at appropriate critical points.

25.6. CLOCKING

The clocking routines make provisions for clock usage by an object program as well as by the system. These routines are available to the user by means of ERs. The clocking routines serve as the basis for all accounting and logging functions, as well as a source of control for many real time applications.

25.6.1. REAL TIME CLOCK

The real time clock routine is used by the system for timing various activities such as I/O functions, operator responses and CPU usage time for each run. This routine is also used by the system to force interrupts after variable amounts of time so that such events as nonresponsive I/O devices, unbalanced usage of CPU time, and so forth are detected. The frequency of interrupt depends on the needs of the system. The routine is so designed that many events may be simultaneously timed, and many interrupts may be simultaneously requested.

25.6.2. DAY CLOCK

The day clock routine is used by the system to maintain an accurate, standard time. This time is used by all processors for annotating listings, by the file control system for maintaining historical information about all files, by the accounting and logging routines for time-tagging events, as well as by other routines for other functions.

25.7. INTERRUPT HANDLING

In general, all interrupts are handled by the interrupt handler. This routine either queues the interrupt or routes control to the appropriate routine to handle the interrupt. Interrupts are received from either the control section of the CPU or from a peripheral subsystem. The interrupts fall into five general categories as follows:

(1) Input/Output Interrupts

(2) Clock Interrupts

(3) Interprocessor Interrupts

(4) Hardware Fault Interrupts

(5) Program-Generated Interrupts

The handling of the real time and day clocks is discussed in 25.6.1 and 25.6.2. This paragraph discusses the handling of the other four classes of interrupts.

25.7.1. INPUT/OUTPUT INTERRUPTS AND QUEUEING

When an I/O interrupt occurs, the type of interrupt and the channel on which it occurred is either queued or routed to the appropriate processing routine. The three types of I/O interrupts are:

- ISI and ESI external interrupts
- ESI and ISI data termination (monitor) interrupts
- ISI function termination (function monitor) interrupts

ISI interrupts are queued if:

- higher priority interrupts are already in the queue;
- another interrupt is being processed by an interlock level routine; or
- the CPU was interrupted while operating in the ESI mode.

ISI interrupts are removed from the queue by channel priority, channel zero having the highest priority.

ESI interrupts are queued if all CPUs are currently busy on other ESI-level work. ESI interrupt handling is further discussed in Sections 15 and 16.

25.7.2. INTERPROCESSOR INTERRUPTS

Interprocessor interrupts are issued within the executive via the privileged Initiate Interprocessor Interrupt (III) instruction. This feature allows one CPU to divert another CPU for the following reasons:

(a) Issue an I/O request to a device which cannot be accessed by the first CPU (due to lack of direct electrical path).

(b) Deactivate an activity which is currently executing on the other CPU; for example, for program suspension prior to swapout.

(c) Process outstanding ESI interrupts in order to distribute the ESI load over the entire system, in the event that the initiating CPU is already busy processing an ESI interrupt.

(d) System start-up.

Prior to issuing the III instruction, the sending CPU sets a lock (Test And Set) and stores a function code in a common cell. When the receiving CPU is interrupted, control is routed to the interprocessor interrupt handler. This handler retrieves the function code, clears the lock, and routes control according to the function code, which defines some action to be taken under one of the four categories previously discussed (see 25.7 and 25.7.1).

25.7.3. HARDWARE FAULT INTERRUPTS

25.7.3.1. STORAGE AND ICR PARITY ERROR INTERRUPTS

The type of interrupts in this category are:

(1) Storage Parity Errors

(2) Input/Output Data Parity Errors

(3) ICR Parity Errors

(4) Access Control Word Parity Errors

An interrupt of this type causes the system to halt. The only way to recover from these errors is to reload the system.

25.7.3.2. POWER LOSS INTERRUPTS

When a CPU receives a power-loss interrupt, three steps are taken to prepare for the total loss of power:

(1) Save the control register contents of the interrupted CPU.

(2) Disconnect all I/O channels of the interrupted CPU.

(3) Go into a Jump Greater And Decrement (JGD) loop and wait for the total loss of power.

If the CPU is still running at the end of the JGD loop, a transient power-loss interrupt has occurred; and the following steps are taken to recover the CPU:

(1) Restore all control registers.

(2) Reactivate all previously active ESI channels. In the case of ISI channels, an I/O TIMEOUT message will occur and an operator answer will reissue the command on the channel.

(3) If a user or executive activity was interrupted, set up the interrupted activity's PSR and SLR and return control to the activity. If the above is not true, return control to the executive at its interrupted point.

In the event of a real power loss, the operator can recover the CPU without rebooting the system.

When the CPU is started, according to the power loss recovery procedures, all internal registers are reset; and the recovery procedure for a transient interrupt is followed.

25.7.4. PROGRAM–GENERATED INTERRUPTS

This class of interrupts includes all those which occur immediately as the result of program instructions (in some cases, these may occur inside the executive). There are three subclasses of program interrupts:

(a) Program Fault Interrupts:*

- Illegal Operation
- Guard Mode Fault

(b) Arithmetic Exception Interrupts:*

- Floating-Point Overflow
- Floating-Point Underflow
- Divide Fault

(c) Programmed Interrupts:

- Executive Request (see Section 4)
- Test And Set Conflict (see 2.4.5, and Section 16)

25.8. CATALOGUED FILE RECOVERY

When the system is reloaded following a hardware or software failure, the executive reconstructs the master file directory of all catalogued files. The catalogued file recovery routine (CATFR) is executed during the initial phases of system reloading following the determination of the current hardware status.

All master file directory information on all mass storage units is read and verified, the look up table and link addresses are recalculated, and mass storage is reallocated. These operations are performed on a unit by unit basis in order to minimize the amount of information which is lost should a unit be lost to the configuration or be partially overwirtten.

In order to perform these operations as efficiently as possible, CATFR utilizes all available main storage. Independent, parallel I/O operations are performed on all available mass storage channels. After main storage is filled with master file directory items, the validation phase begins. Each file is processed independently and multiple validation activities are used to perform this operation as rapidly as possible.

The master file directory information for each file consists of linked items. If a forward link address or granule table entry points to a downed unit or is a nonexistant address, the address is removed, the granule or last part of the file is lost, and the file is marked disabled in a special way ("hardware disabled") such that an attempt to assign the file results in a FAC REJECT message unless the Q option is used on the @ASG control statement.

If the file was not assigned as read only at the time of the system stop, the file is marked disabled (standard disable) as it was possibly in a state of flux at the time of the stop. An assign of a disabled file is accompanied by a FAC WARNING message. Files which were still in the +1 F-cycle state, marked to be dropped, or whose FORWARD ITEM links were bad are dropped.

Following the verification, the granule counts are updated for the number of granules recovered and the file is entered into the look up table, used by the facilities mechanism of the executive.

In addition to recovering as much of the files as possible, CATFR's validation ensures that all information required by the executive is present and correct. Unless some error is detected, the operation of CATFR is transparent to the user.

See 8.2.17 and Section 19 for further information relating to disabled files.

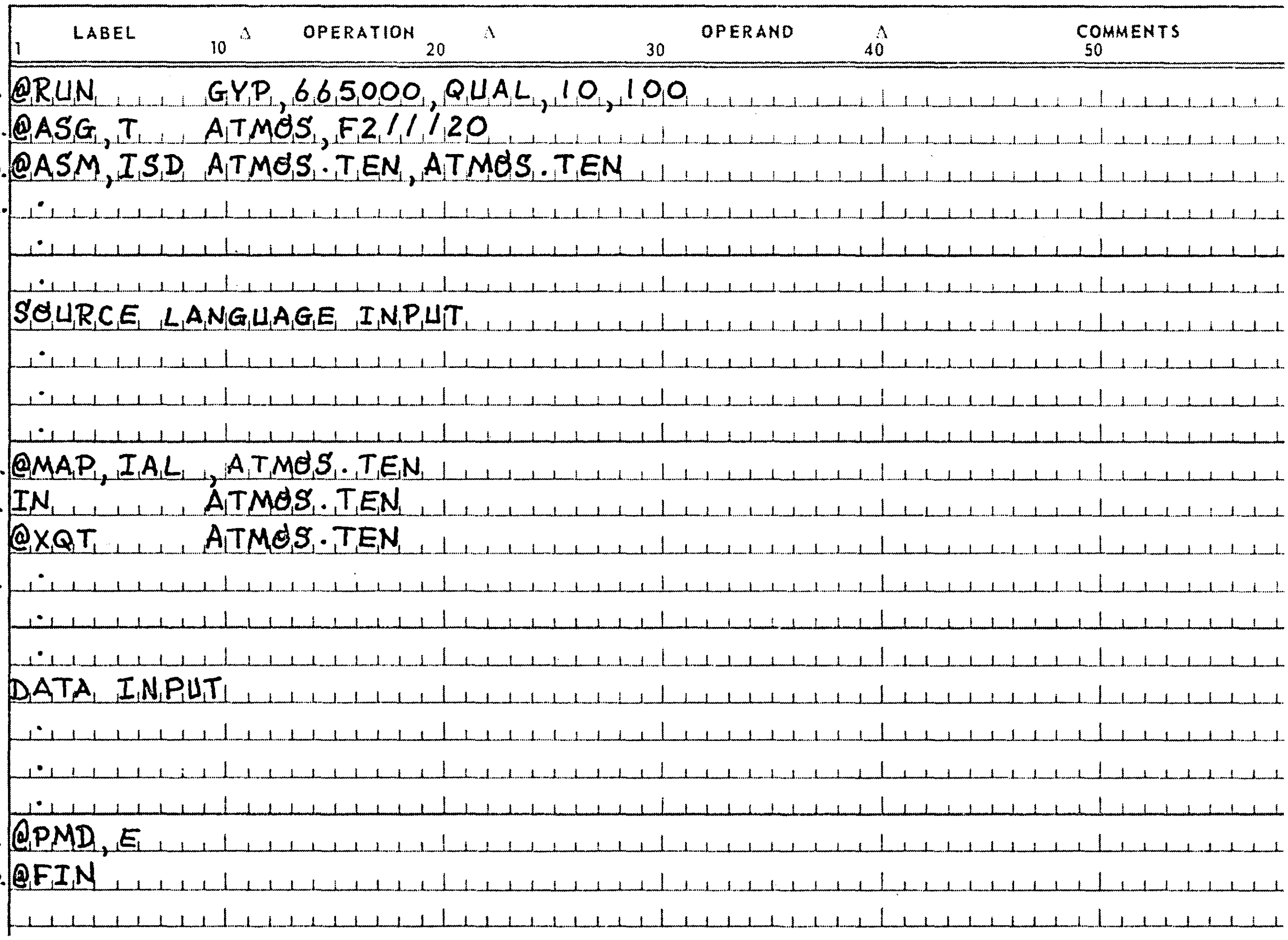
**These are discussed in 4.9.*

26. RUN SETUP EXAMPLES

The following examples illustrate run streams used in executing various jobs.

Example 1:

The following run stream processes an assembly language program.



```
    LABEL      Δ    OPERATION      Δ          OPERAND       Δ          COMMENTS
 1            10              20             30            40             50
 1. @RUN        GYP,665000,QUAL,10,100
 2. @ASG,T      ATMØS,F2///20
 3. @ASM,ISD    ATMØS.TEN,ATMØS.TEN
 4.  .
     .
     .
    SØURCE LANGUAGE INPUT
     .
     .
     .
 5. @MAP,IAL ,ATMØS.TEN
 6. IN          ATMØS.TEN
 7. @XQT        ATMØS.TEN
 8.  .
     .
     .
    DATA INPUT
     .
     .
     .
 9. @PMD,E
10. @FIN
```

1. @RUN control statement: GYP is the run-id; 665000 is the account number; QUAL is the project-id; estimated run time is 10 minutes; and estimated page count is 100.

2. Assign file ATMOS as a temporary file on FASTRAND mass storage with maximum of 20 granules.

3. Assemble source language program TEN and produce a double-spaced listing consisting of both source language input and octal output. Insert symbolic element TEN into file ATMOS and produce á relocatable element called TEN in file ATMOS.

4. Source language input for program TEN.

5. Collect and produce absolute element TEN in file ATMOS. Accept the results of processing even if errors are detected and produce a comprehensive listing.

6. Input to the collector directing that element TEN in file ATMOS be included in the collection.

7. Execute absolute element ATMOS.TEN.

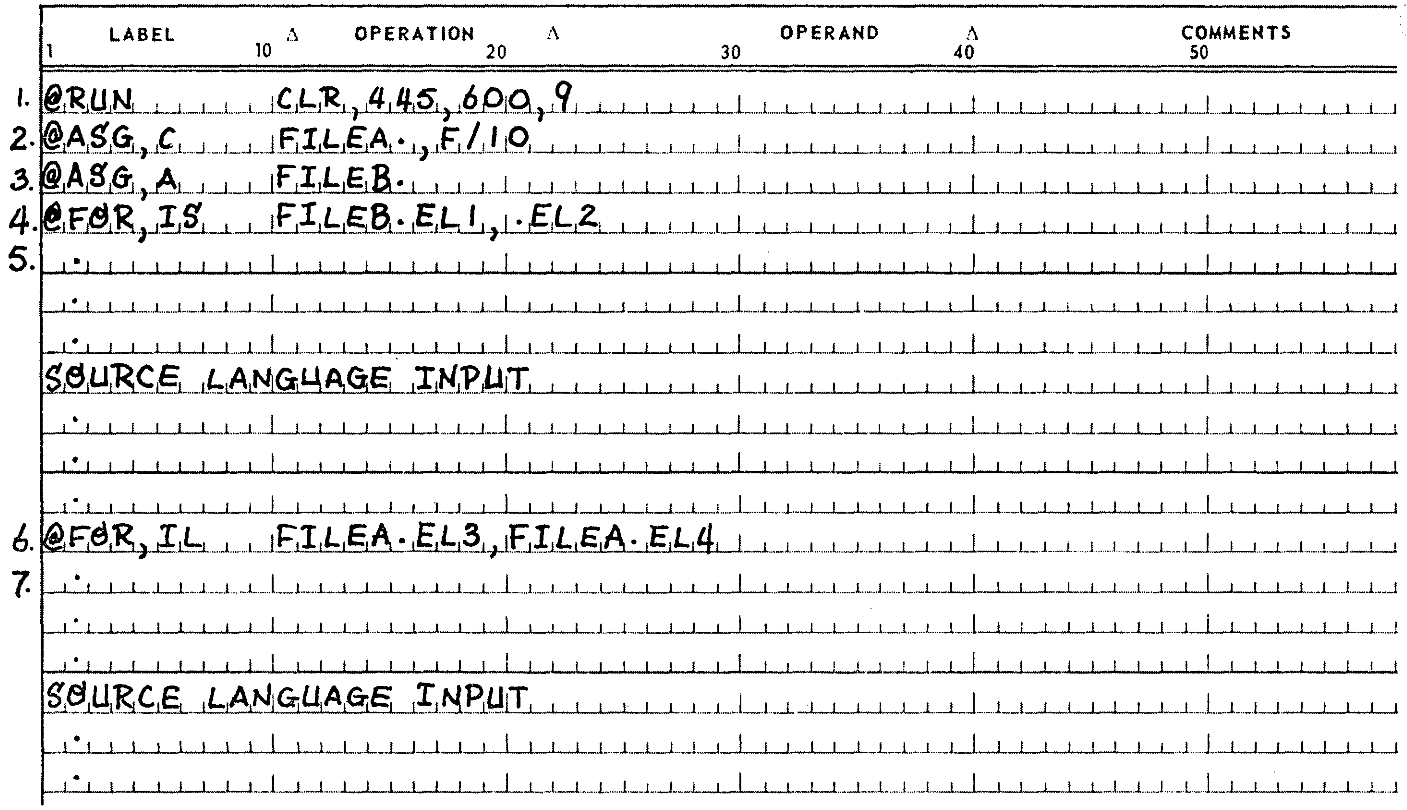
8. Data input necessary for program execution.

9. If the program terminates in error, a postmortem dump is taken.

10. @FIN control statement terminates the run.

Example 2:

The following runstream executes the absolute element ABS produced by the collector from three relocatable elements produced in the run and one element from a user-specified file.



```
     LABEL       Δ  OPERATION      Δ          OPERAND     Δ          COMMENTS
    1            10            20            30           40           50
 1. @RUN          CLR,445,600,9
 2. @ASG,C        FILEA.,F/10
 3. @ASG,A        FILEB.
 4. @FØR,IS       FILEB.EL1,.EL2
 5.  .
     .
     .
    SØURCE LANGUAGE INPUT
     .
     .
     .
 6. @FØR,IL       FILEA.EL3,FILEA.EL4
 7.  .
     .
     .
    SØURCE LANGUAGE INPUT
     .
     .
    .
8.  @ASM,IL     FILEA.EL5,.EL6
9.  .
    .
    .
    SOURCE LANGUAGE INPUT
    .
    .
    .
10. @MAP,IL     .SYM,.ABS
11. IN          FILEB.EL2
12. IN          FILEA.EL4
13. IN          FILEA.EL6
14. IN     .    FILEA.EL7
15. @XQT        .ABS
16. .
    .
    .
    DATA INPUT
    .
    .
    .
17. @FIN
```

1. @RUN control statement: CLR is the run-id; 445 is the account number; 600 is the project-id; and nine minutes is the estimated run time.

2. Assign file FILEA in FASTRAND format on FH-432, FH-880, or FH-1782 drums, FASTRAND mass storage, or 8414 disc with an initial granule reserve of 10 tracks. FILEA is to be catalogued if the file is deassigned with a @FREE control statement or if the run terminates normally.

3. Assign previously catalogued file FILEB.

4. Calls FORTRAN compiler. A new symbolic element EL1 is to be inserted into file FILEB. A listing is produced on the new source language and a relocatable output element EL2 is placed in FILEB.

5. Source language input deck.

6. Calls FORTRAN compiler. A new source language element EL3 is inserted in FILEA. A comprehensive listing is produced of element EL3. The output relocatable element EL4 is placed in FILEA.

7. Source language input deck.

8. Calls assembler. A new source language element EL5 is inserted in FILEA, and a complete listing of the element is produced. A new relocatable element EL6 is placed in FILEA.

9. Source language input deck.

10. Calls the collector. Insert new MAP source language element SYM in the temporary program file (TPF$). Lines 11 through 15 are the source language statements which will comprise element SYM. Produce the absolute element ABS in file TPF$ based on directives contained in element SYM.

11. Include relocatable element EL2 from file FILEB in the collection to produce element ABS.

12. Include relocatable element EL4 from file FILEA in the collection to produce element ABS.

13. Include relocatable element EL6 from file FILEA in the collection to produce element ABS.

14. Include relocatable element EL7 from file FILEA in the collection to produce element ABS.

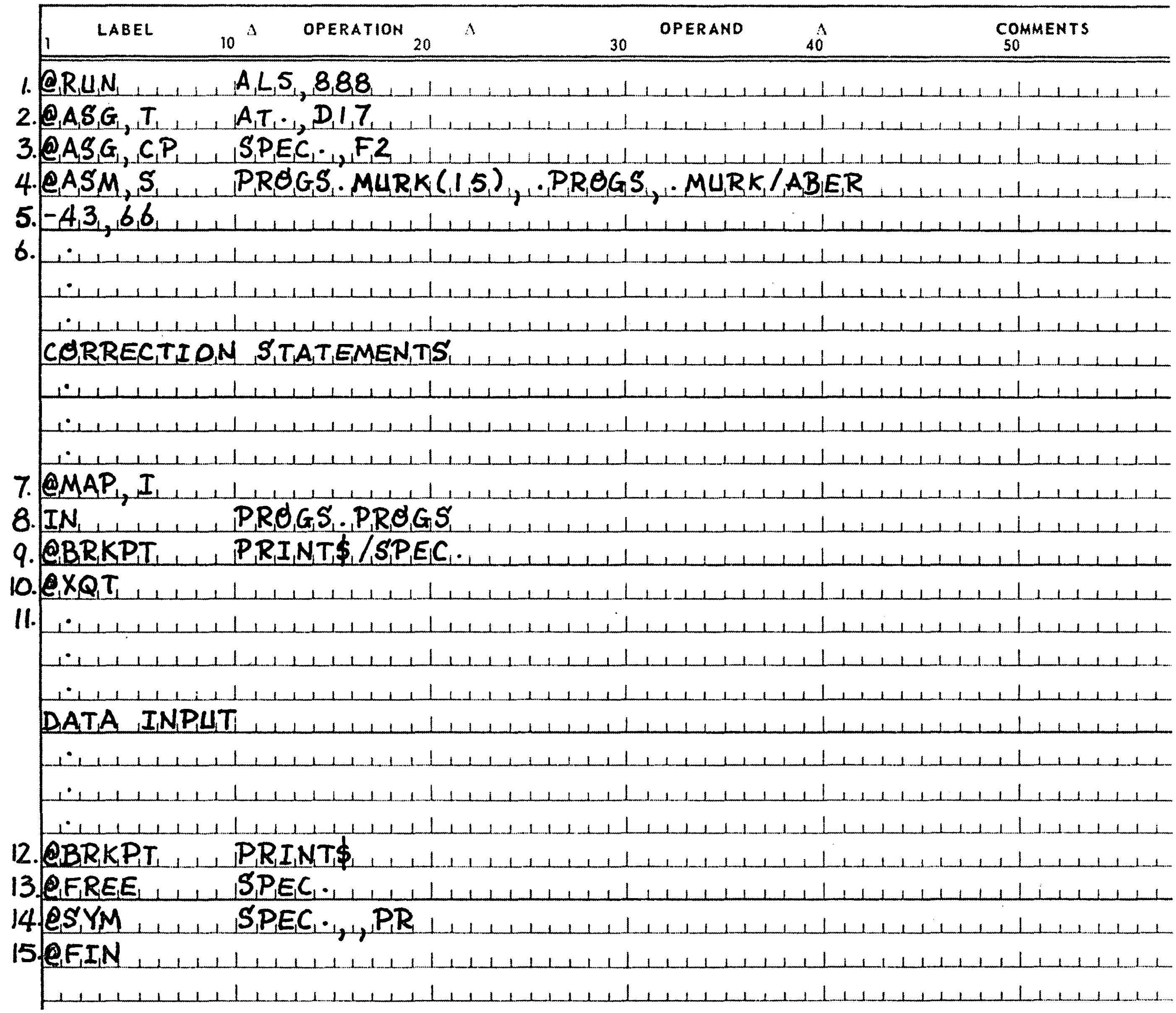
15. Execute absolute element ABS located in file TPF$.

16. Data input necessary for program execution.

17. @FIN control statement terminates the run.

Example 3:

In the following run stream, a program file is updated. A separate file is assigned to accept results of program execution; then deassigned; the contents made public; and queued for output on a printer subsystem.



```
     LABEL         OPERATION              OPERAND                COMMENTS
   1         10            20          30          40          50
1.  @RUN        ALS,888
2.  @ASG,T      AT.,D17
3.  @ASG,CP     SPEC.,F2
4.  @ASM,S      PROGS.MURK(15),.PROGS,.MURK/ABER
5.  -43,66
6.   .
     .
     .
    CORRECTION STATEMENTS
     .
     .
     .
7.  @MAP,I
8.  IN          PROGS.PROGS
9.  @BRKPT      PRINT$/SPEC.
10. @XQT
11.  .
     .
     .
    DATA INPUT
     .
     .
     .
12. @BRKPT      PRINT$
13. @FREE       SPEC.
14. @SYM        SPEC.,,PR
15. @FIN
```

1.. @RUN control statement: AL5 is the run-id, and 888 is the account number.

2. Assign file AT as a word-addressable temporary file on the FH-1782 drum.

3. Assign file SPEC on FASTRAND mass storage and catalogue it as a public file if one of the following occurs:

   - the file is deassigned during the run by a @FREE control statement; or

   - the run terminates without error.

4. Calls the assembler. Assembles element MURK(15) from file PROGS, and produces relocatable element PROGS and a new symbolic element MURK/ABER in file PROGS.

5. Replace lines 43 through 66 of element MURK(15) with the lines which follow (item 6) to produce MURK/ABER.

6. Correction statements to the source language element.

7. Calls the collector.

8. Include element PROGS from file PROGS in the collection.

9. Close out the currently active print file and start a new file SPEC.

10. Execute the program.

11. Data statements necessary for program execution.

12. Print file SPEC is closed and a new system-defined print file is opened.

13. File SPEC is catalogued and made available for assignment by other runs.

14. File SPEC is queued for output on a printer subsystem.

15. @FIN control statement terminates the run.

Example 4:

The following example illustrates two run streams. The first generates a program, copies the symbolic element to tape, and executes the program. The second references this tape file, inserts corrections, and produces a new tape file.

```
     LABEL        Δ    OPERATION    Δ            OPERAND     Δ          COMMENTS
  1            10               20             30             40            50
1. @RUN       INIT,12345,TR
2. @ASG,T     TAPE.,T,12
3. @FØR,IS    .PRØG
4.  .
    .
    .

   SØURCE LANGUAGE INPUT
    .
    .
```

```
LABEL          OPERATION               OPERAND                COMMENTS
1         10             20         30           40          50

    .
 5. @COPOUT,S  .PROG,TAPE.
 6. @MARK      TAPE.
 7. @XQT
 8. .
    .
    .
    DATA INPUT
    .
    .
    .
 9. @PMD,E
10. @FIN
11. @RUN,/S    NEW,12345,TR
12. @ASG,T     TAPE.,T,12
13. @ASG,T     NTAPE.,8C9
14. @ASG,T     PROGFILE., F/2/POS/5
15. @COPIN     TAPE.,PROGFILE.
16. @FREE      TAPE.
17. @FOR,S     PROGFILE.PROG,PROGFILE.RB,.PROG/A
    .
    .
    .
18. CORRECTION STATEMENTS
    .
    .
    .
19. @COPOUT    PROGFILE.,NTAPE.
20. @FREE      NTAPE.
21. @MAP,IL    .MAPSYM,.ABS
22. IN         PROGFILE.RB
23. @XQT       .ABS
24. .
    .
    .
    DATA INPUT
    .
    .
    .
25. @PMD,E
26. @FIN
```

1. @RUN control statement: INIT is the run-id; 12345 is the account number; and TR is the project-id.

2. Temporarily assign file TAPE to any magnetic tape unit, reel number is 12.

3. Calls FORTRAN compiler. A new source language input element PROG is inserted from the control stream into file TPF$ and a listing is produced. The relocatable element produced, PROG, is placed in TPF$.

4. Source language statements.

5. Copy program element PROG from mass storage to tape file TAPE.

6. Mark an end-of-file (EOF) on the tape.

7. Execute the program.

8. Data input necessary for program execution.

9. If the program terminates in error, a postmortem dump is taken.

10. @FIN control statement terminates the run.

11. @RUN control statement: NEW is the run-id; 12345 is the account number; and TR is the project-id.

12. Temporarily assign file TAPE to any tape unit, reel number is 12.

13. Temporarily assign file NTAPE to a UNISERVO VIII-C nine-track tape unit.

14. Assign file PROGFILE as a temporary file on mass storage with position granularity with an initial reserve of two granules and a maximum of five granules.

15. Copy input file TAPE from tape to mass storage file PROGFILE.

16. Release the tape unit assigned as file TAPE. It is good practice to release a facility when it is no longer needed by the run.

17. Calls FORTRAN compiler. Compile element PROG from file PROGFILE; produce relocatable binary element RB and a new symbolic element PROG/A.

18. Correction statements to apply against PROG to produce PROG/A.

19. Copy program file PROGFILE on mass storage to element file NTAPE on tape.

20. Release the tape unit assigned as file NTAPE.

21. Calls the collector. Produce symbolic element MAPSYM which contains the MAP directives and produce absolute output element ABS according to the directives.

22. Include element RB from file PROGFILE in the collection.

23. Execute absolute program element ABS located in TPF$.

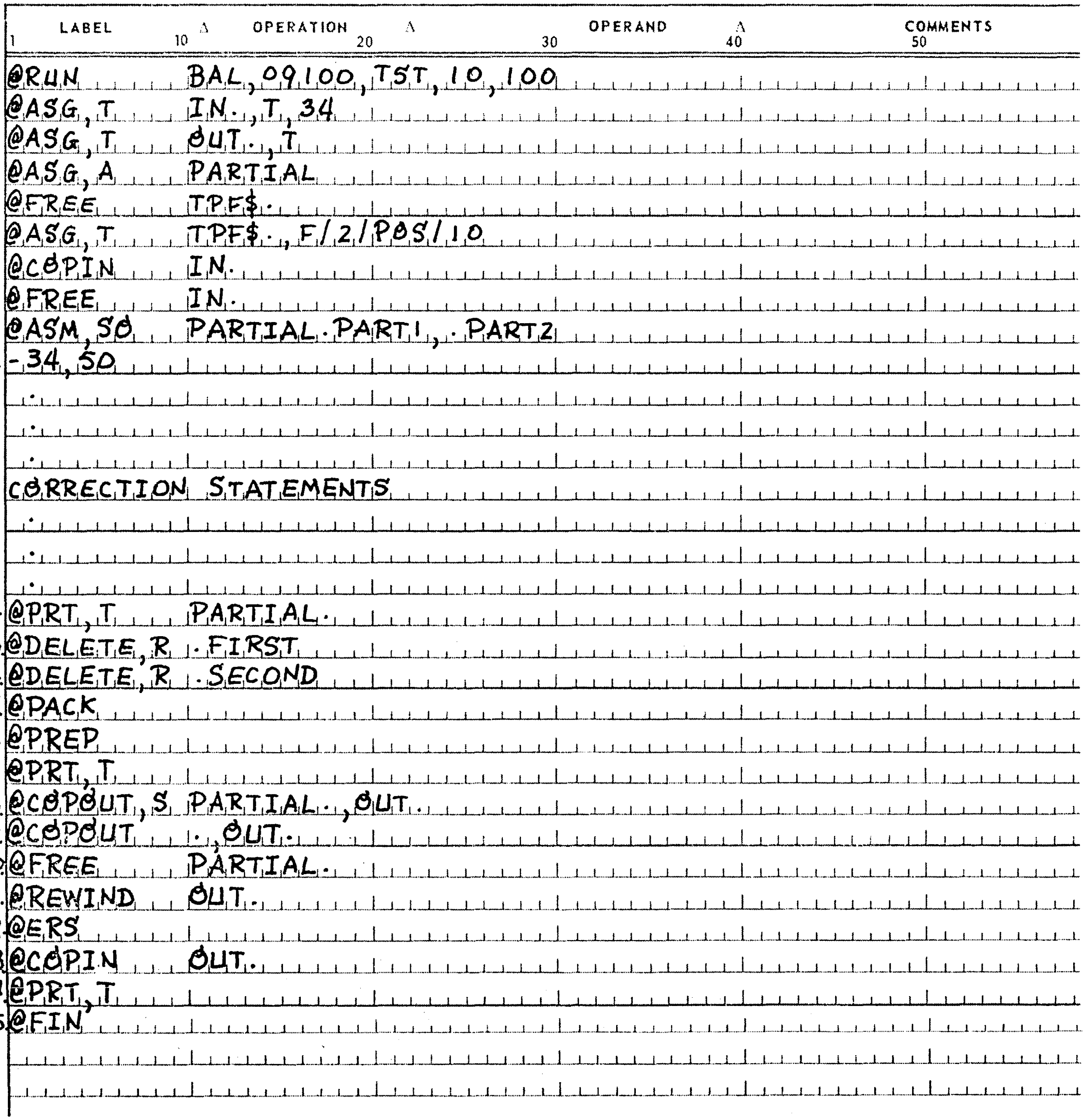
24. Data input necessary for the program.

25. Produce a postmortem dump if the program terminates in error.

26. @FIN control statement terminates the run.

Example 5:

The following run stream inserts corrections into program elements of file PARTIAL and produces a tape file which contains elements from PARTIAL and tape file IN.



```
     LABEL            OPERATION               OPERAND                COMMENTS
 1               10               20               30               40               50
1.  @RUN           BAL,09100,TST,10,100
2.  @ASG,T         IN.,T,34
3.  @ASG,T         ØUT.,T
4.  @ASG,A         PARTIAL
5.  @FREE          TPF$.
6.  @ASG,T         TPF$.,F/2/PØS/10
7.  @CØPIN         IN.
8.  @FREE          IN.
9.  @ASM,SØ        PARTIAL.PART1,.PART2
10. -34,50
11.   .
      .
      .
    CØRRECTION STATEMENTS
      .
      .
      .
12. @PRT,T         PARTIAL.
13. @DELETE,R      .FIRST
14. @DELETE,R      .SECOND
15. @PACK
16. @PREP
17. @PRT,T
18. @CØPØUT,S      PARTIAL.,ØUT.
19. @CØPØUT        .,ØUT.
20. @FREE          PARTIAL.
21. @REWIND        ØUT.
22. @ERS
23. @CØPIN         ØUT.
24. @PRT,T
25. @FIN
```

1. @RUN control statement: BAL is the run-id; 09100 is the account number; TST is the project-id; the estimated run time is 10 minutes; and estimated page count is 100.

2. Temporarily assign file IN to an available tape device, reel number is 34.

3. Temporarily assign file OUT to an available tape device.

4. Assign file PARTIAL which was previously catalogued.

5. Release the facilities assigned to TPF$.

6. Temporarily assign TPF$ to a mass storage device with position granularity, an initial granule reserve of two granules and a maximum of ten granules.

7. Transfer program file IN from tape to mass storage file TPF$.

8. Release the facilities assigned to file IN.

9. Calls the assembler. Assemble element PART1 from file PARTIAL and produce relocatable element PART2. Produce a listing of only the octal output.

10. Delete lines 34 through 50 of the source language input element.

11. Correction statements.

12. Calls FURPUR processor. Produce an edited listing of the table of contents of file PARTIAL.

13. Calls FURPUR processor. Mark relocatable element FIRST in TPF$ as deleted.

14. Calls FURPUR processor. Mark relocatable element SECOND in TPF$ as deleted.

15. Calls FURPUR processor. Pack the nondeleted elements in TPF$.

16. Prepare an entry point table for TPF$.

17. Calls FURPUR processor. Print the table of contents and the entry point table for TPF$.

18. Calls FURPUR processor. Copy the symbolic elements from file PARTIAL intc file OUT.

19. Calls FURPUR processor. Copy the contents of TPF$ onto file OUT.

20. Release the facilities assigned to file PARTIAL.

21. Calls FURPUR processor. Rewind file OUT to its load point.

22. Calls FURPUR processor. Reset the next write location of TPF$ back to the start of the file and release its granules to the system.

23. Transfer file OUT from tape to mass storage file TPF$.

24. Calls FURPUR processor. Produce an edited listing of the table of contents of file TPF$.

25. @FIN control statement terminates the run.

Example 6:

The following run stream builds a boot tape from a tape file that contains relocatable binary elements.

```
    LABEL     Δ   OPERATION    Δ           OPERAND      Δ             COMMENTS
1          10             20              30             40             50
@RUN,A    SYSCEN,400400,BUILD,15,200
@ASG,T    RB.,C,3434C
@ASG,T    BOOTFILE.,C74545C
@ASG,T    RØ.,F2/2/PØS/30
@MØVE     RB.,2
@CØPIN    RB.,RØ.
@FREE     RB.
@HDG      TØCS
@PRT,T    RØ.
@USE      EXEC$,RØ
@HDG,P    EXEC-8  MAP
@MAP,S    RØ.EX8MAP,RØ.EX8MAP
@MAP,I    ,.SCH8PF
IN        EXEC$.SCH8PF
@XQT      .SCH8PF
@MSG      BUILD BØØTFILE
@XQT      EXEC$.EX8MAP
@USE      B,BØØTFILE
@ASG,T    ØLDBØØT.,C,6565C
@MØVE     ØLDBØØT.,1
@CØPY,M   ØLDBØØT.,B,5
@FREE     ØLDBØØT.
@REWIND   B.
@FREE     TPF$.
@ASG,T    TPF$.,F2/2/PØS/20
@MØVE     B.,1
@CØPIN    B.
@PRT,T
@ERS
@CØPIN    B.
@PRT,T
@FIN
```

1. @RUN control statement with A priority; SYSCEN is the run-id; 400400 is the account number; BUILD is the project-id; and estimated run time is 15 minutes; estimated page count is 200.

2. Temporarily assign filename RB to any UNISERVO VIII-C, VI-C, or IV-C tape unit; reel number is 3434C.

3. Temporarily assign file BOOTFILE to any UNISERVO VIII-C, VI-C, or IV-C tape unit; reel number is 4545C.

4. Temporarily assign file RO to FASTRAND II or III mass storage unit with position granularity an initial reserve of two and a maximum size of 30 granules.

5. Calls FURPUR processor. Move tape file RB over two EOF marks.

6. Transfer tape file RB to mass storage file RO.

7. Release the facilities assigned to file RB.

8. Print the heading TOCS at the top of each page.

9. Calls FURPUR processor. Produce an edited listing of the table of contents of file RO.

10. The internal file name EXEC$ is attached to file RO and the file can be referenced by either name throughout the remainder of this run.

11. Print the heading EXEC-8 MAP beginning at page one instead of at the print file's current page number.

12. Calls the collector. Use the directives in symbolic element EX8MAP in file RO to produce an absolute element of the same name.

13. Calls the collector. Use the directive which immediately follows in the run stream and produce absolute element SCH8PF in file TPF$.

14. Relocatable element SCH8PF in file EXEC$ is to be included in the collection.

15. Execute absolute element SCH8PF which is in file TPF$.

16. Transmit the message BUILD BOOTFILE to the operator's console.

17. Execute element EX8MAP in file EXEC$.

18. Internal file name B is attached to file BOOTFILE and the file can be referenced by either name throughout the remainder of the run.

19. Temporarily assign file OLDBOOT to any UNISERVO VIII-C, VI-C, or IV-C tape unit; reel number is 6565C.

20. Calls FURPUR processor. Move tape file OLDBOOT over one EOF mark.

21. Copy five files from OLDBOOT to tape file B and place one EOF mark after each file copied to tape file B.

22. Release the facilities assigned to file OLDBOOT.

23. Rewind tape file B to its load point.

24. Release the facilities assigned to TPF$.

25. Temporarily assign TPF$ to FASTRAND mass storage with initial granules reserve of two, position granularity, and a maximum size of 20 granules.

26. Move tape file B past one EOF mark.

27. Transfer tape file B to mass storage file TPF$.

28. List the table of contents for TPF$.

29. Reset the next write location of TPF$ back to the start of the file.

30. Transfer tape file B to mass storage file TPF$.

31. Print the table of contents of file TPF$.

32. @FIN control statement terminates the run.

APPENDIX A. SUMMARY OF CONTROL STATEMENTS

@ADD Provides a means of inserting images into the run stream from any file currently assigned to the user or any catalogued file, provided that the file is FASTRAND-formatted in SDF format or from any source element of a program file.

```
@label:ADD,options    filename.eltname
```

@ALG Call the ALGOL language processor to translate source language instructions into relocatable binary code. The output is saved in a new relocatable element.

```
@label:ALG,options    eltname-1,eltname-2,eltname-3
```

@ASG Assigns physical facilities to a given run under the specified filename.

For FASTRAND-formatted mass storage, word addressable mass storage, and simulated word addressable mass storage:

```
@label:ASG,options    filename,type/reserve/granule/maximum,packid-1/packid-2.../packid-n
```

For magnetic tape files:

```
@label:ASG,options    filename,type/units/log/noise,/MSA-trans/unit-trans/format,;
        reel-1/reel-2.../reel-n,expiration-period
```

For all devices except FASTRAND mass storage, drum, and magnetic tape units. Used primarily for the assignment of special I/O devices and communication equipment.

```
@label:ASG    filename,type
```

@ASM Call the assembly language processor to translate source language instructions into relocatable binary code. The output is saved in a new relocatable element.

```
@label:ASM,options    eltname-1,eltname-2,eltname-3
```

@BRKPT Enables the user to establish breakpoints in the current print and punch files (PRINT$, PUNCH$, and so forth).

```
@label:BRKPT,options    generic-name/part-name
```

@CAT Catalogues one or more files independently of assigned facilities.

```
@label:CAT,options    filename,read-key/write-key,type/noise/MSA-trans/unit-trans/format,;
        reel-1,reel-2,reel-n
```

```
@label:CAT,options    filename,type/reserve/granule/maximum,packid-1/packid-2.../packid-n
```

@CHG Changes element name, filename, version name, read key, write key, and mode of a file.

```
@label:CHG,options   name-1,name-2
```

@CKPAR Initiates a program checkpoint.

```
@label:CKPAR,options   filename.element
```

@CKPT Saves the complete status of a run at a given point.

```
@label:CKPT,options   filename
```

@CLOSE Writes two hardware EOF marks on a magnetic tape file and rewinds the tape.

```
@label:CLOSE,options   filename-1,filename-2,...filename-n
```

@COB Call the COBOL language processor to translate source language instructions into relocatable binary code. The output is saved in one main relocatable output element and possibly in several other dependent relocatable elements.

```
@label:COB,options   eltname-1,eltname-2,eltname-3
```

@COL Permits the user to change the read mode from the system-defined standard to the read mode specified by the first parameter on the control statement. The @COL control statement is only valid when read from an onsite card reader.

```
@COL   xx,sentinel
```

@CON78 Accept CUR-formatted symbolic elements on magnetic tape and convert them to FURPUR-formatted symbolic elements.

```
@label:CON78,options   filename-1,filename-2
```

@COPIN Copies elements from an element file located on magnetic tape into a program file on FASTRAND-formatted mass storage.

```
@label:COPIN,options   name-1,name-2
```

@COPOUT Copies a program file, or seleted elements from a program file, located on FASTRAND-formatted mass storage onto a magnetic tape file in element file format.

```
@label:COPOUT,options   name-1,name-2
```

@COPY Copies a file or element from one file to another.

```
@label:COPY,options   name-1,name-2,nbr-of-files
```

@CULL Call the CULL processor to produce an alphabetically sorted, cross-referenced listing of all symbols found and the elements and lines on which they occur.

```
@label:CULL,options   pro/scol(res),name-1,...name-n
```

@CYCLE Sets the maximum range of absolute F-cycle numbers to be retained for specified files which are listed in the master file directory or sets the maximum number of element cycles to be retained for a specified symbolic element.

```
@label:CYCLE   name,n
```

@DATA Introduces, updates, and corrects data files from the control stream.

```
@label:DATA,options   filename-1,filename-2,sentinel
```

@DELETE Drops catalogued files or marks elements in a program file as deleted.

```
@label:DELETE,options   name-1,name-2,...,name-n
```

@ED Allows the user to conversationally edit a symbolic file or element by permitting insertion, deletion, and replacement of text.

```
@label:ED   name-1,name-2
```

@ELT Introduces an element into a particular program file or makes corrections to a symbolic element in a program file from the run stream.

```
@label:ELT,options   eltname-1,eltname-2,sentinel
```

@ENABLE Removes disable state from catalogued files.

```
@label:ENABLE   filename-1,filename-2,...,filename-n
```

@END Notifies the system that this is the end of control stream images that are to be transferred as data by the previous @ELT,D or @DATA control statements.

```
@END   s
```

@ENDCL Terminates the mode established by the @COL control statement.

```
@ENDCL
```

@ENDX When encountered by an READ$ request in CLIST$ mode, a return with a CLIST index of 77_8 occurs.

```
@ENDX
```

@EOF When encountered by an READ$ request, this control statement causes an end-of-file return.

```
@EOF   s
```

@ERS Resets next write location to the first sector of the text area, clears the table of contents, and returns to the system all FASTRAND-formatted mass storage allocated to a program file.

```
@label:ERS   filename-1,filename-2,...,filename-n
```

@FIN Identifies the end of a run. The @FIN control statement must appear as the last statement in all runs.

```
@FIN
```

@FIND Locates an element in a magnetic tape file (file must be in element file format) and positions the file before the element's label block.

```
@label:FIND,options   eltname
```

@FOR Call the FORTRAN language processor to translate source language instructions into relocatable binary code. The output is saved in a new relocatable element.

```
@label:FOR,options   eltname-1,eltname-2,eltname-3
```

@FREE Release the physical facilities assigned to this run under the specified filename.

```
@label:FREE,options    filename
```

@HDG Provides a means of printing a heading on successive pages of printer output along with the print file's cumulative page number and the current date.

```
@label:HDG,options    header
```

@JUMP Advances control to the specified labeled statement within the control stream.

```
@label:JUMP    label
```

@LIST Call the LIST processor to produce an edited dump of one or more absolute, relocatable, or source language elements.

```
@label:LIST,options    eltname-1,...,eltname-n
```

@LOG Places user-specified information in the master run log.

```
@label:LOG    message
```

@MAP Call the @MAP processor (the collector) to collect a specified set of relocatable elements to produce an executable program in absolute element format.

```
@label:MAP,options    eltname-1,eltname-2,eltname-3
```

@MARK Writes two hardware EOF marks on a magnetic tape file and positions the tape between the EOF marks.

```
@label:MARK,C    filename-1,filename-2,...,filename-n
```

@MODE Changes the mode settings initially set by a previous tape @ASG control statement. The file must be currently assigned to the run.

```
@label:MODE,options    filename,noise/MSA-trans/unit-trans/format
```

@MOVE Moves a magnetic tape file forwards or backwards over a specified number of EOF marks.

```
@label:MOVE,options    filename,n
```

@MSG Used to display messages on the system console.

```
@label:MSG,options    message
```

@PACK Rewrites an entire program file, removing all elements marked as deleted from the program file.

```
@label:PACK,options    filename-1,filename-2,...,filename-n
```

@PCH Punches program file elements into 80-column cards.

```
@label:PCH,options    eltname,seq-char
```

@PDP Call the procedure definition processor (PDP) to produce program file procedure entries suitable for use by the assembler, COBOL, and FORTRAN language processors.

```
@label:PDP,options    eltname-1,eltname-2
```

@PMD Calls the postmortem dump (PMD) processor to dump all or specified portions of a program that resides in main storage at program termination.

```
@label:PMD,options    eltname-1,addr/loc,length,format

@label:PMD,options    name-1,name-2,...name-n
```

@PREP Creates an entry point table from the preambles of the nondeleted elements of a program file.

```
@label:PREP    filename-1,filename-2,...,filename-n
```

@PRT Provides a listing of the master file directory items for catalogued files, the table of contents of a program file, or the text of a symbolic element. Listings of absolute or relocatable elements must be obtained using the LIST processor.

```
@label:PRT,options    name-1,name-2,...,name-n
```

@QUAL Specifies a standard filename qualifier for implied use on succeeding statements involving filenames.

```
@label:QUAL    qualifier
```

@REWIND Rewinds magnetic tape files back to the load point of the first reel.

```
@label:REWIND,options    filename-1,filename-2,...,filename-n
```

@RSPAR Used to restart a program after a program checkpoint has been taken.

```
@label:RSPAR,options    filename.element
```

@RSTRT Restore the run to its previous state after a checkpoint has been taken.

```
@label:RSTRT,priority/options    runid,acct-id,filename,ckpt-nbr,reel-nbr
```

@RUN Identifies the run to the executive and provides information needed for accounting and scheduling purposes. The run control statement must be the first statement of every run.

```
@RUN,priority/options    run-id,acct-id,project-id,run-time/deadline,pages/cards,start-time
```

@SETC Stores a value in T2 of the condition word.

```
@label:SETC,options    value/j
```

@START Permits the user to schedule independent batch runs where the run streams for these runs have been created and previously entered into the system.

```
@label:START,priority/options    name,set,run-id,acct-id,project-id,run-time/deadline,pages/cards,;
         start-time

@label:START    name,set

@label:START,priority/options    name,set,run-id,acct-id,project-id,run-time/deadline,pages/cards,;
         start-time
```

@SYM Permits the user to select a symbiont or class of symbionts to print or punch selected files.

```
@label:SYM,options    filename,,symbiont
```

@TEST Tests the value of the condition word to select the particular control statements to be executed or skipped.

```
@label:TEST    f/value/j,f/value/j
```

@USE Equates filenames so that any particular file can be referenced by more than one filename.

```
@label:USE    internal-filename,external-filename
or
@label:USE    internal-filename,internal-filename
```

@XQT Initiate the execution of a program which is in absolute element notation:

```
@label:XQT,options    eltname
```

APPENDIX B. SUMMARY OF EXECUTIVE REQUESTS

ABORT$ Unconditionally terminate all activities and the run.

```
ER      ABORT$
```

ACT$ Initiate an activity which must have been previously named through the NAME$ request.

```
L       A0,activity-name
ER      ACT$
```

ADACT$ Used to exit from an ESI activity, release specified buffers, and activate a previously named real time activity.

```
L       A1,name
L,U     A0,pktaddr
ER      ADACT$
```

APCHCA$ Specify an ASCII punch control image to a punch device routine for an alternate punch file.

```
L       A0,(image-length,buffer-addr)
ER      APCHCA$
```

APCHCN$ Specify an ASCII control function to a punch device routine for a punch file.

```
L       A0,(image-length,buffer-addr)
ER      APCHCN$
```

APNCHA$ Place a quarter-word ASCII punch image into a user-defined punch file.

```
L,U     A0,pktaddr
ER      APNCHA$
```

APRINT$ Places a quarter-word ASCII image into the system-defined print file.

```
L       A0,(PF line-spacing;
        nbr-of-words,image-addr)
ER      APRINT$
```

APRNTA$ Places a quarter-word ASCII image into a user-defined print file.

```
L,U     A0,pktaddr
ER      APRNTA$
```

APRTCA$ Specify an ASCII control function to a print device routine for an alternate print file.

```
L       A0,(image-length,buffer-addr)
ER      APRTCA$
```

APRTCN$ Specify an ASCII control function to a print device routine for a print file.

```
L       A0,(image-length,buffer-addr)
ER      APRTCN$
```

APUNCH$ Place a quarter-word ASCII image into the system-defined punch file.

```
L       A0,(nbr-of-words,image-addr)
ER      APUNCH$
```

AREAD$ Obtains an image in quarter-word ASCII from the run stream located in the RUN file.

```
L       A0,(EOF-return-addr;
        buffer-addr)
ER      AREAD$
```

AREADA$ Obtain an image in quarter-word ASCII from a user-specified file.

```
L       A0,pktaddr
ER      AREADA$
```

AWAIT$ Delay further execution of the requesting activity until all specified activities have terminated.

```
L       A0,(activity-id-mask)
ER      AWAIT$
```

BBEOF$

References the EOF address located in word 21 of the FCT and records the value in MBBI of the respective files master file directory main item.

```
L,U    A0,pktaddr
ER     BBEOF$
```

CADD$

Returns a number of buffers to the pool.

```
L,U    A0,pktaddr
ER     CADD$
```

CEND$

Notifies the executive that the requesting activity has completed contingency processing.

```
ER     CEND$
```

CGET$

Removes buffers from the pool only when the open chain method is employed.

```
ER     CGET$
```

CJOIN$

Expands or adds to a previously established pool by joining it to an additional pool area.

```
L,U    A0,pktaddr
ER     CJOIN$
```

CLIST$

Allows the user to define his own set of control statements and register them with the executive.

```
L      A0,list-designator
ER     CLIST$
```

CMD$

Initiates a communications handler dialing operation.

```
L,U    A0,lttaddr
ER     CMD$
```

CMH$

Initiates a communications handler hangup operation.

```
L,U    A0,lttaddr
ER     CMH$
```

CMI$

Initiates a communications handler input operation.

```
L,U    A0,lttaddr
ER     CMI$
```

CMO$

Initiates a communications handler output operation.

```
L,U    A0,lttaddr
ER     CMO$
```

CMS$

To initialize one or more line terminal groups.

```
L      A0,(lttcount,lttaddr)
ER     CMS$
```

CMSA$

Initiates a communications handler operation.

```
L,U    A0,lttaddr
ER     CMSA$
```

CMT$

Deactivates the input or output line terminals and performs various housekeeping functions associated with a line terminal group.

```
L,U    A0,lttaddr
ER     CMT$
```

COM$

Requests use of the onsite operator's console to display output messages and solicit operator input.

```
L,U    A0,pktaddr
ER     COM$
```

COND$

Transfers the program condition word in A0.

```
ER     COND$
```

CPOOL$

Establishes a pool of I/O buffers for communications usage.

```
L,U    A0,pktaddr
ER     CPOOL$
```

CREL$

Enables symbionts or real time user programs to release buffer pools obtained through CPOOL$.

```
LXI,U  A0,1
ER     CREL$

L      A0,poolid
ER     CREL$
```

CSF$

Submit control statements for interpretation and processing from within an executing user program rather than from the run stream.

```
LA     A0,(image-word-length;
       image-addr)
ER     CFS$
```

DACT$

Deactivate the calling activity must have been previously named through the NAME$ function.

```
ER     DACT$
```

DATE$ Places in A0 and A1 the current Fieldata date and time.

```
ER      DATE$
```

EABT$ Unconditionally terminate all activities but allow error diagnostics.

```
ER      EABT$
```

ERR$ Terminate the requesting activity and place it in ERR mode.

```
ER      ERR$
```

EXIT$ Terminates the requesting activity wherever it is within a reentrant processor or nest of reentrant processors.

```
ER      EXIT$
```

EXLNK$ Returns control to the instruction immediately following the LINK$ request.

```
ER      EXLNK$
```

FACIL$ Obtains the first nine words of the FITEM$ packet.

```
L,U     A0,pktaddr
ER      FACIL$
```

FACIT$ Obtains the first ten words of the FITEM$ packet.

```
L,U     A0,pktaddr
ER      FACIT$
```

FITEM$ Provides a method to obtain a variable amount of information on file or facility assignments.

```
L       A0,(pkt-lngth,pktaddr)
ER      FITEM$
```

FORK$ Register and initiate an asynchronous program activity.

```
LA      A0,(parameter-word)
ER      FORK$
```

IALL$ Registers a routine to handle one or more contingency types, either for the entire program or just for the requesting activity.

```
L       A0,(contingency-parameter)
ER      IALL$
```

IO$ Request an operation on an I/O file indicated and return control to the executing program without waiting for completion of the I/O operation.

```
LA      A0,pktaddr
ER      IO$
```

IOARB$ Initiates an arbitrary device I/O operation with control returned, in line, as soon as the request is either listed or the operations have been initiated.

```
L       A0,pktaddr
ER      IOARB$
```

IOAXI$ Initiates an arbitrary device operation with the referenced activity simulating an exit function, and controls the return to the program at the appropriate interrupt activity specified in the request packet.

```
L       A0,pktaddr
ER      IOAXI$
```

II$ Provides a means to define the activity which is to accept any unsolicited input directed to the program.

```
ER      II$
```

IOI$ Same as IO$ except that an interrupt activity is initiated at completion of the I/O request.

```
L,U     A0,pktaddr
ER      IOI$
```

IOW$ Identical to IO$ except control is not returned to the executing program until completion of the I/O operation.

```
L,U     A0,pktaddr
ER      IOW$
```

IOWI$ Same as IOW$ except that an interrupt activity is initiated upon completion of the I/O operation.

```
L,U     A0,pktaddr
ER      IOWI$
```

IOXI$ Requests an operation on the I/O file indicated and terminate the requesting activity. Upon completion, initiate an interrupt activity.

```
L,U     A0,pktaddr
ER      IOXI$
```

LABEL$ Enables the user to read or write any label block in the first label group on the volume except the VOL1 block.

```
L,U     A0,pktaddr
ER      LABEL$
```

LCORE$ Release unneeded main storage in the I or D banks.

```
LA,U    A0,high-req-addr-in-I-or-D-bank
ER      LCORE$
```

LINK$ Transfer control to any reentrant processor in the SYS$*LIB$ or the RLIST$ list.

```
LA,U    A0,('repname')
ER      LINK$
```

LOAD$ Load a segment of a program.

```
LA,U    A0,segname
LA,U    A1,jump-addr
ER      LOAD$
```

MCORE$ Obtain additional main storage for the I or D bank.

```
LA,U    A0,high-expansion-addr-req
ER      MCORE$
```

MCT$ To retrieve information from the master configuration table (MCT) or to read/update the program area in the MCT.

```
L       A0,packet
ER      MCT$
```

MSCON$ Enables the user to obtain information either the entire MFD or entries pertaining to a particular file.

```
L,U     A0,pktaddr
ER      MSCON$
```

NAME$ Attack a name to an activity so that the activity may be later referenced by a ACT$ or DACT$ activity.

```
LA,U    A0,18-bit-activity-name
ER      NAME$
```

NRT$ Reduce an activity or program from real time status.

```
ER      NRT$
```

OPT$ Makes available the options letters from the @XQT control statement.

```
ER      OPT$
```

PCHCA$ Specify Fieldata punch control image to a punch device routine for an alternate punch file.

```
L       A0,(image-addr,buffer-addr)
ER      PCHCA$
```

PCHCN$ Specify Fieldata control functions to a punch file routine for a punch file.

```
LA      A0,(image-length,buffer-addr)
ER      PCHCN$
```

PCT$ Obtain a copy of requested portions of this program's PCT in the user-specified buffer.

```
L       A0,(0,buffer-addr)
L       A1,(n,relative-addr)
ER      PCT$
or
L       A0,(word-count,buffer-addr)
ER      PCT$
```

PFD$ Sets delete flag in element table item for requested element.

```
L,U     A0,pktaddr
ER      PFD$
```

PFI$ Inserts an entry in the program file's element table.

```
L       A0,pktaddr
ER      PFI$
```

PFS$ Searches a program's file table of contents for a given item.

```
L,U     A0,pktaddr
ER      PFS$
```

PFUWL$ Updates the next write location in a program file.

```
L,U     A0,pktaddr
L       A1,(new-address-in-program-file)
ER      PFUWL$
```

PFWL$ Obtains the next write location in the program file.

```
L,U     A0,pktaddr
ER      PFWL$
```

PNCHA$ Transfer a Fieldata image to a user-specified punch file.

```
LA,U    A0,pktaddr
ER      PNCHA$
```

PRINT$

Transfer a Fieldata image to the system-defined print file.

```
L       A0,(PF line-spacing;
        nbr-of-words,image-addr)
ER      PRINT$
```

PRINTA$

Transfer a Fieldata image in a user-specified print alternate (PRNTA$) file. If an image is greater than 22 words, the printing of the PRNTA$ file is aborted.

```
LA,U    A0,pktaddr
ER      PRNTA$
```

PRNTA$

Places a Fieldata print image into a user-defined print file.

```
L,U     A0,pktaddr
ER      PRNTA$
```

PRTCA$

Specify Fieldata control functions to a print device routine for an alternate print file.

```
LA      A0,(image-length,buffer-addr)
ER      PRTCA$
```

PRTCN$

Specify Fieldata control functions to a print device routine for a print file.

```
LA      A0,(image-length,buffer-addr)
ER      PRTCN$
```

PSR$

Set or clear bits within processor state register (PSR) bit positions.

```
LA      A0,(parameter-word)
ER      PSR$
```

PUNCH$

Transfer a Fieldata image to the system-defined punch file.

```
LA      A0,(nbr-of-words,image-addr)
ER      PUNCH$
```

READ$

Obtain a Fieldata image from the run stream located in the READ$ file.

```
LA      A0,(EOF-jump-addr;
        buffer-addr)
ER      READ$
```

READA$

Obtain a Fieldata image from a user-specified file.

```
L       A0,pktaddr
ER      READA$
```

RLINK$

References any reentrant processor in the SYS$*LIB$ list or the RLIST$ list.

```
L       A0,('repname')
ER      RLINK$
```

RLIST$

Enters a list of user-created reentrant processors.

```
LA      A0,(nbr-entries,pktaddr)
ER      RLIST$
```

ROUTE$

Dynamically alters the primary paths of communication.

```
L       A0,(mode,lttaddr)
L,U     A1,pointer
ER      ROUTE$
```

RT$

Raise the program status to real time or change the switching priority of an activity that is already real time.

```
L,U     A0,switching-priority-level
ER      RT$
```

SETC$

Dynamically set the contents of T3 of the program condition word.

```
L,U     A0,value
ER      SETC
```

SNAP$

Obtain a snapshot dump of selected control registers and areas of main storage.

```
SA      A0,pktaddr+2
L,U     A0,pktaddr
ER      SNAP$
```

TDATE$

Places in A0 the current binary date and time.

```
ER      TDATE$
```

TFORK$

Create a timed activity. A TFORK$ request is similar to a FORK$ request except that the new activity does not begin execution for a specified amount of time.

```
L       A0,(parameter-word)
L       A1,(wait-time-in-milliseconds)
ER      TFORK$
```

TIME$

Place in A0 in binary the concurrent time in milliseconds past midnight.

```
ER      TIME$
```

TINTL$ Causes a specified tape file to be logically rewound so that a subsequent pass be made from the load point of the first reel.

```
L,U     A0,(function,pktaddr)
ER      TINTL$
```

TRED$ Displays the Fieldata message supplied and obtains in Fieldata the response.

```
L       A0,pktaddr
ER      TREAD$
```

TSWAP$ Close the current reel for a tape file and request loading of the next reel.

```
L,U     A0,(function,pktaddr)
ER      TSWAP$
```

TWAIT$ Delay execution for a specified timed wait period.

```
L,U     A1,(wait-time-in-milliseconds)
ER      TWAIT$
```

UNLCK$ Releases the high priority of an I/O interrupt activity.

```
ER      UNLCK$
```

UNLNK$ Returns control to a main program.

```
ER      UNLNK$
```

WAIT$ Delay execution until the I/O operation controlled by a specified I/O packet has been completed.

```
TP      pktaddr+3
ER      WAIT$
```

WANY$ Delay execution of an activity until an I/O operation for that activity has completed.

```
ER      WANY$
```

APPENDIX C. SYSTEM DIAGNOSTIC MESSAGES AND STATUS CODES

C.1. RUN STREAM DIAGNOSTIC MESSAGES

When a code from 1_8 to 37_8 is contained in an error message, it often points to one of the I/O problems described under type 1, ERR mode (EMODE) and I/O status codes (see C.3). Note that most of the messages issued by the FURPUR processor correspond to a specific I/O error and status code.

Some diagnostic messages refer to operator response keyins. Here are the usual meanings of the most common operator response keyins:

Keyin	Description
A	Try again with standard recovery
B	Return I/O status 12_8 to packet
D	Declare device down. I/O status 13_8
E	Treat as end of file, or error off a run
G	Treat as unrecoverable error, since I/O device positioning appears to be good. I/O status 11_8
H	Halt the operation
I	Initiate a locked out or suspended symbiont
L	Lock out a symbiont
N	The reply is "no"
Q	Reenter a symbiont file in its appropriate queue
R	Reprint or repunch a symbiont file
S	Suspend a symbiont
T	Terminate a symbiont
X	Abort a symbiont or abort a run
Y	The reply is "yes"

One of the three abreviations, SI (symbolic input), RO (relocatable or absolute output), or SO (symbolic output), is frequently used to identify the element named in the corresponding specifications subfield of a processor control statement, such as @ASM, @COB, @FOR, or @MAP. For processors such as @ELT which have no RO subfield, only SI and SO are meaningful.

The run stream diagnostic messages are:

ADD ELEMENT NOT FOUND

ADD FILE NOT ASSIGNED OR CATALOGUED

ADD LOOP, FILE OR ELEMENT PREVIOUSLY ADDED IN THIS NEST

AWAIT/DEACT AMBIGUITY

The executive determined that all activities of the run were either in an AWAIT state (ER AWAIT$) or a DEACT state (ER DACT$) and could not nor would not be activated by the run.

BAD INPUT SEQUENCE

The input control message was rejected because it conflicts with the current status of the run.

x BADLY CODED FIELD

Fill from previous field requested and filled field is illegal or ambiguous.

BAD RUN STATEMENT

The @RUN control statement is improperly formatted. This message is displayed only for demand runs; for batch runs the device is placed in the run search mode. If the statement was submitted from a demand terminal, it may be immediately retyped in the proper format.

CANNOT SYM PU$ WHILE IN USER FILE

Caused by @SYM PUNCH$,,CP control statement when the current PUNCH$ file has been @BRKPT'd to a user file.

CANNOT SYM TEMPORARY FILE

x CYCLE SPECIFICATION IGNORED

RO or SO cycle specification is meaningless and is ignored. Does not cause error return.

DATA IGNORED – IN CONTROL MODE

Data statements were encountered when the coarse scheduler was attempting to read control statements; that is, a program or processor was not in control of the run at the time these statements were encountered.

DBANK CANNOT BE LOADED WITH NEGATIVE BD
DBANK RELATIVE STARTING ADDRESS = 0sss000, DBANK SIZE = 0nnn000,
LAST ADDRESS OF USER CORE = 0xxx777.

The program cannot be loaded without causing Bd to become negative. This is the result of using a SETMIN directive to specify a minimum starting address for the DBANK. See *UNIVAC 1108 Processor and Storage Programmer Reference Manual, UP-4053* (current version)

DYNAMIC PCT EXPANSION FOR RT PROG NOT ALLOWED

A PCT expansion attempt for a real time program could not be done. If the program expects PCT expansion it should use a @RUN control statement option to initialize PCT size so expansion is not attempted.

ELEMENT ADD FROM TAPE NOT ALLOWED

ELEMENT UNOBTAINABLE xx

The element specified on the @START control statement cannot be found. xx is a program file search code (see C.5.2).

@END IGNORED – IN CONTROL MODE

An @END control statement was encountered when the coarse scheduler was attempting to read control statements; that is, the DATA OR ELT,D processor was not in control of the run at the time this statement was encountered.

@EOF IGNORED – IN CONTROL MODE

An @EOF control statement was encountered when the coarse scheduler was attempting to read control statements; that is, a program or processor was not in control of the run at the time this statement was encountered.

EQUIPMENT TYPE ERROR, ADD FILE

ERROR – DYNAMIC DUMPS NOT CLOSED

I/O error encountered when writing system diagnostic file (DIAG$)

ESTIMATED RUN TIME EXCEEDED

The run either exceeded the system standard or the time specified on the @RUN control statement and the T option was set.

FAC REJECTED xxxxxxxxxxxx

This message appears for a run that aborted due to a statement that cannot be honored by the system. See C.2 for an explanation of the 12 digit (x . . . x) octal code.

FAC WARNING xxxxxxxxxxxx

This message is a warning that the statement could cause a problem. See C.2 for explanation of 12 digit (x . . . x) octal code.

FILE ALREADY IN USE

@BRKPT control statement issued for a file currently being used as a symbiont file (read alternate, print alternate, and so forth).

x FILE CAN NOT BE READ

Input file is in read inhibited mode due to absence of read key, write-only mode set for file, or Y option used on the file assignment.

FILE ERROR

The file requested on a @XQT or processor control statement could not be assigned. If the run is not demand, it is terminated.

x FILE NOT FOUND – STATUS: n

File x is neither assigned to the run nor catalogued. n is the status returned when an attempt was made to assign file x.

FILE UNOBTAINABLE xxxxxxxxxxxx

The file specified on the @START control statement cannot be accessed by the executive. xxxxxxxxxxxx is a 12-digit octal status code returned when the file cannot be assigned (see C.2).

FIRST FILE NAME IS IN ERROR

First file name was not given for @BRKPT or @SYM control statement, or a @BRKPT control statement was for an inactive alternate file.

nn ILLEGAL CHARACTER x

The coarse scheduler encountered an illegal character x at column nn of the above control statement.

ILLEGAL CONTINUATION

Continuation of the above control statement is not allowed or the next control statement has the control character (@) in column 1. The control statement is not honored and the run is terminated (if it is not a demand run).

x ILLEGAL DEVICE

Output file is not FASTRAND format or input file is neither tape nor FASTRAND format.

x ILLEGAL FIELD

Field is ambiguous with option given (for example, I option specified and source output field coded).

nn ILLEGAL OPTION x

An illegal option x was encountered at column nn of the above control statement. The control statement is not honored and the run is terminated (if it is not a demand run).

IMPROPER RUN STREAM IN FILE

The first image in the file or element specified on the @START control statement is an invalid @RUN control statement.

IMPROPER SYM CONTROL MSG

The operator has submitted a symbiont control message which contains improper information.

INTERVENING STATEMENTS SKIPPED

A conditional statement has been encountered and has caused one or more control statements to be bypassed.

INVALID SYMBIONT NAME

@SYM file is directed to an illegal device.

I/O ERROR ENCOUNTERED

The executive returns this message after receiving an error code while trying to read the file specified on the @START control statement.

I/O ERROR IN TERMINATION – PMD NOT INITIALIZED

I/O error encountered when writing system diagnostic file (DIAG$)

I/U OPTION CONFLICT

Both I and U options given on processor control statement – ambiguous options.

LABEL FORMAT ERROR

User tried to illegally access a labeled tape or hardware error occurred when trying to validate a tape/disc pack label. Also, operator responded E to a request to mount a disc pack or tape.

L IS LEGAL FOR TAPE ONLY

The L option was specified on the @BRKPT control statement and file does not reside on tape.

MASS STORAGE OVERFLOW

Mass storage request cannot be satisfied because mass storage is not currently available.

MAX CARDS

The estimated card output has been exceeded and a system generation parameter or the C option on the @RUN control statement specified that the run be aborted if this occurred.

nn MAX NUMBER OF CHARACTERS EXCEEDED

The character at column nn of the above control statement is not a field/subfield terminator and the maximum number of characters for this subfield has been reached. The control statement is not honored and the run is terminated (if it is not a demand run).

nn MAX NUMBER OF FIELDS OR SUBFIELDS EXCEEDED

The character at column nn of the above control statement is the field terminator and no more fields are allowed for the control statement, or the character is a subfield terminator and no more subfields are permitted for that particular field. The control statement is not honored and the run is terminated (if it is not a demand run).

MAX PAGES

The estimated page output has been exceeded and a system generation parameter or the P option on the @RUN control statement specified that the run be aborted if this occurred.

MAX TIME

The estimated running time has been exceeded and a system generation parameter or the T option on the @RUN control statement specified that the run be aborted if this occurred.

MESSAGE TOO LONG

The operator has attempted to transmit a message which exceeds the input limit of 72 characters.

NO FILE NAME

NO FILE SPECIFIED

File name is not specified on a @START control statement.

NO RUN ACTIVE

Applies only to demand processing – message appears on demand terminal if statements are entered before the @RUN control statement.

NO SPACE FOR MAJOR SAVE ON ABORT$ CONTINGENCY

No space available in PCT for register save when processing an ABORT$ contingency. Run is terminated.

x NOT A PROGRAM FILE

x = SI: RO:, or SO:; file specified may not be used as a program file.

NUMBER OF ADDS IN NEST EXCEED MAXIMUM ALLOWED

OPERATION IS ILLEGAL FOR DEMAND

@BRKPT PRINT$ or @BRKPT PRINT$/PRINT$ control statement from a demand terminal before @BRKPT PRINT$/file

OPERATOR REMOVED RUN

The operator removed the run from the backlog by the RM keyin.

OPERATOR TERMINATED RUN BY AN E-KEYIN

The operator replied with an E to @MSG control statement with a W option: batch runs are terminated in this case.

OPERATOR TERMINATED RUN BY AN X-KEYIN

Similar to above diagnostic except response was an X rather than an E keyin.

xOUTPUT FILE IS TAPE

Output file should be FASTRAND format and is tape instead.

*** PARITY ERROR ***

A parity error has been detected in at least one character of the input image. The entire image is discarded. (For teletypwriter only)

PCT EXPANDED BEYOND SYSTEM LIMITS

The number of main storage blocks required for expansion of this run's PCT exceeds the systems generation parameter PCTMAX. When a run aborts with this message, a postmortem dump of the PCT (obtained using @PMD,P) may show one of the following to be the cause:

- Excessive number of granule tables (change track granularity to position granularity)
- Excessive number of activities (check for ER FORK$ loop)
- Excessive number of files assigned (check for ER CSF$ loop)

PCT FULL, CANNOT ASSIGN ADD FILE

PCT/PROGRAM SIZE EXCEEDS USER CORE

Either an internal main storage bank has been downed so the program does not fit or a real time program has started and this program is too large to fit the available main storage.

PMD NOT ALLOWED

Postmortem dump is not allowed for a system processor (called from the file SYS$*LIB$) unless a Y option appeared on the @RUN control statement. If an N option appeared on the @RUN control statement, no postmortem dumps of any programs are allowed.

PROGRAM NOT FOUND

The requested program or processor is not in the given file, LIB$, or TPF$ (depending on the statement). If the run is not demand, it is terminated.

PROGRAM TOO LARGE PROGRAM SIZE nnn BLKS.
CORE SIZE xxx BLOCKS.

The program is too large to fit in the space available to user programs. The program requires nnn main storage blocks and user main storage consists of xxx main storage blocks. A main storage block is 512_{10} (1000_8) words and the sizes nnn and xxx are octal numbers, thus giving the sizes in octal 1000's.

PUNCH FILE CANNOT BE PUNCHED: NO PUNCH DEVICES CONFIGURED

x READ ONLY OUTPUT FILE

Output file is in write inhibited mode, due to absence of write key, read-only mode set for file, or Y option used on the file assignments.

REAL TIME PROGRAM ATTEMPTED PCT EXPANSION

A PCT expansion attempt for a real time program could not be done. If the program expects PCT expansion it should use a @RUN control statement option to initialize PCT size so expansion is not attempted.

nn REQUIRED FIELD OR SUBFIELD MISSING

A field or subfield which is required on the above control statement has not been specified. The omission was detected when the field/subfield terminator or the end of the control statement was encountered at column nn. The control statement is not honored and the run is terminated (if it is not a demand run).

RUN KILLED VIA AN E-KEYIN

The operator typed an unsolicited E keyin for this run. The message is printed only for batch runs. An unsolicited E keyin for a demand run simply terminates the currently executing task.

RUN KILLED VIA AN X-KEYIN

Same as above except that the operator used the X keyin.

RUNSTREAM ANALYSIS TERMINATED

The run has been terminated because of an error condition and the remaining control statements are not processed.

SECOND FILE NAME IS IN ERROR

Caused by second file name on @BRKPT control statement not currently assigned to the user.

SECOND FILENAME NOT ASSIGNED, PUNCH$ NOT BREAKPOINTED

SECOND NAME IS ILLEGAL

Second file cannot be given on a @BRKPT of a read alternate, print alternate, or punch alternate file.

SI: CYCLE NON-EXISTENT OR IN ERROR

Requested cycle of specified element does not exist or cycle field has improper format.

SI: ELEMENT NOT FOUND

Element name given cannot be found as a symbolic element in the specified program file.

SI: IMPROPER LABEL BLOCK

Source input file is tape, and tape is not positioned at the label block for requested element, probably because a @FIND has not been done.

SI: MISSING FIELD

A field of required information (for example, element name) was not given.

SI: TAPE I/O ERROR – STATUS: n

SIR EDIT ERR c r e

Line correction diagnostics produced by SIR in the edit mode.

where:

c – Indicates the cause of the error. A list of possible causes is shown below.

r – First four words of the range correction statement under whose control the error occurred.

e – Specifies the change correction statement that caused the error.

- SEPARATOR – The separator used in the change correction statement is invalid or nonexistent.
- COLUMN – The column number specified on a format 1 or 2 change correction statement is out of range, or that C > D for a format 2 change correction statement.
- NO FIND – The characters given in the old data parameter of a format 3 or 4 change correction statement could not be found in the line being corrected.

NOTE: Whenever one of the above errors occurs, the change correction statement is ignored and the line remains unchanged.

- ASCII MODE – Indicates that symbolic input or output is in ASCII code, or that the user requested ASCII code. Since SIR cannot correct ASCII code, all range and change correction statements are ignored.
- CARD COUNT <– Not enough change correction statement were provided. Those lines for which no change correction statement was provided remain unchanged.
- CARD COUNT >– Too many change correction statements were provided. The excess change correction statements were ignored.

SYM FILE IS NOT CATALOGUED OR CANNOT BE FOUND IN THE DIRECTORY

TAB STATEMENT ERROR

If a @TABSET control statement is in error, the terminal operator is notified by the message and any previous tab definitions are ineffective.

TAPE IMAGE LIST*REREAD

Applies only to demand terminals – indicates that images were lost while inputting images in form II paper tape mode.

TAPE LABEL FORMAT ERROR

Additional diagnostic generated after LABEL FORMAT ERROR to indicate that the file was a tape file.

TIME OUT

No activity has occurred on the line for a predefined interval. If another time interval elapses without activity, the terminal is terminated and the message TIME-OUT TERMINATION is displayed at the terminal.

UNRECOVERABLE I/O ERROR WHEN READING FILE filename

The coarse scheduler encountered an unrecoverable I/O error when searching file filename for a program or processor. If the run is not demand, it is terminated.

USER DID AN ER ABORT$

USER DID AN ER EABT$

WAIT

Applies only to demand terminals – indicates that the system is not ready for further input.

WARNING IMPROPER OPTION

Caused by @SYM,C control statement with printer symbiont name (warning only).

C.2. FACILITY REQUEST STATUS CODES

If a facilities request made by one of the facilities control statements (@ASG, @MODE, @CAT, @FREE, and @USE) is found to be in error, a status word is generated in which the various bits set define the error. For incorrect facilities control statements submitted in the run stream, the status word is given as part of the FAC REJECTED . . . or FAC WARNING . . . message. For control statements submitted by a CSF$ request, the status word is returned in register A0. The meanings of the possible bit settings are given in Table C–1.

Bit Set	Description
35*	Request not accepted; check other bits for reason.
34*	Field error in control statement other than syntax. Option conflict (MHL, OE, or IB) or noise constant specification error.
33	File is already assigned for @ASG or @CAT control statement specified, already freed for the @FREE control statement specified, or not assigned for the @MODE control statement specified. This setting is fatal for @CAT and @MODE control statements.
32*	The file was previously catalogued.
31*	Equipment type specified on @ASG control statement is not compatible with catalogued type or file specified on @MODE control statement is not magnetic tape.
30	Not used.
29	That portion of the filename used as the internal name for I/O packets is not unique.
28	Not used.
27*†	Incorrect read key for catalogued file.
26*†	Incorrect write key for catalogued file.
25	Write key that exists in the master file directory is not specified in the @ASG control statement (file assigned in the read-only mode).

Table C–1. Facility Status Bits
(Part 1 of 2)

Bit Set	Description
24	Read key that exists in the master file directory is not specified in the @ASG control statement (file assigned in the write-only mode).
23*	Read key specified in the @ASG control statement; none exists in the master file directory.
22*	Write key specified in the @ASG control statement; none exists in the master file directory.
21*	A option was specified in the @ASG control statement and the filename cannot be found in the master file directory.
20*	Invalid reel number specified in the @ASG control statement for a catalogued tape file.
19*	Mass storage file has been rolled out.
18*	Request on wait status for facilities. For a tape file, this usually means a tape unit is not currently available. For a drum file, this usually is caused by an exclusive use conflict with another concurrent run.
17*	Option conflict for catalogued files; both O and K or CUPRW were specified, which are options for new files.
16	Not used.
15	Find was made on a catalogued file request and the file was already asigned to another run.
14	Not used.
13*	Project-id incorrect for catalogued private file
12	Not used.
11	Read-only file catalogued with an R option.
10	Write-only file catalogued with a W option.
9	Equipment requested is down.
8*	File specified in a @ASG control statement is disabled because the links pertinent to its master file directory items have been destroyed.
7	File specified in a @ASG control statement has been disabled because the file was assigned write-enabled during a file recovery.
6*	File specified in a @ASG control statement has been disabled because the file has been rolled out and the backup copy is unrecoverable.
5-0	Not used.

*Request was rejected. If facility request was submitted by the run stream, this results in a FAC REJECTED message and termination if a batch run (results in only a FAC REJECTED message for demand runs). For dynamic requests through a CSF$ request, bit 35 is set in the status word returned in register A0.

†If the statement was submitted by a CSF$ request, the run is aborted and no status word is returned in register A0.

Table C–1. Facility Status Bits
(Part 2 of 2)

C.3. ERR MODE (EMODE) AND I/O STATUS CODES

This set of error codes is categorized as being under contingency type 12_8.

Most of these codes relate to errors users make when setting up executive requests (ER's). The most common user errors are improperly set up, improperly referenced, and inadvertently overwritten packets.

The ER's are categorized as follows:

Type	Mnemonic	Definition
1_8	I/O	All I/O ER's (IO$, IOWS, etc.)
2_8	SYMB	All symbiont ER's (READ$, PRINT$, etc.)
3_8	ERR$	ER$ ER
4_8	ER	Miscellaneous ER's
5_8	CONS	COM$ ER
6_8	COM2	Communication ER's (CPOOL$, CMS$, etc.)
7_8	COMM	
10_8	REP	Reentrant processor ER's (LINK$, RLIST$, etc.)

Table C–2 is the full set of defined ERR mode codes. The following type 1 (I/O) codes 0_8 through 17_8 and 40_8 are included for the sake of completeness, even though they represent status conditions that are not necessarily errors, and do not directly force a run into ERR mode. Many of these codes cause the system processors to take an error exit, after passing on the code to the user in an I/O error diagnostic message.

Type	Code Octal	Description
I/O (1)	0	The request has been completed normally. If data transfer is involved, the count is given in H2 of word 4.
	1	End-of-file block detected on magnetic tape. ■ Answer of E was given to an I/O error message. ■ End-of-file block was detected on magnetic tape. ■ Block read drum function was truncated by encountering an end-of-block word. ■ Block search read function was truncated by encountering an end-of-block word before the specified number of words were transferred.
	2	End-of-tape mark encountered on magnetic tape on a read backward from load point or on a write. No transfer takes place for the read backward. The write is done in the normal manner. Subsequent writes are performed in the same fashion and, barring other problems, results in returning the same status code.
	3	No find was made on a mass storage device search. The search was terminated by an end-of-block, end-of-track, end-of-position, or expiration of sufficient time to pass over the entire area of concern depending upon the physical device and type of search.
	4	A nonintegral block was read from magnetic tape. The number of data characters accepted from the last word is indicated by S3 of word 3 of the packet.

Table C–2. ERR Mode (EMODE) and I/O Status Codes
(Part 1 of 10)

Type	Code Octal	Description
	5	An attempt was made to initiate a mass storage search or read from an area which is wholly or partially unassigned. If the starting address is legal the read is truncated as reflected by the word count in the substatus field.
	10	The area of the FASTRAND mass storage file being unlocked by this write or unlock request timed out in the locking list or a subsequent request by the same activity had a packet format error detected between the time of submitting the request and the time of servicing. Other requests by other activities for the area may have been honored in the interim. If the function is write, the transfer is not performed.
	11	A nonrecoverable error has occurred. The suppress recovery mode is set for magnetic tape or an answer of G was given to an error message. If the suppress recovery mode is set, the EI status code is stored in A0 of the interrupt activity control register set. All suppress recovery operations come back with this status.
	12	A read, or write error on magnetic tape has resulted in loss of position on the unit. This code is returned for all outstanding requests at the time the answer of B was entered in response to the I/O error message. Any subsequent request is honored but the lost position is maintained and no further program checkpoints are valid.
	13	The peripheral unit was declared down either by an unsolicited operator keyin or in response to an error message typed after the normal recovery failed to resolve a malfunction.
	20	Some form of write or a function causing area release was attempted on a file assigned in the read-only mode, or a form of read was attempted on a file in the write-only mode.
	21	An attempt was made to reference a filename for which no assignment has been made.
	22	An attempt was made to reference beyond the maximum for mass storage file or to expand a word addressable format file when no space is available.
	23	The packet address specified in the A0 register is not within the program limits or defines a packet split between the instruction and data banks of the program.
	24	The function code is not defined for the assigned equipment type. This code also covers noncompatible fields on a set mode request.
	25	The I/O access word refers to a buffer which is wholly or partially outside of the program area or split between the instruction and data bank of the program. For GW$, SCR$, and SCRB$ functions, this error code is given if the number of access words is 0 or more than 50 or if the total word count is more than 65535.
	27	An I/O request was made with the status word of the request packet set negative indicating a possible program loop. Also, this error occurs when the packet (access word or file address) is found to have been changed while in the process of executing.
	31	A magnetic tape operation was issued with user recovery specified and an interrupt activity was not specified (that is, entrance was not made via IOI$, IOXI$ or IOWI$).
	32	An I/O request could not be performed immediately, and when the packet was examined on a subsequent attempt to perform the request, the filename was found to be attached to a different subsystem or unit. This means the user program changed the packet between attempts.
	33	A FASTRAND-formatted or word-addressable I/O request may cause the PCT to expand past its maximum. The I/O request is not initiated.

Table C–2. ERR Mode (EMODE) and I/O Status Codes (Part 2 of 10)

Type	Code Octal	Description
	34	An absolute read or write was attempted by user program.
	35	Errors on read and lock, and unlock requests:
		(a) A second read and lock (RDL$) request by an activity for a particular area.
		(b) An unlock (UNL$) request for an area that the activity had not previously locked.
	36	Errors on WAIT$ requests:
		(a) The outstanding I/O request count for a program goes to zero and an activity is still waiting for completion of an I/O packet.
		(b) WAIT$ request was not immediately preceded by a Test Positive instruction or the Test Positive instruction had a nonzero h or i field.
		(c) A WAIT$ or WANY$ request was made without a previous outstanding I/O request for the program. (The executive tests for the case of I/O completion between the time the ER is initiated and the time it is processed without error notification.)
	40	The request is either in the process of being executed or is listed on the request queue for the particular channel.
SYMB (02)	2	Second abnormal return from READ$
	3	I/O error (READ$ or READA$)
	4	Image length error
	5	@ADD error, run stream
	6	READ$ access word failure
	7	Improper SDF control image, READ$ request, or READA$ request.
	10	Element ADD from tape
	11	Nested level exceeds maximum (ADD)
	12	File not assigned or catalogued
	13	Element not found in file
	14	Nested ADD loop
	15	ADD file equipment type error
	16	Cannot assign ADD file because PCT is at maximum size
	20	Alternate file not assigned for demand or real time run
	21	Cannot assign punch file
	22	Type of call does not match type of file

Table C–2. ERR Mode (EMODE) and I/O Status Codes (Part 3 of 10)

Type	Code Octal	Description
	23	Alternate packet out of limits
	24	Read alternate file not assigned
	25	Error on first read from read alternate
	26	Alternate file not FASTRAND or tape
	27	Maximum number of active alternate files exceeded
	30	Maximum number of breakpoints for print or punch exceeded.
	34	Exceeded print alternate file
	35	Exceeded punch alternate file
	36	Exceeded punch file
	37	Exceeded print file
	40	Buffer out of limits
	41	MAX pages
	42	MAX cards
	43	Illegal syntax in control image
	44	Maximum length exceeded on control image
ER (04)	1	ER index out of range or ER for executive only
	2	Bad packet limits on ER (abnormal return from access word check)
	3	ER index within range, but not in use
	4	Error encountered on AWAIT$ request
	5	Bad activity number (id) specified on FORK$ request. Either out of range or already in use
	6	Account number does not permit requested real time priority (RT$, FORK$)
	10	FACIL$/FACIT$/FITEM$ packet/failed access word check
	11	FACIL$/FACIT/FITEM I/O error or PCT name section in error
	20	Bad BBEOF$ packet or invalid file control table address
	21	File not catalogued or file not mass storage FASTRAND format
	31	Illegal creation of real time activity via FORK$ request
	32	An activity marked as "named" cannot be found in the activity name table on a NAME$, ACT$, or DACT$ request
	33	Illegal II$ request

Table C–2. ERR Mode (EMODE) and I/O Status Codes (Part 4 of 10)

Type	Code Octal	Description
	37	Filename not assigned for tape swap
	40	Syntax error on CSF$ image
	41	CSF$ image length is greater than 40 words
	42	Illegal command for CSF$
	43	Image is outside user's program limits
	44	Log entries for this run exceed MAX allowed by CSF
	46	Unsuccessful request for PCT buffer for use in loading a relocatable segment
	47	Invalid input parameter to LOAD$ (A1,H1 or A2,H1 nonzero)
	50	I/O error encountered when loading segment
	51	Request to load an undefined segment
	52	Invalid information in segment load table or segment out of limits.
	53	Invalid MCORE$ request
	54	Attempt to release core not currently held via LCORE$
	55	MCORE$ request for core not available
	57	Attempt to release contingency or reentry address via LCORE$ request
	60	Incorrect recognition key for DLOC$ or DIW$
	61	Bad packet on ER SNAP$
CONSOLE (05)	0	Packet not within limits
	1	Output buffer not within limits
	2	Expected input count exceeds 50 characters
	3	Input buffer not within limits
COMM (06)	1	A CMD$ request has specified an invalid dial digit count or has specified a dial buffer address which is beyond the limits of the program.
	2	A CMS$ request specifying ESI completion activities with entry points in the I bank when attached to a reentrant processor.
	3	A CMO$ request has specified: (1) an invalid pool-id for POOL mode output; (2) an invalid buffer address; or (3) a buffer which has been released
	4	A CMO$ request for pool mode output has specified an invalid buffer character count
	6	A CGET$ or a CADD$ request has been made with an invalid buffer pool-id or has specified a closed-chain pool of buffers

Table C–2. ERR Mode (EMODE) and I/O Status Codes (Part 5 of 10)

Type	Code Octal	Description
	11	A CMD$ request utilizing automatic dialing is only supported for the CTMC
	12	A CMD$ request has been made with the dialing operation currently in progress
	13	A CMD$ request has been made with an invalid BCD dial digit
	16	The requestor's PCT is not large enough to contain the necessary processing information generated from servicing the CMD$ request.
	20	A ROUTE$ request has been made from the wrong type of coding. Use of this ER is not permitted for demand, deadline batch, or batch activities.
	21	A ROUTE$ request has specified a LTT address in H2 of A0 which is beyond the limits of the program.
	22	A ROUTE$ request has specified a filename which does not appear in the facility assignment section of the user's PCT.
	23	A CGET$ request has specified a buffer count which is greater than the number of buffers available.
	24	A ROUTE$ request has specified an invalid mode parameter in R1 of A0.
	25	A CADD$ request has specified an invalid buffer address or the buffer count exceeds the number of buffers in the chain.
	26	A ROUTE$ request has been made for output, but output is not defined for the alternate route.
	27	A ROUTE$ request has been made for input, but input is not defined for the alternate route.
	30	A ROUTE$ reference has requested an alternate path which has already been assigned.
	31	A ROUTE$ request has been made with a zero alternate drum address in the LTG's site table.
	32	A zero absolute drum address was computed for the alternate site table of a ROUTE$ REQUEST'
	33	A ROUTE$ request has specified a line terminal group which has not been initialized via a CMS$ request.
	34	A ROUTE$ request has specified an invalid pointer when requesting an alternate path. The pointer is greater than the number of alternates specified at systems generation time.
	35	A ROUTE$ request has been made for an output alternate, but that path has been downed or there is no path available.
	36	A ROUTE$ request has been made for an input alternate, but that path has been downed or there is no path available.
	40	A CMS$ request has been made with output specified for a terminal which was not set up for output at system generation time.
	41	A CMS$ request has been made with input specified for a terminal which was not set up for input at system generation time.

Table C–2. ERR Mode (EMODE) and I/O Status Codes (Part 6 of 10)

Type	Code Octal	Description
COMM (07)	1	A CMS$ request has been made from the wrong type of coding. Use of this ER is not permitted for demand, deadline batch, or batch activities.
	2	A CMS$ request has specified a LTT address in H2 of A0 which is beyond the limits of the program.
	3	The initialization request specified a LT table which is currently in use or has been initialized, but has never been terminated or a terminal which is inoperable at this time.
	4	A CMO$, CMI$, or a CMSA$ request has been made with the CPU path being downed by the system.
	5	An invalid value was specified for the input completion activity usage code within the requesting packet for a CMS$ request.
	6	The filename for an initialization request does not appear in the facility assignment section of the user's PCT.
	7	The end-of-input activity code in the packet for an initialization request is invalid.
	10	A CMI$, CMO$, or CMSA$ request has been made with the LTG having been deactivated due to an ESI contingency error or no ESI contingency specified for an ESI contingency.
	11	The control group address specified in a packet to be used for pool mode operation is invalid.
	12	Pool mode operation has been specified, but no buffers presently exist in the pool specified as the one to be used.
	13	The input completion activity address specified within the packet for a CMS$ request is beyond the limits of the program.
	14	The output usage code specified within the packet for an initialization request is invalid.
	15	A CMS$ request was made specifying output pool mode with an invalid pool-id or with closed mode.
	16	A CMT$ request has specified a LT group which has been terminated previously or has never been initialized.
	17	A CMT$ request has been made from the wrong type of coding. Use of this ER is not permitted for demand, deadline batch, or batch activities.
	20	A CMT$ request has specified a LTT address of H2 or A0 which is beyond the limits of the program.
	21	A request for an input operation has specified a filename for which input is not permissible. This indicates no input facility for the LT group was defined at systems generation time.
	22	An invalid value was specified for the input completion usage code within the requesting packet for CMI$ request.
	23	The number of characters specified for a single mode input or output operation exceeds the maximum permissible value established at systems generation time.

Table C–2. ERR Mode (EMODE) and I/O Status Codes (Part 7 of 10)

Type	Code Octal	Description
	24	CMI$, CMO$, or CMSA$ request has specified a LTT address in H2 of A0 which is beyond the limits of the program.
	25	A nonmultiple of 6 was specified as the character count for an input operation on the WTS subsystem.
	26	An input operation has specified a character count and buffer starting address such that a portion of the input buffer exists beyond the limits of the program.
	27	An input or output operation has specified pool mode operation for a filename associated with equipment which uses the ISI method of buffering. Buffer swapping for ISI devices is not permitted because of the design of the hardware which operates on data by blocks rather than as a continuous input stream.
	30	The filename for a CMI$, CMO$, or CMSA$ request does not appear in the facility assignment section of the user's PCT.
	31	A request for output operation has specified a filename for which output is not permissible. This indicates that no output facility for the LT group was defined at systems generation time.
	32	An invalid value was specified for the output completion usage code within the requesting packet for a CMO$ request.
	33	A CMO$ request specifying pool mode operation has been made at the same time from more than one activity using the same LTT.
	34	An output operation has specified a character count and a buffer starting address such that a portion of the output buffer exists beyond the limits of the program.
	35	An initialization request has specified a filename associated with equipment which is not handled by the communications complex.
	36	A CPOOL$ request specified an illegal pool area because: (1) the pool size equals zero words; (2) pool area is split between user program's D and I banks; or (3) pool area is beyond the limits of user's program.
	37	The main storage area specified by a CPOOL$ request is located in the program's I bank. Pool buffering using the I bank is not permitted since a program should be positioned in main storage to allow the hardware memory overlap feature for ESI data transfers to reduce the number of main storage accesses from three to two for each transfer.
	40	A CPOOL$ request has specified an illegal buffer size because: (1) character count equals zero; or (2) character count exceeds system's specified maximum.
	41	A CPOOL$ request has specified a pool area which overlaps that of a previous request which has yet to be released.
	42	A CMI$, CMO$, CMSA$, CPOOL$, or CREL$ request to the communications handler has been executed from the wrong type of coding. Use for demand, deadline batch, or batch activities.
	43	The single mode of buffering has specified an input buffer which is in the I bank. Communications buffers are not permitted in the I bank because real time programs are positioned in main storage to allow the D-bank to utilize the memory overlap feature for each character transfer.

Table C–2. ERR Mode (EMODE) and I/O Status Codes
(Part 8 of 10)

Type	Code Octal	Description
	44	A CMS$ request for pool mode operations and the buffer mode conflicts with the subsystem mode specified on the SGR configuration card.
	45	The address within the packet specified as the output completion activity was not valid for a CMS$ request.
	46	A CMI$ or CMO$ request has been made requesting use of an LT group by the communications handler before that LT group has been initialized via a CMS$ request.
	47	The requestor's PCT is not large enough to contain the necessary processing information generated from servicing the initialization request.
	50	A CMI$ request for single buffer mode has specified an input buffer area which conflicts with the main storage module locations of the ESI access control words. Communications buffers are not permitted in the same main storage module as the ESI ACW's because of the extended time required for each character transfer when the memory overlap capability is not utilized.
	51	A CMO$ request for single buffer mode has specified an output buffer area which conflicts with the main storage module locations fo the ESI ACW's. Communications buffers are not permitted in the same main storage module as the ESI ACW's because of the extended time required for each character transfer when the memory overlap feature is not utilized.
	52	The main storage area specified by a CPOOL$ request is located in the same main storage module as the ESI ACW's. Communications buffers are not permitted in the same main storage module as the ESI ACW's because of the extended time required for each character transfer when the memory overlap capability is not utilized.
	53	A CMS$ request has specified an LTT which exists in the real time program's I bank. Real time programs are expected to utilize the hardware feature of main storage overlap by being divided into appropriate I and D bank portions.
	54	A CMD$ or CMH$ request has been made from the wrong type of coding. Use of this ER is not permitted for demand, deadline batch, or batch activities.
	55	The dial usage code specified within the packet for a CMD$ request is invalid.
	56	A CMD$ or CMH$ request has specified an LTG which has been terminated previously or has never been initialized
	57	The address within the packet specified as the dial completion activity address was not valid for a CMD$ request.
	60	ESI contingency error for contingency types 1 through 5 from the ESI activity.
	61	ESI contingency error for ACT$ or ADACT$ requests from the ESI completion activity.
	62	ESI contingency error for CADD$ or ADACT$ requests from the ESI completion activity.
	63	ESI contingency error for illegal ER from the ESI completion activity.
	64	An ESI completion activity for an ESI contingency activity has exceeded the maximum amount of execution time set at systems generation time.
	65	A CMT$ request has specified a filename which does not appear in the facility assignment section of the user's PCT.

Table C–2. ERR Mode (EMODE) and I/O Status Codes (Part 9 of 10)

Type	Code Octal	Description
	66	A CMD$ or CMR$ request has specified a filename which does not appear in the facility assignment section of the user's PCT.
	67	A CMD$ or CMH$ request has specified a filename for which no dialing or hang-up capability exists.
	70	A CREL$ request has specified an invalid pool name.
	71	A CREL$ request has specified an invalid release code.
	72	A CREL$ request has specified a buffer pool which is not assigned to the requestor.
	73	A CREL$ request has specified a pool to be released which is being used for input.
	74	Neither input nor output mode has been specified in the packet for an initialization request.
	75	A CMD$ request has been made with no CPU path available or a path marked as down by the system.
	76	A CMS$ request has been made with no output path available or the path has been downed by the system.
	77	A CMS$ request has been made with no input CPU path available or the path has been downed by the system.
REP (10)	1	RLIST$ packet not within program limits.
	2	LINK$ or RLIST$ request and REP's entry point is zero.
	3	RLIST$ request and either the file is not assigned or not on mass storage.
	4	RLIST$ entry name not found.
	5	LINK$, RLINK$, or RLIST$ request and REP's I bank exceeds program's starting D bank address.
	7	Attempt to link to multiple REPs simultaneously.
	11	RLIST$ request to remove previous REP list with REP's active.
	12	LINK$ or RLINK$ request and specified name not found by system's search.
	14	EXLNK$ or UNLNK$ request not from linked routine.
	15	Number of RLIST$ REP names exceeds system's maximum.
	16	A LINK$, RLINK$, or RLIST$ request and no PCT space available.
	17	A LINK$, RLINK$, or EXLNK$ request and system detected an I/O error in loading a REP. Or an I/O error reloading the REP after its storage had been released for timesharing.
	20	The main program plus the REP's main storage requirements exceed total user main storage.
	21	Attempt to change REP size via MCORE$ or LCORE$ requests.
	22	Same as 20 except activity has been returned to main I bank, PSR, and storage limits.

Table C–2. ERR Mode (EMODE) and I/O Status Codes
(Part 10 of 10)

C.4. CSF$ EXECUTIVE REQUEST STATUS CODES

C.4.1. FACILITY REQUEST STATUS CODES (@CAT, @ASG, @FREE, @LOG, @MODE, @USE)

The meaning of the bits of the status code returned in register A0 for dynamic facility requests are described in Table C–1.

C.4.2. @SYM AND @BRKPT STATUS CODES

The status codes returned in register A0 for dynamic @SYM or @BRKPT control statements are given in Table C–3, with the equivalent diagnostic that would have been generated if the control statement had been submitted in the run stream I (see C.1 for explanation of the diagnostics).

Status Code (Octal)	Equivalent Diagnostic
100000	SECOND FILENAME NOT ASSIGNED, PUNCH$ NOT BREAKPOINTED
40000	SYM FILE IS NOT CATALOGUED OR CANNOT BE FOUND IN THE DIRECTORY
20000	PUNCH FILE CANNOT BE PUNCHED; NO PUNCH DEVICES CONFIGURED
10000	CANNOT SYM TEMPORARY FILE
4000	WARNING IMPROPER OPTION
1000	CANNOT SYM PU$ WHILE IN USER FILE
200	INVALID SYMBIONT NAME
100	NO FILE NAME
40	FILE ALREADY IN USE
20	SECOND FILE NAME IS IN ERROR
10	SECOND NAME IS ILLEGAL
4	L IS LEGAL FOR TAPE ONLY
2	OPERATION IS ILLEGAL FOR DEMAND
1	FIRST FILE NAME IS IN ERROR

Table C–3. @SYM and @BRKPT Status Codes

C.4.3. @ADD STATUS CODES

No status code is returned. If an error is discovered, the requesting activity is given error mode contingency type 12_8, error type 2 (see C.3).

Status Code (Octal)	Equivalent Diagnostic
10	ELEMENT ADD FROM TAPE NOT ALLOWED
11	NUMBER OF ADDS IN NEST EXCEED MAXIMUM ALLOWED
12	ADD FILE NOT ASSIGNED OR CATALOGUED
13	ADD ELEMENT NOT FOUND
14	ADD LOOP, FILE OR ELEMENT PREVIOUSLY ADDED IN THIS NEST
15	EQUIPMENT TYPE ERROR, ADD FILE
16	PCT FULL, CANNOT ASSIGN ADD FILE

C.4.4. @START DIAGNOSTICS AND STATUS CODES

On return from a CSF$ @START control statement, register A0 contains codes as follows:

Status Codes (Octal)	Description
0	REQUEST PROCESSED NORMALLY
1	REQUEST REJECTED DUE TO IMPROPER RUN STREAM IN FILE
2	REQUEST REJECTED DUE TO FILE UNOBTAINABLE
3	REQUEST REJECTED DUE TO ELEMENT OBTAINABLE
4	REQUEST REJECTED DUE TO FILENAME NOT SPECIFIED

C.4.5. CHECKPOINT/RESTART STATUS CODES (@CKPT, @CKPAR, @RSTRT, @RSPAR)

The meaning of the status codes returned in register A0 for dynamic checkpoint requests are described in Table C–6. No status is returned for a dynamic restart request.

C.5. MSCON$ AND PFP STATUS CODES

C.5.1. MSCON$ REQUEST STATUS CODES

When MSCON$ returns control to the user, register A0 contains status information. The format is:

35	34 30	29 24	23 18	17 0
S	executive indicator	I/O-error-indicator	status-code	packet-addr

The S bit is not set upon return for successful completion; if the S bit is set, the request was terminated in error. The possible status codes and their meaning are defined in Tables C–4 and C–5.

Status Codes (Octal)	Description
0	The requested function was completed normally
1	Special status returned for DREAD$ request. The end of the user buffer was encountered and more directory items are to be returned.

Table C–4. Status Codes for Successful Completion (S=0)

Status Codes (Octal)	Description
20	Wrong MSCON$ function code in user packet
21	User packet not within program limits
22	Referenced file is not assigned to this user
23	User is referencing a temporary file
24	The I/O error indicator (bits 29–24) contains the I/O status code received by MSCON$.
25	User buffer not within program limits
26	User is referencing a nonexistent start item (returned by the DREAD$ function)
27	User buffer area not large enough (returned by functions DLINK$, DADD$, and MSALL$)
30	The function requires a main item extension sector which is not in existence for this file (returned by functions DLAPS$ and DLINK$)
31	The referenced disc drive has been marked down or reserved (returned by the DGETP$ function)
32	The requested pack id cannot be found in the FASTRAND format availability table (returned by the DGETP$ function)
33	The output file initial reserve is too small to contain the current total of system directory items (returned by the DGET$ function)
34	The cumulative total of system directory items has dynamically expanded beyond the capacity of the output file (returned by the DGET$ function). This situation differs from that described above for status code 33_8, in that, in this instance DGET$ has been in process and directory items have been output to the file.

Table C–5. Status Codes for Error Termination (S=1)

C.5.2. PROGRAM FILE PACKAGE STATUS CODES

A status code is returned in register A2 upon return from the program file package executive requests (PFI$, PFD$, PFUWL$, and PFWL$). The possible values for the status code and their meanings are described below.

Status Code (Octal)	Description
0	Normal status (operation completed)
1	No find
2	I/O error
3	Program file not defined
5	Program file overflow

C.6. CHECKPOINT/RESTART ERROR CODES

Tables C–6 and C–7 list the error codes which are supplied when a checkpoint or restart is error terminated. For a dynamic checkpoint request (via CSF$), the error code is returned in register A0; no status is returned for dynamic restart requests.

When a checkpoint terminates in error, the following message is placed in the run log:

run-id CHECKPOINT ERROR TYPE ccc

where:

ccc — An error code as listed in Table C–6.

When a restart terminates in error, the following message is placed in the run log:

run-id RESTART ERROR TYPE ccc element-name nnn

where:

cc — An error code as listed in Table C–7.

element-name — Name of the element in which the error occurred.

nnn — Line number in the element.

Error Codes (Octal)	Description
1	Unrecoverable magnetic tape error
2	Unrecoverable FASTRAND-formatted mass storage error
3	The filename specified for the checkpoint is not assigned
4	The user is attempting to take a checkpoint on a facility other than magnetic tape or FASTRAND-formatted mass storage
5	Read or write is inhibited on the FASTRAND-formatted checkpoint file
6	Unused
7	The mode has been changed on the checkpoint tape, possibly making the restart impossible.
10	The PCT is at its maximum size and cannot be expanded.
11	The run is already being checkpointed.
12	A program is executing in real time (for keyin only), or ESI activities are present or the run is demand, or the run has reentrant processors attached.
13	Keyin format error
14	Program file not assigned or I/O error on program file
15	Checkpoint file is not large enough – 22_8 status was returned

Table C–6. Checkpoint Error Codes

Error Codes (Octal)	Description
1	Unrecoverable magnetic tape error
2	Unrecoverable FASTRAND-formatted mass storage error
3	The checkpoint could not be found on the specified file
4	A restart has been attempted on a facility other than magnetic tape or FASTRAND-formatted mass storage
5	The checkpoint file was not catalogued
6	A facility for the program to be restarted cannot be assigned
7	Unused
10	The checkpoint was not on the reel that was requested on the restart request
11	The checkpoint is in error and is incomplete
12	The next part of the checkpoint is not on the reel that was mounted
13	Keyin format error
14	Restart of this run is already in progress
15	Unused
16	Program file was not found

Table C–7. Restart Error Codes

C.7. BLOCK BUFFERING AND ITEM HANDLER ERROR CODES

C.7.1. DEVICE AND FILE EXIT CODES

For each occurrence of a device or file error, control is returned to the user program through the respective control words specified in the file control table (FCT). Upon such a return, an error status code is contained in H1 of register A0. The possible codes and their meanings are given in Table C–8.

Error Codes (Octal)	Description
1	Buffer pool link not provided
2	Request to close a file already closed.
3	Request to open a file already opened
4	Request to read or write a closed file
5	Request to write a block greater than maximum block size specified in FCT or a request to rewrite a block in in/out mode and block size requested to write is greater than the block size read.
6	FASTRAND variable block size specified on block read request larger than maximum block size, or reading variable blocks from fixed block file
7	Random request and file not assigned to FASTRAND mass storage
10	Random request and block size not fixed
11	Insufficient buffers in pool to satisfy LAF for input or output request
12	Invalid block number for random read request
13	Read request for a block greater than block size read.
14	File not assigned to FASTRAND mass storage for in/out mode processing
15	Random write request for input file
16	Random read request for output file
17	Read request with move-length parameter specified but no move-address specified
20	Read request for output file
21	Buffer size less than specified block size
23	Location of link or buffer area outside of user's assigned area
24	Block size not fixed for reverse mode for FASTRAND mass storage file
25	No I/O facilities assigned or improper equipment type
26	Write request in input file
27	Mark request for input file
64	Invalid mode parameter for open request

Table C–8. Device and File Exit Codes (Part 1 of 2)

Error Codes (Octal)	Description
The following error codes occur only when processing at the item level:	
30	Move address not given on in/out read or write request
31	Move address not given on read or write request of spanned item
32	Item size fixed on request to write a spanned item
33	Address of item same as move address given on in/out read or write request
34	Address of item same as move address given on random read or write request
35	Random request made in file without fixed items and blocks
36	Operation attempted on a closed file
37	Label specified but not found on open input request
40	Update flag not set on in/out file read request
41	Highest address written not set on in/out file write extend request
42	Drain requested on in/out file
43	A request was made to write a spanned item after reading it
44	A request was made to write more than the size read on in/out file
45	Read backwards requested on a spanned item
46	Format not set in FCT
47	Tape positioned improperly on open request
50	Frame count error detected and user did not specify to accept this as normal.
51	An EOF block was detected on a random read request
52	Fixed item larger than max block size

Table C–8. Device and File Exit Codes (Part 2 of 2)

C.7.2. ABNORMAL EXIT CODES

The occurrence of an abnormal condition causes control to be returned to the user's program by means of the abnormal exit control word in the PCT. When control is returned, a status code is contained in H1 of register A1. The abnormal status codes are:

Error Codes (Octal)	Description
1	End-of-file mark or load point has been detected for input tape file; or FASTRAND highest address (EOF) has been detected
2	End-of-tape mark has been detected for output file
4	Label word error has been detected in label block for an input file
6	Sentinel block has been detected for file
10	Block previously read exclusively has been timed out by the executive for an in/out file
12	Label block cannot be located, bad tape position
13	Block size specified in FCT is less than 14 words for an output file; or input file block size of sentinel is greater than block size specified in FCT
14	Item size specified in FCT not equal to item size in sentinel block for an input file

APPENDIX D. CONVERSION TABLES

D.1. INTRODUCTION

This Appendix provides a series of conversion tables as an aid to the programmer. The following conversion tables are printed:

Table D–1	Fieldata-to-ASCII
Table D–2	ASCII-to-Fieldata
Table D–3	UNISCOPE 100 Display Terminal Control Functions
Table D–4	Illegal Text Characters (UNISCOPE 100)
Table D–5	Cursor/SOE Coordinates (UNISCOPE 100)
Table D–6	XS-3-Fieldata-EBCDIC-BCD Conversion
Table D–7	Binary/Hexadecimal Conversion
Table D–8	Octal/Decimal Conversion

D.2. ASCII AND FIELDATA CONVERSION TABLES

Codes, which also represent collating sequence, are given in octal.

ASCII codes from 00_8 to 37_8 are for communication, format, and separator control characters. These are not converted into Fieldata.

The ASCII symbols represented by codes 40_8 to 137_8 are converted into the identical Fieldata symbols, except that the quotation marks symbol is converted into a lozenge, the circumflex is converted into a delta, and the underscore is converted into a not equal sign.

There are no remaining unique Fieldata symbols into which to convert the balance of the ASCII symbols, represented by codes 140_8 to 177_8, so these codes are 'folded' over codes 100_8 to 137_8 (by clearing bit 5, which amounts to subtracting 40_8). This means that ASCII codes 101_8 (A) and 141_8 (a), for example are both translated as if they were codes 101_8 (converted to Fieldata 06_8 for A). These 'folded' codes are shown boxed in below.

Although ASCII is presently a seven-bit code, it may eventually be extended to eight bits to allow for additional controls and special graphic characters, including possibly whole alternate alphabets. On a 36-bit machine, each ASCII code is stored within a 9-bit quarter word.

Fieldata		ASCII		Fieldata		ASCII	
Octal Code	Symbol	Octal Code	Symbol	Octal Code	Symbol	Octal Code	Symbol
00	@	100	@	40	)	51	)
01	[	133	[	41	-	55	-
02	]	135	]	42	+	53	+
03	#	43	#	43	<	74	<
04	Δ	136	∧	44	=	75	=
05	SP	40	SP	45	>	76	>
06	A	101	A	46	&	46	&
07	B	102	B	47	$	44	$
10	C	103	C	50	*	52	*
11	D	104	D	51	(	50	(
12	E	105	E	52	%	45	%
13	F	106	F	53	:	72	:
14	G	107	G	54	?	77	?
15	H	110	H	55	!	41	!
16	I	111	I	56	,	54	,
17	J	112	J	57	\	134	\
20	K	113	K	60	0	60	0
21	L	114	L	61	1	61	1
22	M	115	M	62	2	62	2
23	N	116	N	63	3	63	3
24	O	117	O	64	4	64	4
25	P	120	P	65	5	65	5
26	Q	121	Q	66	6	66	6
27	R	122	R	67	7	67	7
30	S	123	S	70	8	70	8
31	T	124	T	71	9	71	9
32	U	125	U	72	'	47	′
33	V	126	V	73	;	73	;
34	W	127	W	74	/	57	/
35	X	130	X	75	.	56	.
36	Y	131	Y	76	¤	42	"
37	Z	132	Z	77	≠	137	_

Table D–1. Fieldata-to-ASCII Conversion

ASCII		Fieldata		ASCII		Fieldata	
Octal Code	Symbol	Octal Code	Symbol	Octal Code	Symbol	Octal Code	Symbol
40	SP	05	SP	106	F	13	F
41	!	55	!	107	G	14	G
42	"	76	¤	110	H	15	H
43	#	03	#	111	I	16	I
44	$	47	$	112	J	17	J
45	%	52	%	113	K	20	K
46	&	46	&	114	L	21	L
47	'	72	'	115	M	22	M
50	(	51	(	116	N	23	N
51	)	40	)	117	O	24	O
52	*	50	*	120	P	25	P
53	+	42	+	121	Q	26	Q
54	,	56	,	122	R	27	R
55	– (minus)	41	– (minus)	123	S	30	S
56	.	75	.	124	T	31	T
57	/	74	/	125	U	32	U
60	0	60	0	126	V	33	V
61	1	61	1	127	W	34	W
62	2	62	2	130	X	35	X
63	3	63	3	131	Y	36	Y
64	4	64	4	132	Z	37	Z
65	5	65	5	133	[	01	[
66	6	66	6	134	\	57	\
67	7	67	7	135	]	02	]
70	8	70	8	136		04	Δ
71	9	71	9	137	_	77	≠
72	:	53	:	140	'	00	@
73	;	73	;	141	a*	06	A**
74	<	43	<	through	through	through	through
75	=	44	=	172	z	37	Z
76	>	45	>	173	{	01	[
77	?	54	?	174	'	57	\
100	@	00	@	175	}	02	]
101	A	06	A	176	~	04	Δ
102	B	07	B	177	DEL	77	≠
103	C	10	C				
104	D	11	D				
105	E	12	E				

*lowercase alphabet **uppercase alphabet

Table D–2. ASCII-to-Fieldata Conversation

D.2.1. THE SPECIAL CHARACTERS IN ASCII

SP designates space, which is normally nonprinting.

DEL designates delete, and has a code of all 1 bits.

This code obliterates any unwanted previous character — even on paper tape or other nonerasable medium.

The names of the 8 new special characters in ASCII are:		Some additional standardized names of interest:	
"	Quotation marks	#	Number sign
∧	Circumflex	&	Ampersand
_	Underline	'	Apostrophe
	Grave accent	*	Asterisk
{	Opening brace	>	Greater than sign
|	Vertical line	@	At sign
}	Closing brace	[	Opening bracket
	Tilde	\	Reverse slant

Definitions of the 32 ASCII control characters, codes 00_8 to 37_8:

00	NUL	Null — all zeros character which may serve as time fill	
01	SOH	Start of heading	
02	STX	Start of text	
03	ETX	End of text	
04	EOT	End of transmission	
05	ENQ	Enquiry — 'Who Are You?'	
06	ACK	Acknowledge — 'Yes'	
07	BEL	Bell — human attention required	
10	BS	Backspace	format effectors for printing or punching
11	HT	Horizontal tabulation	
12	LF	Line feed	
13	VT	Vertical tabulation	
14	FF	Form feed	
15	CR	Carriage return	
16	SO	Shift out — nonstandard code follows	
17	SI	Shift in — return to standard code	

20	DLE	Data link escape — change limited data communication controls	
21	DC1	Device controls for turning on or off ancillary devices	
22	DC2		
23	DC3		
24	DC4		
25	NAK	Negative acknowledge — 'No'	
26	SYN	Synchronous idle — from which to achieve synchronism	
27	ETB	End of transmission block — relates to physical communication blocks	
30	CAN	Cancel previous data	
31	EM	End of medium — end of used, or wanted portion of information	
32	SUB	Substitute character for one in error	
333	ESC	Escape — for code extension — change some character interpretations	
34	FS	File separator	These information separators are ordered in descending hierarchy. They are followed by ASCII 40_8 (space), which can also be thought of as a word separator.
35	GS	Group separator	
36	RS	Record separator	
37	US	Unit separator	

D.3. UNISCOPE 100 DISPLAY TERMINAL

Tables D–3 through D–5 provide the Fieldata equivalents of the ASCII representations of various UNISCOPE 100 functions and characters.

ASCII	Fieldata	Description
ESCa	≠;A≠	Erase to End of Display
ESCb	≠;B≠	Erase to End of Line
ESCc	≠;C≠	Delete in Line
ESCC	≠;≠C	Delete in Display
ESCD	≠;≠D	Insert in Display
ESCd	≠;D≠	Insert in Line
ESCe	≠;E≠	Cursor to Home
ESCf	≠;F≠	Scan Up
ESCg	≠;G≠	Scan Left
ESCh	≠;H≠	Scan Right
ESCi	≠;I≠	Scan Down
ESCj	≠;J≠	Insert Line
ESCk	≠;K≠	Delete Line
HT	≠)≠	Tab
BEL	≠'≠	Message Waiting
ESCHT	≠;)≠	Tab Stop Set
FS	≠<≠	Start Blink Field
DC4	≠4≠	Lock Keyboard
GS	≠≠	End Blink Field
DC2	≠2≠	Print
ESC DC2	≠;2≠	Print Transparent
DC1	≠1≠	Transmit
LF	≠*≠	Line Feed
FF	≠,≠	Form Feed
EM	≠9≠	End of Medium
RS	≠>≠	Start of Entry
CR	≠–≠	Carriage Return

Table D–3. UNISCOPE 100 Display Terminal Control Functions

ASCII		Fieldata	
Character	Value (Octal)	Character	Representation
SOH	001	-	≠!
ETX	003	-	≠#
EOT	004	-	≠$
DLE	020	-	≠0
*SYN	026	-	≠6

**Does not set illegal character flag, but is stripped from text.*

NOTE: If these characters are allowed within text, a message incomplete hardware failure is generated.

Table D–4. Illegal Text Characters

X/Y Coordinate	ANSCII		Fieldata			X/Y Coordinate	ANSCII		Fieldata	
	Character	Value (Octal)	Numerics	Value (Octal)			Character	Value (Octal)	Numerics	Value (Octal)
1		040	01	6061		41	H	110	41	6461
2	!	041	02	6062		42	I	111	42	6462
3	"	042	03	6063		43	J	112	43	6463
4	#	043	04	6064		44	K	113	44	6464
5	$	044	05	6065		45	L	114	45	6465
6	%	045	06	6066		46	M	115	46	6466
7	¢	046	07	6067		47	N	116	47	6467
8	'	047	08	6070		48	0	117	48	6470
9	(	050	09	6071		49	P	120	49	6471
10	)	051	10	6160		50	Q	121	50	6560
11	*	052	11	6161		51	R	122	51	6561
12	+	053	12	6162		52	S	123	52	6562
13	,	054	13	6163		53	T	124	53	6563
14	-	055	14	6164		54	U	125	54	6564
15	.	056	15	6165		55	V	126	55	6565
16	/	057	16	6166		56	W	127	56	6566
17	0	060	17	6167		57	X	130	57	6567
18	1	061	18	6170		58	Y	131	58	6570
19	2	062	19	6171		59	Z	132	59	6571
20	3	063	20	6260		60	[	133	60	6660
21	4	064	21	6261		61	\	134	61	6661
22	5	065	22	6262		62	]	135	62	6662
23	6	066	23	6263		63	∧	136	63	6663
24	7	067	24	6264		64	_	137	64	6664
25	8	070	25	6265		65	`	140	65	6665
26	9	071	26	6266		66	a	141	66	6666
27	:	072	27	6267		67	b	142	67	6667
28	;	073	28	6270		68	c	143	68	6670
29	<	074	29	6271		69	d	144	69	6671
30	=	075	30	6360		70	e	145	70	6760
31	>	076	31	6361		71	f	146	71	6761
32	?	077	32	6362		72	g	147	72	6762
33	@	100	33	6363		73	h	150	73	6763
34	A	101	34	6364		74	i	151	74	6764
35	B	102	35	6365		75	j	152	75	6765
36	C	103	36	6366		76	k	153	76	6766
37	D	104	37	6367		77	l	154	77	6767
38	E	105	38	6370		78	m	155	78	6770
39	F	106	39	6371		79	n	156	79	6771
40	G	107	40	6460		80	o	157	80	7060

Table D–5. Cursor/SOE Coordinates

D.4. CHARACTER CODES, XS-3, BCD CONVERSION TABLE

Table D–6 cross references printer symbols with the card punch, XS-3, Fieldata, EBCDIC, and BCD codes.

High Speed Printer Symbol	80-Column Card Code	XS-3	Fieldata	EBCDIC	BCD
A	12-1	24	06	C1	31
B	12-2	25	07	C2	32
C	12-3	26	10	C3	33
D	12-4	27	11	C4	34
E	12-5	30	12	C5	35
F	12-6	31	13	C6	36
G	12-7	32	14	C7	37
H	12-8	33	15	C8	38
I	12-9	34	16	C9	39
J	11-1	44	17	D1	21
K	11-2	45	20	D2	22
L	11-3	46	21	D3	23
M	11-4	47	22	D4	24
N	11-5	50	23	D5	25
O	11-6	51	24	D6	26
P	11-7	52	25	D7	27
Q	11-8	53	26	D8	28
R	11-9	54	27	D9	29
S	0-2	65	30	E2	12
T	0-3	66	31	E3	13
U	0-4	67	32	E4	14
V	0-5	70	33	E5	15
W	0-6	71	34	E6	16
X	0-7	72	35	E7	17
Y	0-8	73	36	E8	18
Z	0-9	74	37	E9	19
0	0	03	60	F0	0A
1	1	04	61	F1	01
2	2	05	62	F2	02
3	3	06	63	F3	03
4	4	07	64	F4	04
5	5	10	65	F5	05
6	6	11	66	F6	06
7	7	12	67	F7	07
8	8	13	70	F8	08

Table D–6. XS-3 Fieldata-EBCDIC-BCD Conversion Table (Part 1 of 2)

High Speed Printer Symbol	80–Column Card Code	XS-3	Fieldata	EBCDIC	BCD
9	9	14	71	F9	09
+	12	20	42	50	30
— (minus)	11	02	41	60	20
?	12-0	23	54	6F	
!	11-0	43	55	5A	
/	0-1	64	74	61	11
&	2-8	63	00	7A	30
=	3-8	35	44	7B	0B
'	0-3-8	56	72	7C	1B
:	5-8	21	53	7D	
>	6-8	76	45	7E	0E
@	7-8	40	01	7F	0C
.	12-3-8	22	75	4B	3B
)	12-4-8	75	40	4C	3C
[	12-5-8	17	57	4D	3D
<	12-6-8	36	43	4E	3E
#	12-7-8	37	03	4F	0B
$	11-3-8	42	47	5B	2B
*	11-4-8	41	50	5C	2C
]	11-5-8	01	76	5D	2D
;	11-6-8	16	73	5E	2E
Δ	11-7-8	57	52	5F	2F
≠	0-2-8	60	77	E0	
, (comma)	0-3-8	62	56	6B	1B
(	0-4-8	61	51	6C	1C
%	0-5-8	55	02	6D	1C
/	0-6-8	15	46	6E	11
¤	0-7-8	77	04	6F	3C
ƀ	Blank	00	05	40	10

NOTE: For 0758 printer, if XS-3 60 is specified, a blank occurs; if Fieldata 77 is specified, a blank occurs for that print position and all ensuing print positions for that line.

Table D–6. XS-3 Fieldata-EBCDIC-BCD Conversion Table (Part 2 of 2)

D.5. BINARY/HEXADECIMAL CONVERSION TABLE

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Table D-7. Binary/Hexadecimal Conversion

D.6. OCTAL/DECIMAL CONVERSION TABLE

The table of octal/decimal equivalents (Table D–8) provides a rapid means of converting from octal to decimal and vice versa. The range of the table is 0000 – 4095_{10} or 0000 – 7777_8.

To convert a decimal number greater than 4095 to its octal equivalent, reduce the number to 4095 or less by subtracting sufficient multiples of 4096. Convert this residue to octal by means of the table, and add 10000_8 for each multiple of 4096 which had been subtracted.

To convert an octal number greater than 7777 to the decimal equivalent, reduce the number to 7777 or less by subtracting sufficient multiples of 10000. Convert this residue to octal by means of the table, and add 4096_{10} for each multiple of 10000 which had been subtracted.

OCTAL 0000 to 0777	DECIMAL 0000 to 0511						
	0	1	2	3	4	5	6	7

	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

OCTAL 1000 to 1777	DECIMAL 0512 to 1023							
	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

Table D–8. Octal/Decimal Conversion (Part 1 of 4)

OCTAL 2000 to 2777 DECIMAL 1024 to 1535

	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1509	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

OCTAL 3000 to 3777 DECIMAL 1536 to 2047

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

Table D–8. Octal/Decimal Conversion (Part 2 of 4)

OCTAL 4000 to 4777 DECIMAL 2048 to 2559

	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	2537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

OCTAL 5000 to 5777 DECIMAL 2560 to 3071

	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

Table D–8. Octal/Decimal Conversion (Part 3 of 4)

OCTAL 6000 to 6777 DECIMAL 3072 to 3583

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

OCTAL 7000 to 7777 DECIMAL 3584 to 4095

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

Table D–8. Octal/Decimal Conversion (Part 4 of 4)

APPENDIX E. EQUIPMENT CODES

Equipment Code (octal)	Equipment Type
1	UNISERVO VIII-C Magnetic Tape Subsystem (seven-track)
2	UNISERVO VI-C Magnetic Tape Subsystem (seven-track)
3	UNISERVO VIII-C Magnetic Tape Subsystem (hardware translate)
4	UNISERVO VI-C Magnetic Tape Subsystem (hardware translate)
5	UNISERVO VIII-C Magnetic Tape Subsystem (nine-track)
6	UNISERVO VI-C Magnetic Tape Subsystem (nine-track)
7	UNISERVO IV-C Magnetic Tape Subsystem
10	UNISERVO IV-C Magnetic Tape Subsystem (hardware translate)
11	UNISERVO 12 Magnetic Tape Subsystem
12	UNISERVO 16 Magnetic Tape Subsystem
13	UNISERVO 12 Magnetic Tape Subsystem (nine-track)
14	UNISERVO 16 Magnetic Tape Subsystem (nine-track)
15	not used
16	UNISERVO III-A Magnetic Tape Subsystem
17	UNISERVO II-A Magnetic Tape Subsystem
20	FH-432 Magnetic Drum Subsystem
21	FH-880 Magnetic Drum Subsystem
22	FH–1782 Magnetic Drum Subsystem
23	not used

Equipment Code (octal)	Equipment Type
24	not used
25	Unitized Channel Storage
26	not used
27	not used
30	FASTRAND II or III Magnetic Drum Subsystem
31	not used
32	FASTRAND-formatted FH-432 Magnetic Drum Subsystem
33	FASTRAND-formatted FH-880 Magnetic Drum Subsystem
34	FASTRAND-formatted FH-1782 Magnetic Drum Subsystem
35	UNIVAC 8414 Disc Subsystem
36	UNIVAC 8440 Disc Subsystem
37	FASTRAND-formatted Unitized Channel Storage
40	Card Reader and Punch Subsystem
41	not used
42	UNIVAC 0920/0926 Paper Tape Subsystem
43	not used
44	UNIVAC 751 High Speed Printer Subsystem
45	UNIVAC 758 Multiple High Speed Printer Subsystem
46	not used
47	UNIVAC 9200/9300 Subsystem
50	UNIVAC 1004 Subsystem
51	not used
52	not used
53	not used
54	not used
55	not used
56	not used

Equipment Code (octal)	Equipment Type
57	not used
60	not used
61	not used
62	not used
63	not used
64	not used
65	not used
66	not used
67	not used
70	Communications Terminal Synchronous Subsystem
71	Word Terminal Synchronous Subsystem
72	Communications Terminal Module Controller Subsystem
73	not used
74	not used
75	not used
76	not used
77	arbitrary device

INDEX